UNIVERSITÉ DE MONTRÉAL

# AUTOMATIC GENERATION OF NC TOOL PATH PROGRAMS FOR MULTI-AXIS MOULD MANUFACTURING

par

Abbas VAFAEESEFAT

DÉPARTEMENT DE GÉNIE MÉCANIQUE

ÉCOLE POLYTECHNIQUE

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU GRADE DE MAÎTRE ÈS SCIENCES APPLIQUÉES (M.Sc.A.)

(GÉNIE MÉCANIQUE)

JANVIER 1994

ISBN 0-315-97129-0

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE

Ce mémoire intitulé:

AUTOMATIC GENERATION OF NC TOOL PATH PROGRAMS FOR MULTI-AXIS MOULD MANUFACTURING

Présenté par: Abbas VAFAEESEFAT

en vue de l'obtention du grade de : Maître ès Sciences Appliquées (M.Sc.A)

a été dûment accepté par le jury d'examen constitué de :

M. FORTIN Clément, Ph.D., président

M. BALAZINSKI Marek, Ph.D., membre et directeur de recherche

M. TROCHU François, Ph.D., membre et codirecteur

M. MAYER René, Ph.D., membre

## SUMMARY

The need for applying advanced CAD/CAM methods to manufacture sculptured surfaces arises in moulds, stamping dies, forging tools, and tooling shapes. Most moulds consist of a combination of complex curved surfaces that are difficult and expensive to produce. Moreover, the manual programming is a task very long and tedious and it is very difficult to control the tolerances of machined surfaces. The growing industrial demand motivated the development of computer-aided systems for the design and manufacture of those free-form surfaces. To address this problem, an algorithm for multi-axis NC tool path generation on sculptured surfaces is presented. The algorithm can be divided in 3 parts: surface creation, tool path generation, and post-processor.

The free-form surface is modeled parametrically by dual kriging. Kriging is simply the best linear unbiased estimator of a random function. A tool path is defined by a list of cartesian coordinates that indicate the successive positions of a tool for machining the surface. A series of profiles on the surface are machined via tool movements. To reduce tool wear and milling time, the algorithm is driven by the goal of minimizing the total number of tool motions while maintaining a specified overall milling tolerance.

Using the cartesian method, cutting curves are defined by the intersection of a series of parallel planes and the kriged surface. The next step is to select a series of points from the list of intersection points so that they provide the given tolerance. Moreover, because of the shape of the tool and surface, some material is left between two cutting planes called cusp height. Therefore, the distance between two cutting planes must be calculated to provide the given tolerance.

The interference between the surface and the tool is a critical problem in the tool path generation. This problem might occur because the curvature of the surface is less than the curvature of the tool, or because of the effect of tool shape especially with the ball end-mill. Direct gouge elimination is developed for this problem because it is precise and fast.

When machining the die cavities, rough cutting can be done using the zigzag or spiral methods. The surface can then be machined in 3-axis and 5-axis machining for semi-finishing and final finishing steps respectively.

The selection of the tool shape and size, and the choice of the direction of the tool axis are two important points in complex surface machining. Tool selection depends on the shape of the surface, the specified tolerance, the time of machining, etc, which are all steps that require good machining experience. The direction of the tool axis also depends on the given tolerance, the shape of the tool and surface, and the collision problem.

In the 4 and 5 axis machining cases, the gouging due to the rotation of tool axis and the collision between the tool and the surface are the critical problems. The direction of the tool axis must be determined such that the cusp height is minimum without any collision between the tool and  surface. To solve the gouging problem, the distance between two point is divided in to parts until it provides the given tolerance.

The rotational angles of each axis and the new coordinate of the points should be determined. This is done in post-processor and depends of the machine tool. There are four general types of machine tools and equations are obtained depending of the configuration of each machine.

To validate the algorithm, a complex surface and two turbine blades are machined in 3-axis mode. Moreover, a convex and a concave surface are machined in 5-axis. The results show that the algorithm can provide the given tolerances without any interference and collision between the tool and the surface. Further work will be required to study the effects of tool shape and the direction of tool axis to machined region, use of an expert system for selection of tool, simulate a workpiece and visualise the final shape and use an environment to simplify development work.

## ABSTRACT

An algorithm for multi-axis NC tool path generation on sculptured surfaces is presented where the free-form surface is modeled parametrically by dual kriging. This interpolation method offers several advantages since it permits to obtain piecewise linear and "spline" interpolations as particular cases and least square methods as a limit case. Non-constant parameter tool contact curves are defined on the part by intersecting parallel planes with the part model surface. Seven essential elements of this algorithm are introduced: creation of the kriged surface, tool path generation with direct gouge elimination, tool pass interval adjustment, collision avoidance, rough cutting process, simulation and post-processor. The whole system is implementable on a standard micro-computer.

## RÉSUMÉ

Plusieurs types de surfaces paramétriques sont couramment utilisées dans l'industrie. Les surfaces de Bézier, les splines et les B-Splines Rationnelles Non-Uniformes en constituent des exemples. Le krigeage dual paramétrique est une nouvelle méthode très puissante pour représenter des surfaces complexes. Les moules et étampes, les pales de turbines et les carrosseries d'automobile sont tous des objets typiquement modélisés avec des surfaces complexes.

La programmation manuelle d'une machine outil est une tâche longue et fastidieuse surtout pour usiner des surfaces complexes. Lorsque la programmation manuelle est possible, il est difficile de contrôler les tolérances et de vérifier les interférences possibles entre l'outil et la surface à usiner.

Pour résoudre le problème, un programme fondé sur le krigeage dual des surfaces paramétriques a été développé. Le programme calcule les trajectoires d'un outil à l'aide de la méthode des plans parallèles. La trajectoire de l'outil est la liste des coordonnées cartésiennes qui indiquent les positions successives de l'outil pour une séquence d'usinage de la surface. Le mouvement de l'outil sur la pièce brute découpe une série de profils de la surface permettant de la sculpter. Les positions de l'outil sont calculées à partir des coordonnées des points à usiner. Ces points proviennent de la géométrie de la surface krigée.

Deux méthodes peuvent être envisagées pour obtenir une suite de coordonnées d'usinage. La plus simple consiste à définir les points en faisant varier les paramètres *u* et *v* de la surface. La deuxième méthode est fondée sur

l'intersection d'une série de plans parallèles avec la surface paramétrique générée. La courbe générée par l'intersection de chaque plan avec la surface paramétrique peut être définie par un grand nombre de points dépendant de la résolution utilisée pour l'algorithme d'intersection. Si un grand nombre de points est envoyé à la machine outil, l'usinage prendra beaucoup de temps. Il faut donc bien choisir certains points sur la courbe d'intersection qui permettront d'usiner la surface avec une précision voulue.

Plusieurs méthodes ont été développées pour choisir ces points. La méthode employée calcule la longueur d'un vecteur projeté sur le plan définie par le point de départ et le point qui donne la déviation maximale. Ce deuxième point sera choisi de façon à ce que la longueur du vecteur projeté ne dépasse jamais la tolérance spécifiée par l'utilisateur. La méthode employée ici a l'avantage de fournir une très bonne approximation de l'erreur.

La distance entre deux plans doit être bien calculée pour donner un fini de surface tel que défini par la tolérance donnée. Cette distance représente la hauteur du sommet entre deux trajectoires d'outil (cusp en anglais). La distance entre les plans d'intersection, dépend du rayon de l'outil, du type d'outil (cylindrique, sphérique), de l'angle entre l'axe de l'outil et le vecteur normal de la surface, de la tolérance donnée et du rayon de courbure. Cette distance est calculée de façon à ce que l'erreur ne dépasse jamais la tolérance donnée pour chaque point sur la courbe.

Les points générés par l'intersection des plans parallèles définissent les points de contact entre l'outil et la surface. Il faut ensuite calculer, à partir de chaque point et de sa normale, la position relative de l'outil. Cette information sera ensuite transmise à la machine outil à commande numérique. Mais auparavant, il est important de vérifier que la trajectoire définie est libre d'interférences ou collisions.

Des problèmes d'interférence entre l'outil et la surface peuvent survenir lors de la sélection des points de la trajectoire. Premièrement, la trajectoire de l'outil prévue entre deux points est imprécise car le rayon de l'outil n'a pas été pris en considération. Par exemple, dans le cas d'une surface convexe, la distance réelle est plus grande que la distance calculée.

Le deuxième problème apparaît lorsque le rayon de courbure de la surface est plus petit que le rayon de l'outil. Ceci peut entraîner un usinage trop fort dans certaines zones. Pour éviter ce problème, plusieurs méthodes sont présentées. La méthode qu'on a développée consiste à vérifier à chaque point de tel sorte que la distance entre le centre de l'outil et la courbe soit toujours plus grand que la distance entre le centre de l'outil et la frontière du volume de l'outil. Dans le cas où une interférence est decelée, un point voisin est choisi, pour lequel il n'y aura pas d'interférence.

Dans le cas où le moule est une cavité, il faut d'abord faire un dégrossissage pour enlever les matériaux à l'intérieur du moule. Les méthodes "zigzag" et "spiral", qui sont faciles à interpoler avec la surface krigée, ont été développées. Ces méthodes sont fondées sur l'intersection d'un série de plan horizontaux et parallèles avec la surface paramétrique. La courbe générée par l'intersection de chaque plan sera définie par le krigeage. Les points à usiner ensuite peuvent être calculés par la méthode "zigzag" et "spiral".

Le choix de l'outil est un problème critique lors de l'usinage en mode 4 et 5 axes. Le type de l'outil doit correspondre avec la forme de la surface, la précision voulue, et le temps d'usinage. Il sera donc nécessaire d'avoir un système expert pour déterminer la forme de l'outil.

Avec les machines outil à 3 axes, il est possible d'usiner dans les trois directions d'un système de référence orthogonal. L'outil se déplace dans la direction de l'axe Z et la table se déplace dans les directions des axes X et Y mais il est impossible de changer l'orientation de l'outil. Le contrôle de l'orientation de l'axe de l'outil apporte une plus grande accessibilité à toutes les régions de la pièce et peut apporter une réduction du temps d'usinage.

Les machines outils à 5 axes ne sont pas limitées pour l'usinage comme le sont les machines à 3 axes. Ils ont deux degrés de liberté de plus, ce qui permet de contrôler l'orientation relative de l'outil. Sur ces machines, la table et l'outil peuvent prendre différentes orientations. Il est possible de tourner la pièce de façon à changer l'orientation de l'axe de l'outil par rapport à la pièce à usiner. Un autre avantage est de pouvoir usiner les régions d'une surface inaccessible en 3 axes.

Il existe plusieurs méthodes de choix d'orientation de l'axe d'outil en mode d'usinage en 5 axes. La plus simple consiste à orienter l'outil dans la direction de la normale de la surface, à condition qu'il n'y ait pas de collision entre l'outil et la surface. Il existe plusieurs méthodes pour éviter le problème de collision. La méthode développée prévient les collisions à chaque position en vérifiant une série de points qui parcourent la surface avec le volume de l'outil. Lorsqu'une collision est décelée, l'outil est pivoté graduellement de façon à prendre une position orthogonale.

L'autre problème lors de l'usinage en mode 4 ou 5 axes est l'interférence entre l'outil et la surface à cause de la rotation de l'axe de l'outil ou de la table de la machine. Ceci est appelé "linéairisation". Cette interférence sera augmentée, si on a plus de variation entre deux vecteurs successifs de l'axe de l'outil. Pour résoudre ce problème, des points intermédiaires seront ajoutés jusqu'à ce que la tolérance d'usinage soit satisfaite.

Les machines à 5 axes ont différentes configurations. Par exemple, les axes de rotation peuvent être autour des axes X et Y, X et Z, ou Y et Z. Pour usiner sur la machine outil à 5 axes, les coordonnées des positions relatives de l'outil et les vecteurs d'orientation des axes de l'outil sont nécessaires. Avec cette information, il est possible de calculer les rotations autour des deux axes. Les angles de rotation doivent êtres calculés par rapport au vecteur d'orientation de l'outil. Après ces rotations, la position relative de l'outil sera changée et devra être recalculée.

Le programme de simulation développé dans ce projet permet de visualiser les points à usiner et les positions successives de l'outil. Il permet aussi de visualiser le déplacement de l'outil durant l'usinage en mode 3 axes ou 5 axes. Ceci est très important car le temps de vérification sur une machine outil est très dispendieux. Une fois que la trajectoire de l'outil a été vérifiée avec le programme de simulation, elle peut être transférée à la machine outil pour l'usinage.

Le programme a été validé pour vérifier les objectifs du projet. Des surfaces krigées ont été utilisées:

- un moule d'une bouteille, une surface complexe, et deux pales de turbine pour usiner en 3 axes
- une surface concave et une surface convexe surface pour usiner en 5 axes
- une surface ondulée pour vérifier l'interférence et la collision.
- deux surface pour mesurer la précision de surface usinée.

Le logiciel développé permet de:

1. définir un trajectoire de l'outil sur surface krigée.
2. assurer un rugosité globale avec la précision donnée.
3. assurer un trajectoire qui évite les interférences.
4. programmer le dégrossissage, la semi-finition et la finition en 3, 4, 5 axes avec les outils cylindriques, sphériques, et toriques.
5. simuler la trajectoire pour visualiser les résultats et sortir directement le programme de machine-outil.
6. kriger un surface "offset" sans interférence et marquer les endroits où une reprise

d'usinage est requise.

Les prochains travaux tenteront de:

1. étudier les effets des différents outils et des différentes orientations de l'axe de l'outil, et des rotations de l'outil sur la région usinée.
2. utiliser un système expert pour le choix de l'outil.
3. simuler la trajectoire avec une pièce brute et visualiser la pièce finale.
4. utiliser un environnement pour faciliter le travail.

## ACKNOWLEDGEMENT

I would like to thank Marek BALAZINSKI and François TROCHU, my supervisor and co-supervisor, for their valuable comments which led to the improvement of this project. I would like also to thank my wife and my little daughter for their patience during my study.

# CONTENTS

## LIST OF FIGURES

## LIST OF APPENDIX

## INTRODUCTION

The need for applying advanced CAD/CAM methods to manufacture sculptured surfaces arises in moulds, stamping dies, forging tools, and tooling shapes. Most moulds consist of a combination of complex curved surfaces that are difficult and expensive to produce. The growing industrial demand motivated the development of computer-aided systems for the design and manufacture of those free-form surface cavities. Currently, sculpture surface modelling is one of the main fields of study in computer-aided geometric design and manufacturing. Many sophistical surface description methods have been developed for three dimensional sculptured surfaces. Generally, these methods provide users with powerful interactive modification or shape changing capability. The method based on dual kriging that is presented by F. Trochu [9] is used to generate the surface model in this paper.

With the development of NC technologies, multi-axis control NC machine tools have been recently introduced on the market to complement the 3-axis NC machines already in use. Multi-axis control permits to deal with workpieces of complicated shapes, which cannot be easily machined by conventional 3-axis NC machine tools. The use of 5-axis machining is likely to be increasingly accepted in the light of higher efficiency and accuracy. Especially for machining sculptured surfaces, multi-axis control tools permit saving on the machining time and polishing work required to achieve a good finish. Some well-established tool control methods have been introduced for multi-axis machining tool systems that are discussed in this paper [14, 15, 16].

An algorithm for multi-axis NC tool path generation is presented in this paper.

This algorithm is composed of the eight main modules that are listed below :

1) creation of the kriged surface model.

2) definition of the cutting plane for a given cusp height

3) intersection between the parallel planes and the surface

4) selection of points which give specified tolerance without interference with surface

5) determination of the tool axis vector ( in the case of 4 and 5 axis)

6) elimination of collisions

6) rough cutting (if necessary)

7) simulation

8) post-processing

First, the parametric equation of the sculptured surface is generated by dual kriging. Then, the tool pass interval adjustment technique calculates the maximum tool pass interval for a specified cusp-height tolerance. Intersection points are obtained using intersection between a plane and the surface. The initial chordal approximation method efficiently locates a good initial surface point to calculate the exact tool motion. Meanwhile, the true machining error is calculated for a concave or a convex surface. The direct gouge elimination technique works simultaneously with these procedures. The tool axis vector is determined to give a good finish. The collision between the tool and the surface is checked when machining in 4 and 5 axis mode. Then, the angle of rotation about the axes of the machine and three coordinate values of the tool center are calculated. To avoid errors that can cause damage to the machine tool, the machining process is graphically simulated. Finally, cutter locations are converted to the machine coordinate system.

## DEFINITIONS

The following terms (some of which are not new) are introduced to avoid ambiguities in later discussions.

Definition 1 (*CC* point, *CC* data, *CC* path, *CL* data, offset point)

We restrict ourselves to a toroidal tool but the equations may basically be adapted to other shapes. As shown in Figure 1, a point ***r****(u,v)* on the surface part 'touched' by the tool during machining is called a *CC* point (cutter contact point). Let ***n*** be the unit normal vector to the surface at ***r****(u,v)*, then the pair (***r***, ***n***) is called a *CC* data (cutter contact data). The tool center position is described by the function ***P***(*u,v*) defined as:

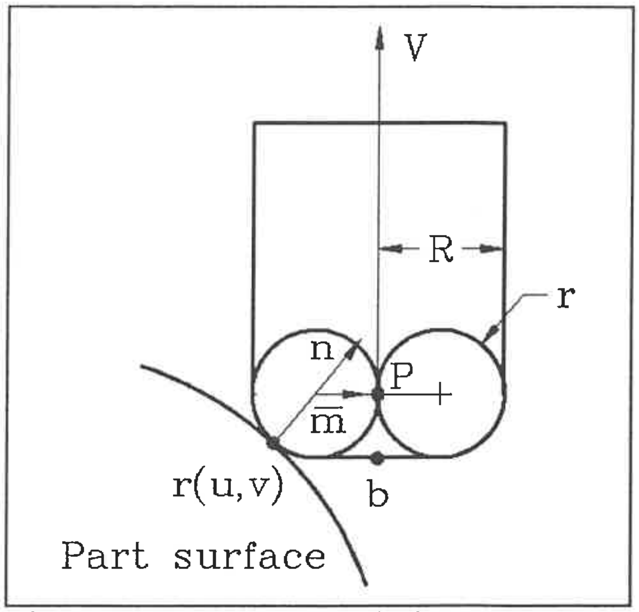


**Figure 1** A tool position

$$\boldsymbol{P}(u,v)=\boldsymbol{r}(u,v)+r\cdot\boldsymbol{n}(u,v)+(R-r)\cdot\boldsymbol{m}(u,v) \qquad \textbf{(1)}$$

where *R* is the tool radius, *r* is the radius of torus shape, and *u* and *v* are the surface parameters and the vector ***m****(u,v)* is defined as (see Figure 1):

$$\boldsymbol{m}(u,v)=\frac{\boldsymbol{n}(u,v)-\boldsymbol{V}\cdot<\boldsymbol{V},\boldsymbol{n}(u,v)>}{|\boldsymbol{n}(u,v)-\boldsymbol{V}\cdot<\boldsymbol{V},\boldsymbol{n}(u,v)>|} \tag{2}$$

where $\boldsymbol{V}$ is a unit vector representing the rotational axis of the cutter. In the case of a ball-mill, $R=r$, and the offset point is described by

$$\boldsymbol{P}(u,v)=\boldsymbol{r}(u,v)+r\cdot\boldsymbol{n}(u,v) \tag{3}$$

and in the case of end-mill r=0, the offset point is

$$\boldsymbol{P}(u,v)=\boldsymbol{r}(u,v)+R\cdot\boldsymbol{m}(u,v) \tag{4}$$

The bottom centre point $\boldsymbol{b}$ is the cutter reference point and is expressed as

$$\boldsymbol{b}(u,v)=\boldsymbol{P}(u,v)-r\cdot\boldsymbol{V} \tag{5}$$

and the pair $(\boldsymbol{b},\boldsymbol{V})$ is called a *CL* data (cutter location data). For a 3-axis NC machine, $\boldsymbol{V}=(0, 0, 1)$. A sequence of line segments obtained by connecting *CC* points sequentially (cutter touched) is called a *CC* path.

Definition 2 (convex, concave, and inflection relations)

Let $(\boldsymbol{r}_i, \boldsymbol{n}_i)$ and $(\boldsymbol{r}_j, \boldsymbol{n}_j)$ be two adjacent *CC* data, then unidirectional relations are defined as (see Figure 2):

CC point $\boldsymbol{r}_i$ is convex to $\boldsymbol{r}_j$ if $(\boldsymbol{r}_j - \boldsymbol{r}_i)\cdot\boldsymbol{n}_i < 0$

CC point $\boldsymbol{r}_i$ is concave to $\boldsymbol{r}_j$ if $(\boldsymbol{r}_j - \boldsymbol{r}_i)\cdot\boldsymbol{n}_i > 0$

CC point $\boldsymbol{r}_i$ is parallel to $\boldsymbol{r}_j$ if $(\boldsymbol{r}_j - \boldsymbol{r}_i)\cdot\boldsymbol{n}_i = 0$

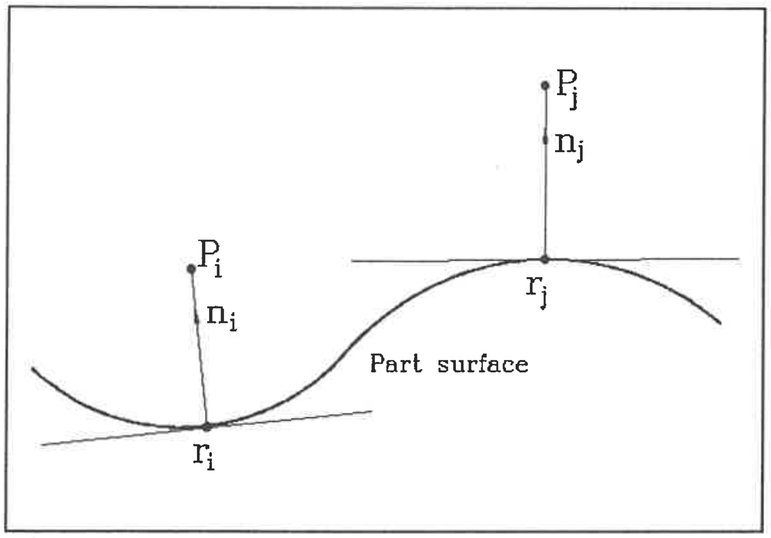


**Figure 2** Relation between the adjacent points (inflection)

Now mutual relations between two adjacent *CC* points are defined. Two *CC* points, $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$ , are said to be in:

- convex relation if they are convex to each other
- concave relation if they are concave to each other
- parallel relation if they are parallel to each other
- inflection relation if one is convex to the other and the other is concave to the first.

If one *CC* point is convex (concave) to the other and the other is parallel to the first, they are said to be in a convex (concave) relation. If the *CC* point $\boldsymbol{r}$ is replaced by its offset point $\boldsymbol{P}$ in the above definitions, the same relations between two adjacent offset points, $\boldsymbol{P}_i$ and $\boldsymbol{P}_j$ , can be defined. For example offset point $\boldsymbol{P}_i$ is convex to $\boldsymbol{p}_j$

if $(\boldsymbol{P}_j - \boldsymbol{P}_i) \cdot \boldsymbol{n}_i < 0$ and if $\boldsymbol{P}_j$ is also convex to $\boldsymbol{P}_i$ , they are in a (mutual) convex relation [1].

Definition 3 (sculptured surfaces)

In Numerical Control Machining, we come across two types of dimensional surfaces, the analytical surfaces and the other non-analytical surfaces. A surface that can be defined in (x, y, z) by a single equation is termed analytic surface. For these surfaces, calculation of surface slopes, normals and curvatures is straightforward. Every point on the analytic surface is precisely defined, so interpolation is not necessary and the surfaces can be evaluated using analytic geometry.

Non analytic surfaces are defined through a grid of points in space and cannot be defined by a single mathematical relationship. Non-analytic surfaces are otherwise called sculptured or free form surfaces. The geometry of many surfaces such as automobile bodies, aircraft structures, ship hulls, turbine blades, impellers and many other industrial products fall into the category of sculptured surfaces.

# CHAPTER 1
# SURFACE DESCRIPTION

A major area of interest in integrated CAD/CAM is the design and manufacture of sculptured surfaces. This has always presented difficulties for manufacturing engineers. During the design phase , these surfaces have to fit a given set of data points, and the whole surface should be smooth and continuous. In order to produce such free form surfaces, various dies and templates with different model curves are usually needed. Nowadays these dies and templates are still made manually or by electrical discharge machining in many factories. A large number of curved templates are required for prototype testing. Although the process is long and consumes much material and labour, it gives surfaces of relatively low quality.

Many sophisticated surface description methods have been developed for three dimensional sculptured surfaces. Generally these methods provide users with powerful interactive shape modification chaining capabilities. The method of free form surface employed here is based on dual kriging.

## 1.1 Kriging interpolation

Kriging is a statistical technique proposed in 1951 by Dr. Krige [3] for natural resource evaluations. Later, Matheron [4] established the mathematical foundation of the method. As presented in the mathematical framework of geostatistics, kriging is simply the best linear unbiased estimator of a random function. In this section, kriging will be briefly presented. First, the philosophy of method is exposed. Then the basic equations of dual kriging interpolation are stated. Note that a complete derivation of the basic kriging equations and the connection with dual kriging can be found in the literature [9]. For the sake of brevity, the method is presented with X denoting the position vector X = x, X = (x, y) or X = (x, y, z) for the 1D, 2D and 3D cases, respectively.

### 1.1.1 Philosophy of kriging

The purpose of kriging is to estimate the value of a random function U(X) at a specified location X = (x, y, z), given a set of measurements or samples $U_i$ taken at positions $X_i$, for $1 \le i \le N$. The simplest estimation of U(X) can be obtained as a linear combination of the data available [9]

$$u(X)=\sum_{i=1}^{N} \lambda_i U_i \tag{6}$$

This means that u(X) is evaluated by a linear combination of the random variables $u(X_i)$ at the observation points $X_i$ , $1 \le i \le N$

$$U^*(x)=\sum_{i=1}^{N} \lambda_i U(X_i) \tag{7}$$

The set of weights $\lambda_i$ will be determined so as to minimize the squared variance of the estimation error $E\{[U(X) - U^*(X)]^2\}$, where E [U(X)] denotes the mathematical expectation of a random function, provided that the estimation based on formula (7) is without bias. This last condition simply means that the expected values of U(X) and U*(X) must be identical, i.e., $E[U(X)] = E[U^*(X)]$.

Kriging permits the construction of an approximate function u(X) that fits the data points, i.e. such that

$$U(X_i)=U_i \qquad 1\leq i\leq N \tag{8}$$

In kriging, the random function u(X) is decomposed into the sum of two terms

$$U(X)=a(X)+b(X) \tag{9}$$

where a(X) = E [U(X)] is called the *drift*. The second term b(X) represents a stationary *fluctuation*, i.e., such that E[b(X)]=0. The drift can be represented by polynomials of degree k (k integer ≥ 0) or even by trigonometric functions. The drift is said to belong to a linear subspace S spanned by M basis functions $\boldsymbol{P}_l$ (X), 1 ≤ l ≤ M

$$a(x)=\sum_{l=1}^{M} a_l P_l(X) \tag{10}$$

The choice of the drift is arbitrary. It represents the average behaviour of the physical phenomenon. The more closely the drift follows the actual phenomenon, the better becomes the kriging interpolator. The correction term b(X) is automatically adjusted in kriging so that the interpolation model matches the data points.

### 1.1.2 Dual kriging interpolation

The fluctuation b(X) depends on the observation points $X_i$. The greatest is the distance between X and $X_j$, the less should be the effect of sample j on the interpolated value u(X). Hence it is reasonable to assume here that the correction weights associated with the data points depend only on the euclidean distance $| X - X_j |$, so the fluctuation term can be written as follows [9]:

$$b(x)=\sum_{j=1}^{N} b_j K(|X-X_j|) \tag{11}$$

In kriging, K(h) is called the *generalized covariance*, such as for example K(h)=h, $h^2$ ln h, $h^3$. The basic dual kriging model can finally be written as

$$U(X)=\sum_{l=1}^{M} a_l P_l(X)+\sum_{j=1}^{N} b_j K(|X-X_j|) \tag{12}$$

Depending on the properties of K(h), the interpolation will be continuous or even differentiable and, by construction, always matches the data points. All these properties are interesting, especially for 2D or 3D interpolations, because a great number of physical phenomena are better represented by smooth functions.

The M + N degrees of freedom $a_l$, $1 \leq l \leq M$, and $b_j$, $1 \leq j \leq N$, of the dual kriging model (12) are determined by a system of M + N linear equations called the kriging system. A first set of N equations is readily obtained by specifying that the interpolation matches the data points, but M additional equations are necessary to identify the unknown coefficients in (9). These supplementary equations are called the no-bias conditions. They consist in adding equations on the drift coefficients so that a symmetric system of N + M linear equations is obtained (see reference [9]):

$$\left[\begin{array}{ccc|c} \cdots & \cdots & \cdots & \cdots \\ \vdots & K_{ij} & \vdots & P_l(X_i) \\ \vdots & \cdots & \vdots & \cdots \\ \hline \cdots & \cdots & \cdots & \cdots \\ \vdots & P_l(X_j) & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \end{array}\right] \cdot \begin{bmatrix} b_1 \\ \vdots \\ b_N \\ - \\ a_1 \\ \vdots \\ a_M \end{bmatrix} = \begin{bmatrix} u_1 \\ \vdots \\ u_N \\ - \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \textbf{(13)}$$

The function K(h) is always chosen such that matrix [K] of general term $k_{ij} = K(\,|X_i - X_j|\,)$ is positive definite. The solution of system (13) yields the coefficients of the interpolation model, hence permitting the evaluation of interpolated values by (12) anywhere in the geometric domain. For example, the kriging equation in three dimensions for a linear drift can be written as follows :

$$u(x,y)=a_1+a_2x+a_3y+a_4z+\sum_{j=1}^{N} b_jK(h_j) \qquad \textbf{(14)}$$

where $h_j$ denotes the euclidean distance between the control point (x,y) and sample $(x_j, y_j)$

$$h_j=\sqrt{(x-x_j)^2+(y-y_j)^2+(z-z_j)^2} \qquad \textbf{(15)}$$

## 1.2 Parametric curve modelling

The geometric model is defined in the three-dimensional space of cartesian coordinates $x$, $y$ and $z$. The parametric equation of a curve is defined by three functions x(t), y(t) and z(t). Dual kriging permits to construct automatically the equations of smooth parametric curves. For example, in the case of a linear drift and cubic covariance, the parametric equations of kriged curves can be written as follows:

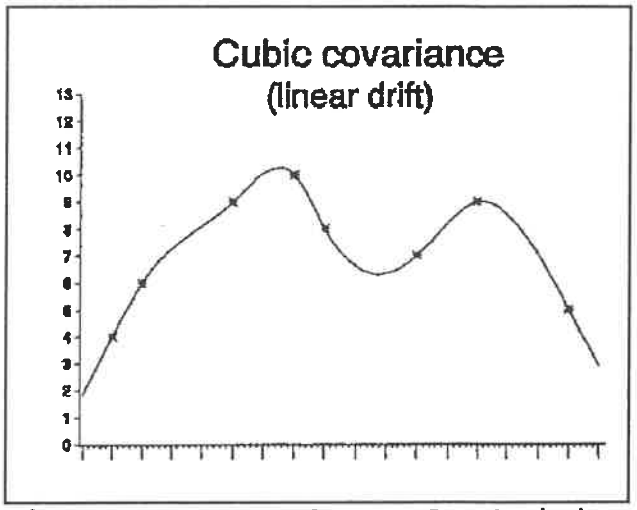


**Figure 3** Example of kriging interpolation

$$x(t)=a_1+a_2x+\sum_{j=1}^{N} b_j|t-t_j|^3,\ y(t)=\cdots,\ z(t)=\cdots \tag{16}$$

where the parameters $t_j$, $1 \le j \le n$ , denote an approximation of the curve length calculated from $t_0 = 0$ by

$$t_{i+1}=t_i+[(x_{i+1}-x_i)^2+(y_{i+1}-y_i)^2+(z_{i+1}-z_i)^2]^{\frac{1}{2}},\ 1\le i\le N-1 \tag{17}$$

The coefficients $a_1$, $a_2$ and $b_j$ are obtained by requiring that the interpolation model (16) fits the data points (see Figure 3)

$$x(t_i)=x_i \tag{18}$$

$$y(t_i)=y_i \tag{19}$$

$$z(t_i)=z_i \tag{20}$$

for $1 \le i \le N$ and (2) by adding the no-bias conditions as in (13).

## 1.3 Parametric description of surfaces

If a deformable curve **r** = **r**(s) is moving in a three dimensional space (see Figure 4), the successive positions of the curve generate a surface, each point of which being identified by its parameters on the moving curve and at time *t*. Thus a parametric equation of the form **r** = **r**(u,v) describes a three-dimensional surface, with the component functions *x = x(u,v), y = y(u,v), z = z(u,v)*. A surface consists of two profiles (i.e. *A* and *B*), one is in *v* and other in *u* direction. Each profile makes up a set of curves that moves in the space. The parametric equation of each curve can be defined by equation (16). The parametric equations of a curve of profile *A* in the direction of *v* with a linear drift and cubic covariance can be

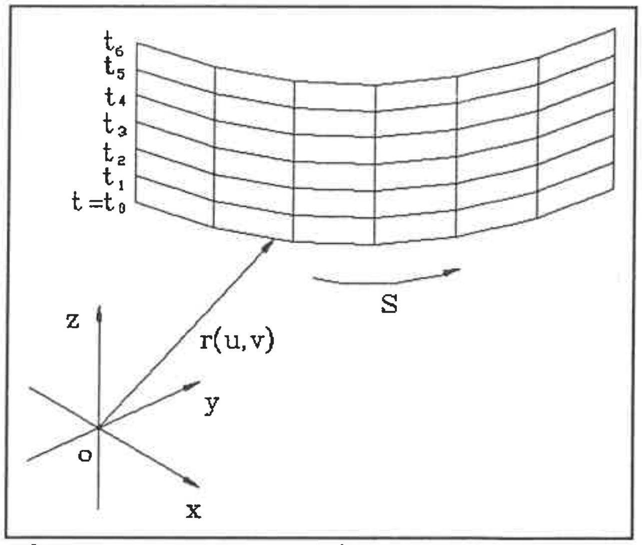


**Figure 4** Parametric surface

written as follows:

$$x_t(s)=a_1+a_2s+\sum_{l=1}^{I} b_l|s-s_l|^3,\ y_t(s)=\cdots,\ z_t(s)=\cdots \tag{21}$$

or

$$\begin{bmatrix} \cdots & \cdots & \cdots & | & 1 & s_1 \\ \vdots & K(|s_i-s_l|^3) & \vdots & | & 1 & s_i \\ \vdots & \cdots & \vdots & | & 1 & s_I \\ --- & --- & --- & - & - & - \\ 1 & 1 & 1 & | & 0 & 0 \\ s_1 & s_i & s_I & | & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_i \\ b_I \\ - \\ a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} x_{11} & x_{1j} & x_{1J} \\ x_{iI} & x_{ij} & x_{iJ} \\ x_{II} & x_{Ij} & x_{IJ} \\ - & - & - \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (1\le i\le I)\ ,\quad (1\le j\le J) \tag{22}$$

The equation (22) can be written as

$$[K_p]\,.[b]=X_{ij} \tag{23}$$

where [b] = $\{b_1 \ldots b_i \ldots b_I\ a_0\ a_1\}^T$

Solving for [b] gives

$$[b]=[K_p]^{-1}\bullet X_{ij} \tag{24}$$

and substitution from (22)

$$[x_{t_j}(s)]^T=[\cdots K(|s-s_l|^3)\cdots 1\ s\,]\,[K_p]^{-1}\begin{bmatrix} x_{ij} \\ - \\ \cdots 0 \cdots \\ \cdots 0 \cdots \end{bmatrix},\ [y_{t_j}(s)]^T=\cdots,\ [z_{t_j}(s)]^T=\cdots \tag{25}$$

The parametric equation of a curve for a profile *B* with a linear drift and cubic covariance can be written as follows:

$$x'_s(t)=a_1+a_2t+\sum_{l=1}^{I} b_l|t-t_l|^3,\ y'_s(t)=\cdots,\ z'_s(t)=\cdots \tag{26}$$

or

$$\left[\begin{array}{ccc|cc} \cdots & \cdots & \cdots & 1 & t_1 \\ \vdots & K(|t_k-t_j|^3) & \vdots & 1 & t_j \\ \vdots & \cdots & \vdots & 1 & t_J \\ \hline 1 & 1 & 1 & 0 & 0 \\ t_1 & t_j & t_J & 0 & 0 \end{array}\right] \cdot \left[\begin{array}{c} B_1 \\ B_j \\ B_J \\ \hline A_0 \\ A_1 \end{array}\right] = \left[\begin{array}{ccc} x_{11} & x_{i1} & x_{I1} \\ x_{1j} & x_{ij} & x_{Ij} \\ x_{1J} & x_{iJ} & x_{IJ} \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right] (1\le i\le I),\ (1\le j\le J) \tag{27}$$

then

$$[x'_{s_i}(t)]^T=[\cdots K(|t-t_k|^3)\cdots 1\ t\,][K_Q]^{-1}\left[\begin{array}{c} x_{ij}^{\ T} \\ \hline \cdots 0 \cdots \\ \cdots 0 \cdots \end{array}\right],\ [y'_{s_i}(t)]^T=\cdots,\ [z'_{s_i}(t)]^T=\cdots \tag{28}$$

Considering the equations (25) and (28), we can write

$$x(s,t) = [\cdots K(|s-s_l|^3)\cdots 1\ s\ ]\ [K_P]^{-1} \left[\begin{array}{c|cc} & 0 & 0 \\ x_{ij} & \vdots & \vdots \\ & 0 & 0 \\ \hline 0 \ldots 0 & 0 & 0 \\ 0 \ldots 0 & 0 & 0 \end{array}\right] [K_Q]^{-1} \left[\begin{array}{c} \vdots \\ K(|t-t_k|^3) \\ \vdots \\ \hline 1 \\ t \end{array}\right],\ y(s,t)=\cdots,\ z(s,t)= \tag{29}$$

It can be verified that

$x(s_i,t_j) = P_{tj}(s_i) = x_{ij}$, for $1 \le i \le I$ and $1 \le j \le J$

Generally, dual kriging permits to construct the smooth and differentiable equation of a parametric surface as follows:

$$P(s,t) = [\cdots\ K(|s-s_l|^3)\ \cdots\ ,\ P_l(s)\ \cdots]\ [K_s(h)]^{-1} \left[\begin{array}{c|c} P_{ij} & 0\cdots0 \\ - \quad - & - \\ 0\cdots0 & 0\cdots0 \end{array}\right] [K_t(h)]^{-1} \begin{bmatrix} \vdots \\ k(|t-t_j|^3) \\ - \\ P_1(t) \\ \vdots \end{bmatrix} \tag{30}$$

where $P_{ij}(x_{ij}\ ,\ y_{ij}\ ,\ z_{ij})$, for $1 \le i \le I$, $1 \le j \le J$, are a set of ordered data points that define the surface

$$x(s_i,t_i)=x_{ij} \tag{31}$$

$$y(s_i,t_i)=y_{ij} \tag{32}$$

$$z(s_i,t_i)=z_{ij} \tag{33}$$

Dual kriging provides a general framework which incorporates several interpolation techniques. For example when $K_s(h) = K_t(h) = h$, a piecewise linear surface model is obtained. It can be shown also that kriging with $K_s(h) = K_t(h) = h^3$ and linear drift is equivalent to a bicubic spline interpolation. The shape function $K_s(h) = K_t(h) = h^2 \ln h$ with a linear drift corresponds to a two-dimensional spline. By changing $K_s$ and $K_t$ , a great variety of shapes can be represented in this general framework. The method permits to represent simple and complex surfaces. For example, Figure 5 shows a surface defined by four curves, and in Figure 6, you can see a surface defined by the rotation of a curve around the x axis [7].

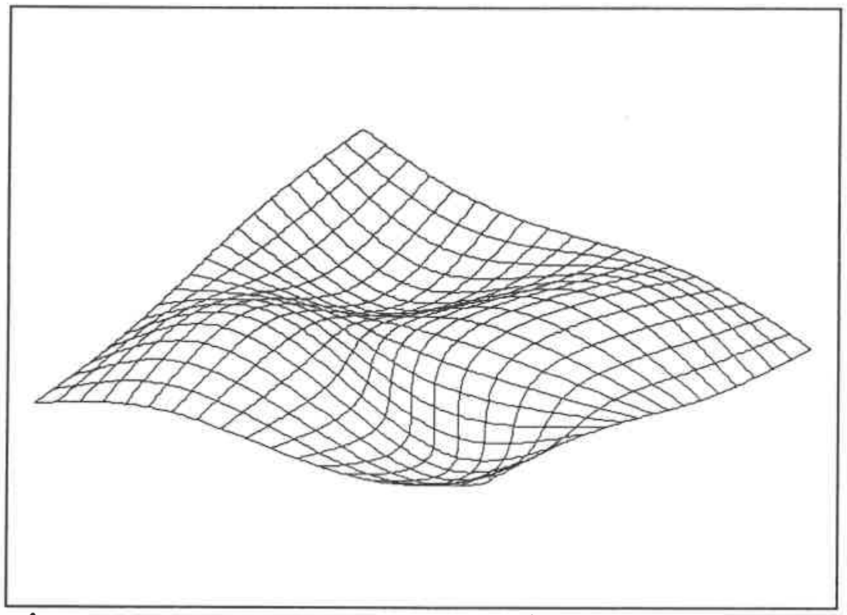


**Figure 5** An example of a kriged surface

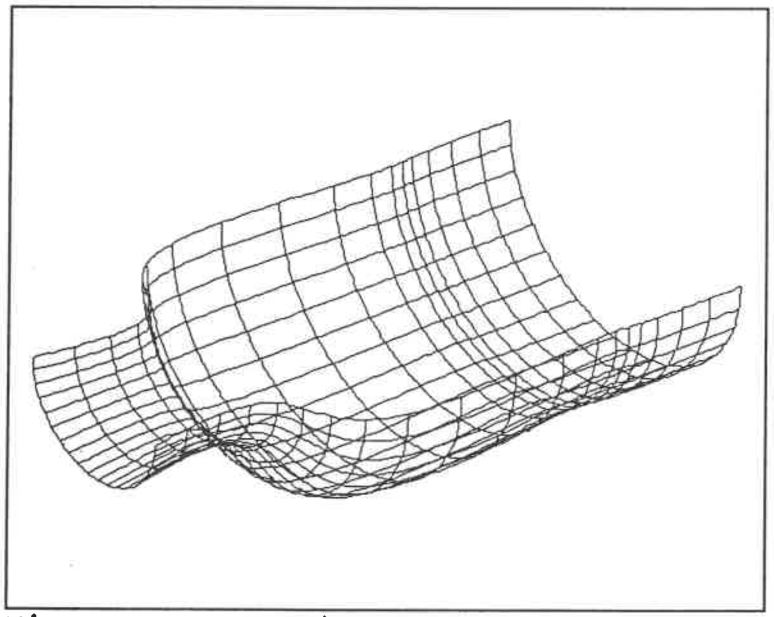


**Figure 6** A rotational surface

# CHAPTER 2

# TOOL PATH GENERATION

A tool path is a list of cartesian coordinates that indicate the successive positions of the tool for machining a surface. A series of profiles on the surface are machined via tool movements . The tool positions are calculated from the selected points obtained from the kriged surface geometry. To reduce tool wear and milling time, NC generation algorithms for parametric sculptured surfaces are generally driven by the goal of minimizing the total number of tool motions while maintaining a specified overall milling tolerance without any interference between the surface and the tool. It is possible to follow a sculptured surface exactly. However, such a process would be very computation intensive and would not be cutting under "optimum" conditions.

In the cases of 4 and 5 axis machining, the direction of the tool axis is a critical problem because this direction must be determined in such a way that the cusp height will be minimized without any collision between the tool and the surface.

Various NC tool path generation algorithms have been described in the literature, and many commercial CAD/CAM systems provide such capabilities. In this chapter, the different methods that are employed in each step as well as the advantages and the disadvantages of each method are discussed, and some new methods are also proposed.

## 2.1 Tool path planning methods

There are many approaches to tool path planning. Two general methods are presented in this section.

### 2.1.1 Parametric machining method

Much of the current research focuses on the constant parameter approach for locating tool contact curves on the surfaces of the part. This approach is generally efficient because the tool contact curves are easy to retrieve from the surface definitions. The drawback of this approach, however, is that the relationship between the parametric coordinates and the corresponding physical (Cartesian) coordinates is not uniform. Therefore, the accuracy and efficiency of the constant parameter approach for tool path generation may vary depending on the surface geometry. A typical example of this drawback is the "fan-shaped" surface shown in Figure 7. On such a surface constant parameter curves are close to one another at one end, but much further apart at the other. Of course, the curves could be generated with a bigger density

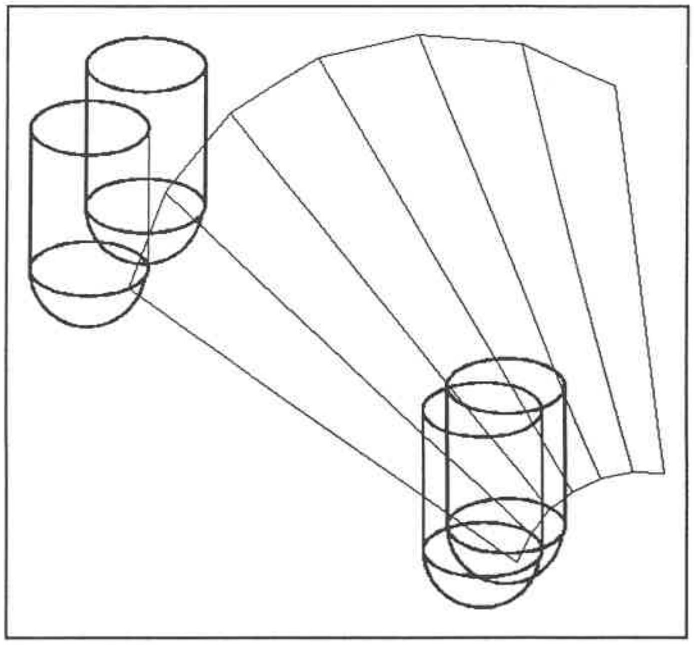


**Figure 7** The tool moves along constant parameter curves on a "fan-shaped" surface

at the wide end of the fan, but this would result in an unnecessarily large number of tool motions at the narrow end [2].

### 2.1.2 Cartesian machining method

The non-constant parameter approach for NC tool path generation does not suffer from the problem of method 1. Typically, cutting curves are defined by the intersections of a set of parallel planes (cutting planes) with freeform surface, as shown in Figure 8. Users can define any tooling direction for each part without being limited by the surface parametrization.

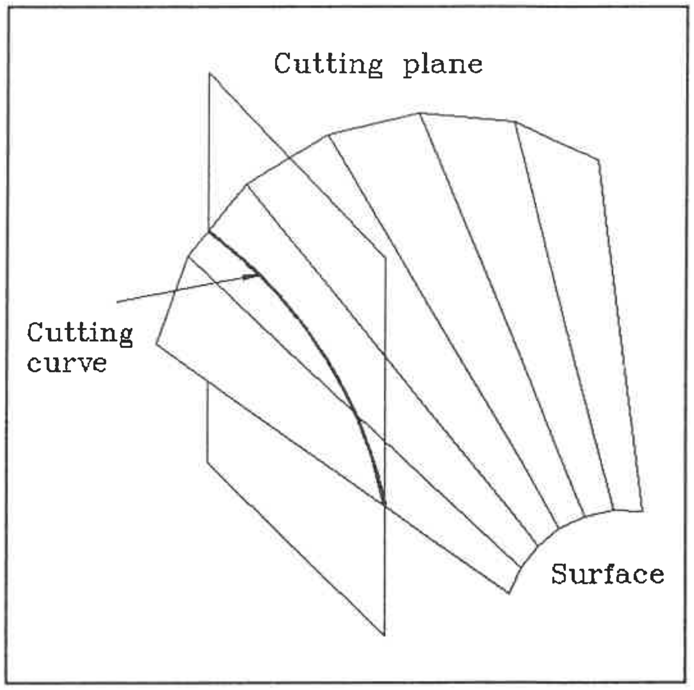


**Figure 8** Cutting plane and cutting curve definitions

## 2.2 Selection of CC points

Normal machining errors in NC milling operation is due to the approximation of surface curves by linear tool motions. Many efficient path generation methods, which bridge between the surface description and tool control methods, have been proposed for multi-axis computer numerical control systems. Figure 9 depicts two points that lie on a curve of a sculptured surface and two unit normal vectors $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$. The normal vectors of two points are projected on the cutting planes. If the curve is to be approximated by a line segment (chord) from point $\boldsymbol{r}_1$ to $\boldsymbol{r}_2$, the chordal deviation is defined as the maximum distance from the curve to the chord. Such a deviation defined by the curve and the chord between two successive cutter contact (*CC*) points is referred to as the *nominal chordal deviation*. Two methods to calculate the nominal chordal deviation are compared in this section.

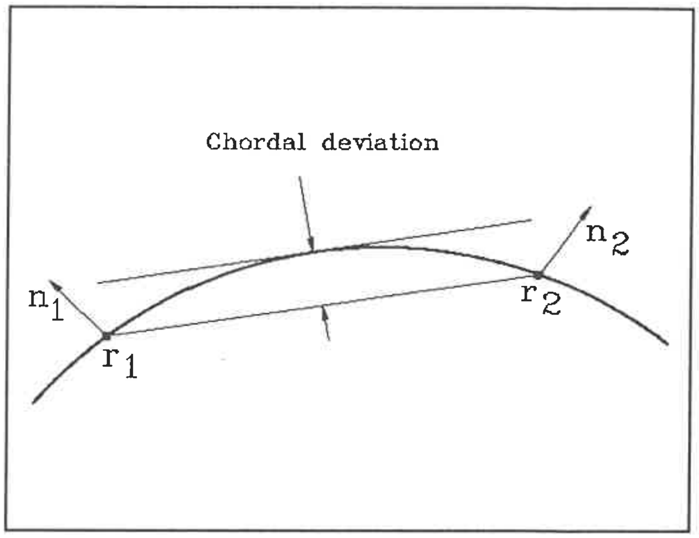


**Figure 9** Linear curve approximation

**< Method 1 >**

The simplest method to approximate the nominal deviation is to assume that the furthest point of the curve from the chord is defined parametrically by half of the total parametric variation. In this method, the parametric curves on a three-dimensional sculptured surface are approximated by a sequence of linear segments. The tool paths for these segments are calculated for ball-type cutters. The method optimizes cutter movements by minimizing the number of intermediate points while the approximated curves stay within a specified tolerance.

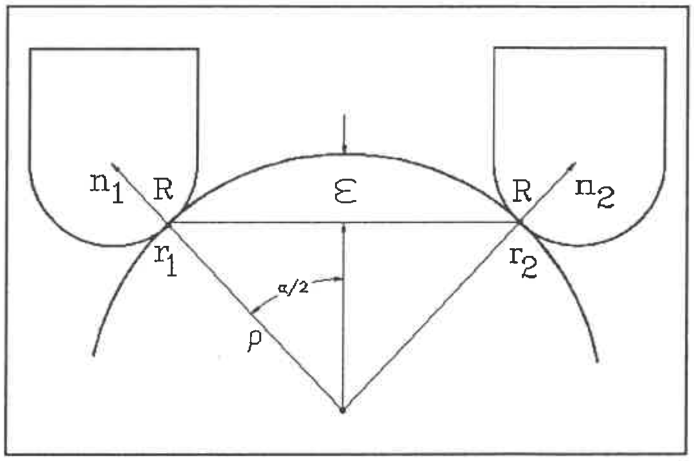


**Figure 10** Estimation of cutting error

Figure 10 shows the geometric relationships between the radius $\rho$ of the circular path, the unit normal vectors $\boldsymbol{n_1}$ , $\boldsymbol{n_2}$ , the cutter radius R, the angular displacement $\alpha$ between two consecutive *CC*-points, and the overcutting error $\epsilon$. For a convex segment as in Figure 10, the cutting error is expressed as

$$\epsilon=\rho(1-\cos(\frac{\alpha}{2})) \qquad \textbf{(34)}$$

where

$$\alpha=\cos^{-1}(\boldsymbol{n}_1\bullet\boldsymbol{n}_2) \qquad \textbf{(35)}$$

The procedure for determining the step length is to estimate the cutting error using equation (34) and (35) for each curve defined by a given cutting plane. If the estimated error is different from the specified tolerance,$\tau$, i.e., $|\epsilon\text{-}\tau|<e$ where $e$ is of small magnitude, then a midpoint is inserted and this procedure is executed recursively.

This technique is generally sufficient for surfaces with a uniform parametric variation, but if the underlying surface definition is characterized by a non-uniform parametric variation, the error of this approach may become significant.

**< Method 2 >**

This method first searches for the location of the climax that yields the specified deviation, then it calculates the end point of the chord. In addition, this method takes advantage of the planar geometry used for the nonconstant parameter NC tool path generation .

As shown in Figure 11, the starting point for the initial chordal approximation is denoted by $r_1$, the nominal chordal deviation at $r_r$ is $\epsilon$, and the end point of the chord is $r_2$. The cutting curve tangent vector is denoted by $T$. The goal of this approach is to find the end point of the chord that yields the specified nominal chordal deviation.

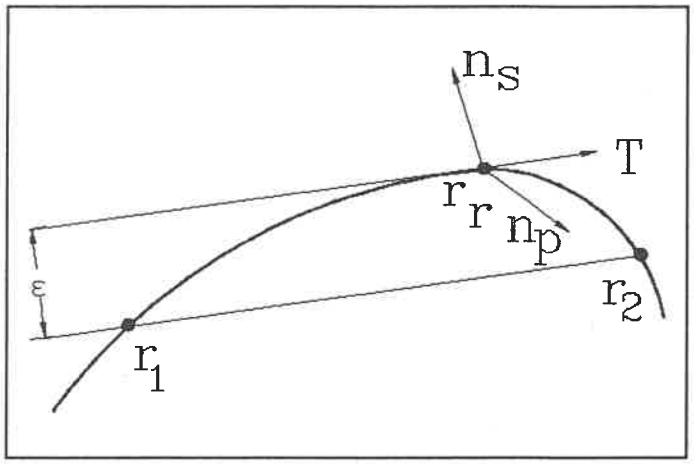


**Figure 11** Initial chordal approximation approach for locating a chord with a specified tolerance

Since the tangent at any point of the cutting curve is contained in the cutting plane and is the tangent plane of the part surface, this indicates that $T$ is perpendicular to both the cutting plane normal $n_p$ and the surface tangent plane normal $n_s$. Thus, $T$ may be calculated by

$$T=\frac{n_s \times n_p}{|n_s \times n_p|} \tag{36}$$

The nominal chordal deviation $\epsilon$ at $r_r$ is obtained by,

$$\epsilon=\overline{r_1 r_r}\cdot\frac{n_p \times T}{|n_p \times T|} \tag{37}$$

where $n_p \times T \,/\, |n_p \times T|$ represents the unit vector located on the cutting plane and perpendicular to $T$ [2]. This method is used in the program for searching the *CC* points .

## 2.3 Convex interference

Many tool path generation algorithms for sculptured surfaces assume that the nominal chordal deviation is the actual machining error. This is true, however, only when the surface normal vectors at $r_1$ and $r_2$ (Figure 12) are parallel, and both are perpendicular to the chord. The true machining error must be determined by

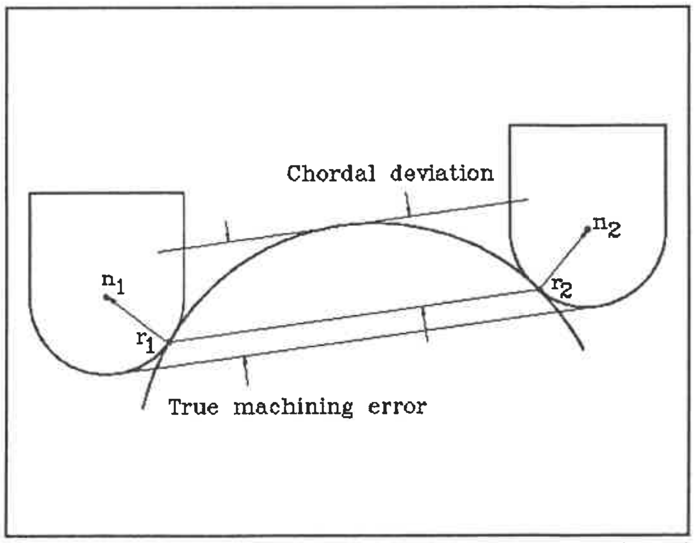


**Figure 12** Physical interference between tool and part surface.

considering the physical interference between the tool and the part surface, as shown in Figure 13. Both the nominal chordal deviation and the distance between the tool center trajectory and the corresponding chord between cutter contact points must be characterized to determine the true machining error.

Typically a pair of tool center (*TC*) points define consecutive tool positions, and the tool moves linearly from one *TC* point to the other. Figure 13 shows a typical convex interference (i.e gouging). The objective here is to protect the *CC* path (not the part surface) from gouging. As depicted in the figure 13, lets $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ be two adjacent *CC* points and let $\boldsymbol{P_i}$ and $\boldsymbol{n_i}$ (i=1,2) be the corresponding offset points and unit normal vectors. Then the amount of gouging $g_i$ at $\boldsymbol{r_i}$ is given by

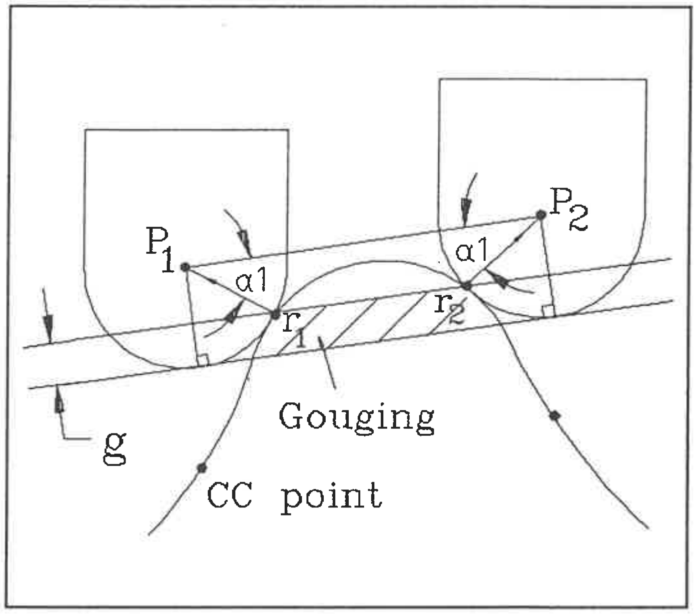


**Figure 13** Convex interference

$$g_i = R(1-\sin(\alpha_i)); \qquad i=1,2 \tag{38}$$

where $R$ is the cutter radius and

$$\alpha_1=\cos^{-1}(n_1\cdot(-f)) \tag{39}$$

$$\alpha_2=\cos^{-1}(n_2\cdot f) \tag{40}$$

where

$$f=\frac{p_2-p_1}{|p_2-p_1|} \tag{41}$$

Then the true error machining $d$ for method 1 is

$$d=g+\epsilon=(\rho+R)(1-\cos(\frac{\alpha}{2})) \tag{42}$$

and for method 2 it is

$$d=g+\epsilon \tag{43}$$

Equations (38) to (43) are also valid for a concave segment, if a negative $R$ value is used in equation (38).

## 2.4 Gouge elimination methods

Cutter interference (part surface gouging) is one of the most critical problems in NC machining of sculptured surface. When the maximum of the normal curvature of the surface at a *CC* point is larger than the normal curvature of the sphere of the ball-endmill, which is given as $1/R$ ($R$ is the radius of the ball-endmill), then the tool interferes with the part surface near the contact point. Some methods of overcoming the gouging problem are presented in this section.

**< method 1 >**

After selecting the *CC* points, gouge detection is accomplished by comparing each individual tool motion to all others along the tool path. Each comparison involves two distinct tool motions and thus four *CL*-points. Each pair is checked for gouging in both the concave and convex sense. If either condition holds, the appropriate corrective action is taken, otherwise the next pair is checked. The algorithm proceeds by comparing the first tool motion to the last, second to the last, and so on until the first motion is compared to the second. Then the second tool motion is compared to the remaining motions in a similar fashion. The method continues until all possible combinations have been considered [19]. But all interferences can't be eliminated by this method. Sometime, there is not an intersection between two distinct tool motions, but there is an interference between the tool and the surface.

**< method 2 >**

Assume that a sequence of *CC* paths is given together with a unit normal vector $\boldsymbol{n}_i$ , that is projected to the cutting plane, at each *CC* point. Let $\boldsymbol{P}_i$ and $\boldsymbol{P}_j$ be the offset point of two adjacent *CC* points $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$ respectively, and let $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$ be in a concave relation. Then if $\boldsymbol{P}_i$ and $\boldsymbol{P}_j$ are in convex relation, both *CC* points become "interfering points" if they are in an inflection relation. Only the *CC* point whose offset point is concave to the other becomes an interfering point (in Figure 14, $\boldsymbol{r}_2$ is the one), and there is no interference if they are in a concave relation [1].

The problem is the same the one that is described in method 1. The selection of *CC* points depends of the maximum tolerance of the surface. Then, checking the interference by verifying the relation between points cannot always give the correct result.

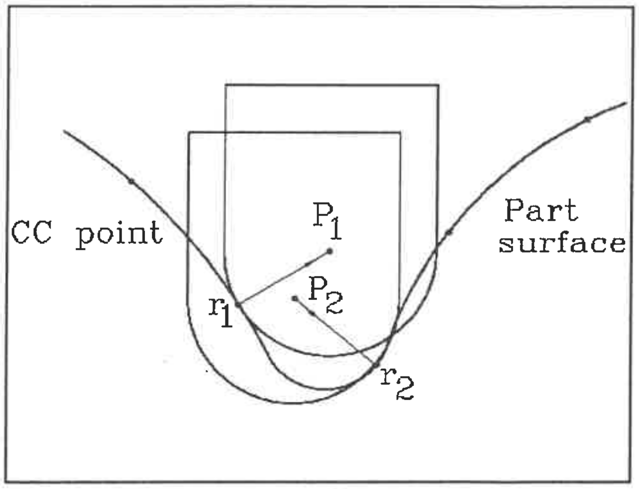


**Figure 14** Concave interference in concave relation

**< method 3 >**

In this method, the intersection between two surfaces are checked in each *CC* point selected. One is the free-form surface and the other is the surface that is generated by the shape of the form of tool. If there is an intersection, the appropriate corrective action must be taken to avoid the interference. This method is time-consuming.

**< method 4 >**

In this method, the nearest distance between the offset point (of selected *CC* point)and the surface will be calculated. There is an interference, if this distance is less than the tool radius. But this method is also time-consuming.

**< method 5 >**

Chen and Raviani [17] presented a method for generating an approximate offset surface using the normal least squares method. By this method, the intersection loop of the offset surface is avoided, if R exceeds the minimum radius of curvature of the contact surface and there is no interference of the tool path if the tool path is planned on it. But it is time-consuming and inaccurate to generate an approximate offset surface.

**Adapted gouge elimination method**

A typical gouge is shown in Figure 15. In this figure, ***IP**$_i$* is the intersection point and *CC0* and *CC1* are cutter contact points. The points *TC0* and *TC1* are offset points from the corresponding *CC* points. As you can see, the ***IP**$_i$* is inside the tool volume. Then, there is an interference. If the point ***IP**$_i$* is in the spherics region ($\alpha \geq \beta$), where

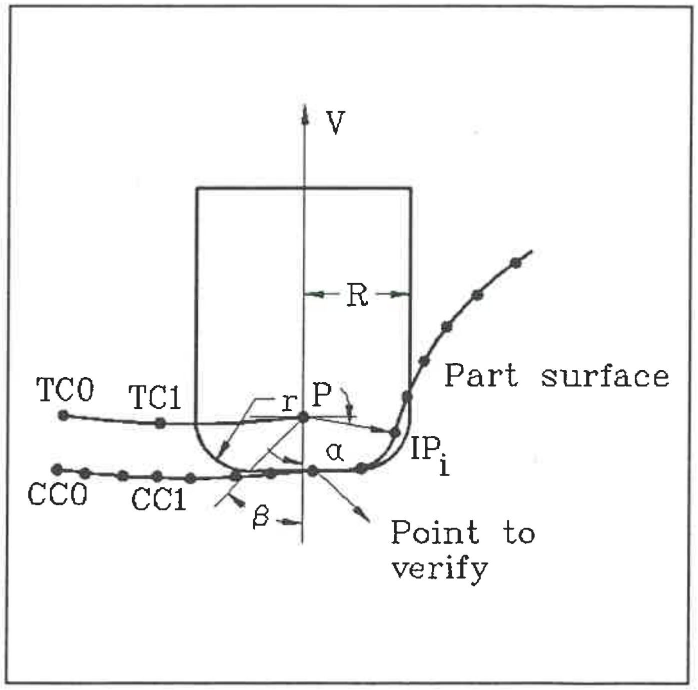


**Figure 15** Adjusting TC points to eliminate gouge

$$\beta = ATANG(\frac{R}{r} - 1) \tag{44}$$

Then

$$r^2 = S^2 + (R-r)^2 - 2S(R-r)\cos(\frac{\pi}{2} - \alpha) \tag{45}$$

where $R$ is the tool radius, $r$ is the small tool radius, and $S$ is the distance between the tool center $\boldsymbol{P}$ and the boundary of tool volume. Solving for $S$ gives,

$$S^2 - 2(R-r)\sin(\alpha)S + [(R-r)^2 - r^2] = 0 \tag{46}$$

$$S = (R-r)\sin\alpha + \sqrt{(R-r)^2\sin\alpha^2 - (R^2 - 2Rr)} \tag{47}$$

If the point $\boldsymbol{IP}_i$ is in the cylindric region ($\alpha \le \beta$), we will have,

$$S = \frac{r}{\cos\alpha} \tag{48}$$

Therefore, if $|\boldsymbol{P} - \boldsymbol{IP}_i| < S$, there is a gouging.

The new method developed in this paper consists of two cases.

1. Simple case: gouging happens along the path direction.

When the new intersection point (that is obtained using the intersection between a plane and the surface) is verified to select the CC point, the interference is checked simultaneously at this point (see Figure 15). This procedure works as follows:

1. read the new intersection point from the list of points (see Appendix B).
2. is there any intersection point in the area contained by the tool diameter, that is inside the tool volume?
3. If yes, the last point in the list of points is selected as a new *CC* point. Then the other points are checked until a point is found where there is no interference between the tool and the surface, if the tool is located at the point.
4. If no, this point is checked for calculating the nominal chordal deviation.

2. Complicated cases: gouging happens in any direction of the surface.

For a more complicated free-form surface, the gouging will not only happen along the path direction but may also possibly happen in any direction of the surface. The checking defined only in the path direction dose not guarantee a good result. A more general gouging detection method is described as follows. The gouging check is made between the current offset point (that is checked for initial chordal deviation) and all the intersection points that are contained by the area of the tool diameter. If there is an interference, in 3-axis machining, the last offset point is selected. In 5-axis machining, with an end-mill, it is possible also to change direction of the tool axis to avoid the interference.

The advantage of this method is that each point is separately checked against the interference in the same time that normal chordal deviation is calculated. Moreover, the method is not dependent on the shape tool. Another advantage of the direct gouge elimination method is that it detects and eliminates gouging tool motions simultaneously to the location procedure. Then, when the *CC* points are selected, there is not any interference between the tool and the surface. The time required is not very long because the offset points are only compared with the points that are contained by the area of tool diameter. This new method is used for verifying the interference in the program.

## 2.5 Tool pass interval adjustment

The distance between two adjacent tool passes is referred to as the tool pass interval, as depicted in Figure 16. The cusp is the remaining material between two adjacent tool passes, and it affects the smoothness of the machined part surface. If the part surface is flat, the tool pass interval is constant for all tool passes. However, for sculptured surfaces, the tool pass interval is generally different from one pass to another, depending on the tool radius and the local surface curvature. Tool pass intervals must be calculated from a specified cusp height tolerance to ensure surface smoothness.

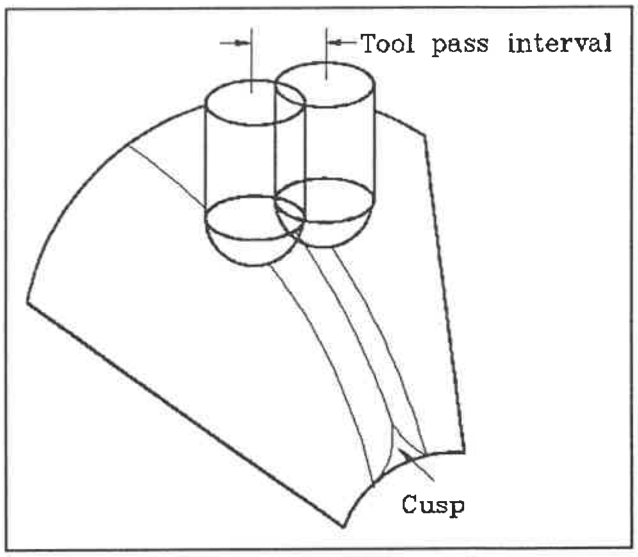


**Figure 16** Tool pass interval and cusp

### 2.5.1 Tool pass interval adjustment with ball-mill

The tool pass interval adjustment technique uses the radius of curvature in the direction of the cutting plane normal. The tool pass interval $l$ on a non-flat surface, as depicted in Figure 17, can be calculated by approximating the actual surface curve between $r_1$ and $r_2$ by a circular arc, and observing that

$$(\rho+R)^2+(\rho+h)^2-R^2 = 2(\rho+R)(\rho+h)\cos(\phi) \tag{49}$$

where $\rho$ is the curvature radius of the surface and

$$\cos\phi = \sqrt{1-(\frac{l}{2\rho})^2} \tag{50}$$

where $l$ is the distance between $r_1$ and $r_2$. Solving for $l$ yields,

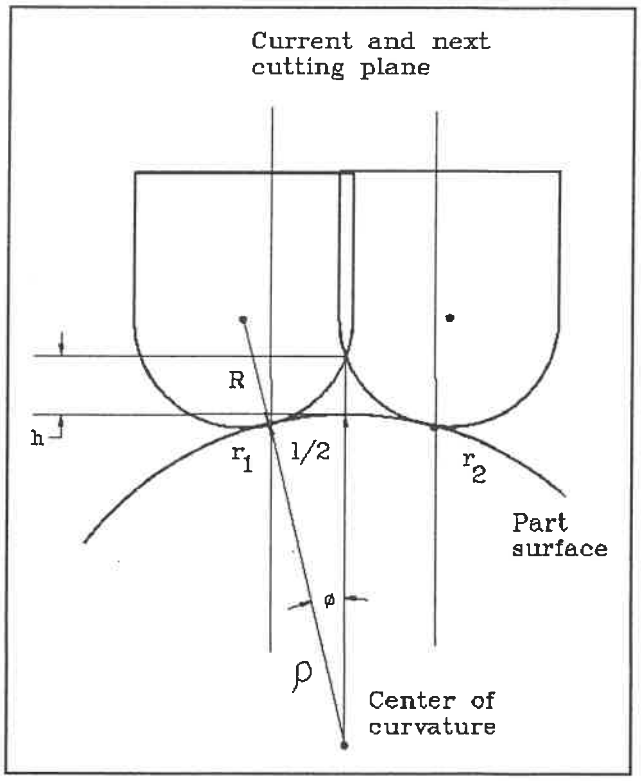


**Figure 17** Tool pass interval calculation

$$l = \frac{\rho\sqrt{4(\rho+R)^2(\rho+h)^2-[\rho^2+2R\rho+(\rho+h)^2]^2}}{(\rho+R)(\rho+h)} \tag{51}$$

and for a concave surface, the value of $l$ is given by

$$l = \frac{\rho\sqrt{4(\rho-R)^2(\rho-h)^2-[\rho^2-2R\rho+(\rho-h)^2]^2}}{(\rho-R)(\rho-h)} \tag{52}$$

where the radius of curvature of the part surface is flat the tool pass interval $l$ becomes

$$l=2\sqrt{2Rh-h^2} \tag{53}$$

### 2.5.2 Tool pass interval adjustment with end-mill

The expression for an inclined end-mill cutter on a different surface is calculated using the equation of the elliptical cutter profile projection that is given by,

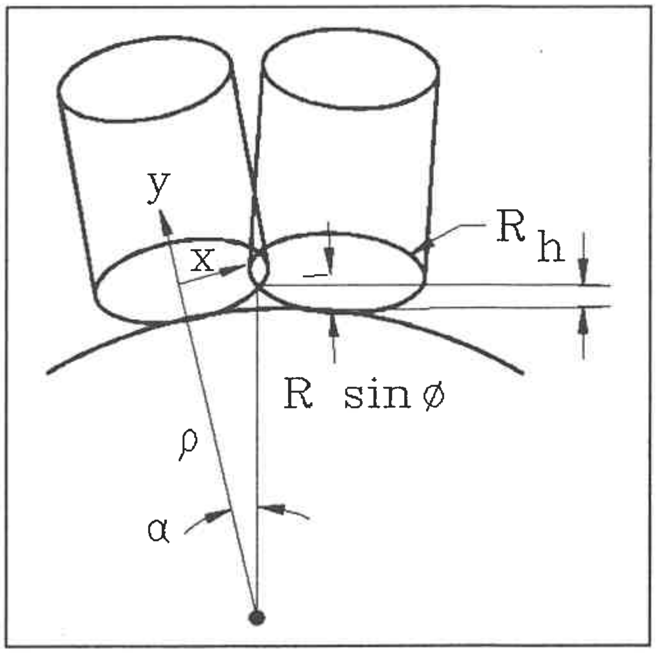


**Figure 18** Cusp height resulting from end-mill on a convex surface

$$\frac{x^2}{R^2}+\frac{y^2}{R^2\sin^2\phi}=1 \tag{54}$$

$$x=(h+\rho)\sin\alpha \tag{55}$$

$$y=(h+\rho)\cos\alpha-(\rho+R\sin\phi). \tag{56}$$

By combining equations(54),(55) and (56), the elliptical equation becomes,

$$\frac{(h+\rho)^2\sin^2\alpha}{R^2}+\frac{(h+\rho)^2\cos^2\alpha-2(h+\rho)\cos\alpha(\rho+R\sin\phi)+(\rho+R\sin\phi)^2}{R^2\sin^2\phi}=1 \tag{57}$$

Rearranging gives

$$\cos^2\alpha(h+\rho)^2\cos^2\phi-\cos\alpha[2(h+\rho)(\rho+R\sin\phi)]+[\rho^2+2\rho R\sin\phi+(h+\rho)^2\sin^2\phi]=0 \tag{58}$$

Solving for $\cos\alpha$ gives

$$\cos\alpha=\frac{\rho+R\sin\phi-\sqrt{(\rho+R\sin\phi)^2-\cos^2\phi[\rho^2+2\rho R\sin\phi+(h+\rho)^2\sin^2\phi]}}{(h+\rho)\cos^2\phi} \tag{59}$$

For a concave surface, the value of $\cos\alpha$ is given by

$$\cos\alpha=\frac{\rho-R\sin\phi-\sqrt{(\rho-R\sin\phi)^2-\cos^2\phi[\rho^2-2\rho R\sin\phi+(h-\rho)^2\sin^2\phi]}}{(h-\rho)\cos^2\phi} \tag{60}$$

The resulting cusp height is given by,

$$\sin\alpha\simeq\frac{l}{2\rho} \tag{61}$$

Then

$$L=2\rho\sin\alpha \tag{62}$$

For a plane surface $\rho$ approaches $\infty$ and therefore $\cos\alpha = 1$ and $\sin\alpha \approx l / 2 \times \rho$(see Figure 19). The resulting cusp height is given by

$$h = R\sin\phi - \sin\phi\sqrt{R^2 - \frac{L^2}{4}} \tag{63}$$

This can be written as,

$$l = \frac{2}{\sin\phi}\sqrt{2Rh\sin\phi - h^2} \tag{64}$$

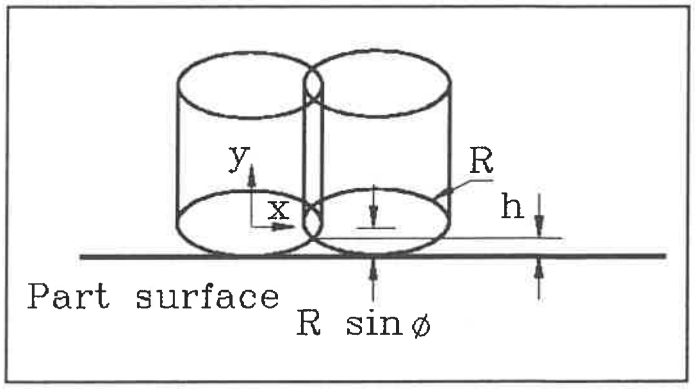


**Figure 19** Cusp height resulting from end-mill on a plane surface

### 2.5.3 Distance between two planes

The line on which the tool pass interval is measured, is not generally parallel to the normal vector of the cutting planes. So, in order to define the next cutting plane, the tool pass interval $l$ is projected onto the cutting plane normal as shown in Figure 20. Note that since a circular arc approximation is used to find the tool pass interval $l$, angle $\alpha$ can be calculated by

$$\alpha = \cos^{-1}\left(\frac{l}{2\rho}\right) \qquad \textbf{(65)}$$

Denoting the angle between the cutting plane normal and the surface normal at the cutting contact point O as angle ß the projection magnitude of the tool pass interval can be calculated by

$$l' = l\cos(\alpha + \beta - \pi) \qquad \textbf{(66)}$$

and for a concave surface

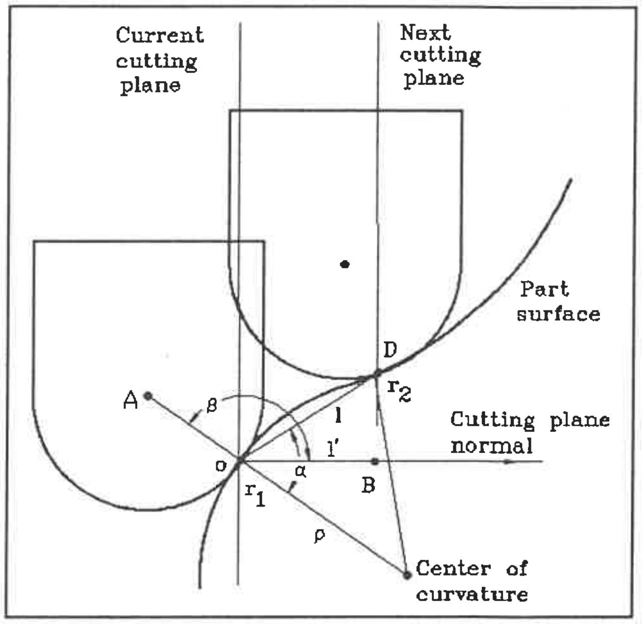


**Figure 20** Projecting tool pass interval onto the cutting plane normal

$$l' = l\cos(\alpha - \beta) \qquad \textbf{(67)}$$

where *l'* is the projection magnitude of *l*. On a flat surface, *l* lies exactly on the flat surface and angle DOC is equal to $\pi/2$; then,

$$l' = l\cos\left(\alpha - \frac{\pi}{2}\right) \qquad \textbf{(68)}$$

The tool pass interval is calculated at all *CC* points of a tool pass by using the tool pass interval adjustment method combining with the magnitude projection formulation. The resulting tool pass intervals at all CC points are compared and the smallest one is applied to define the next cutting plane.

**2.6 Milling in 3-axis machining**

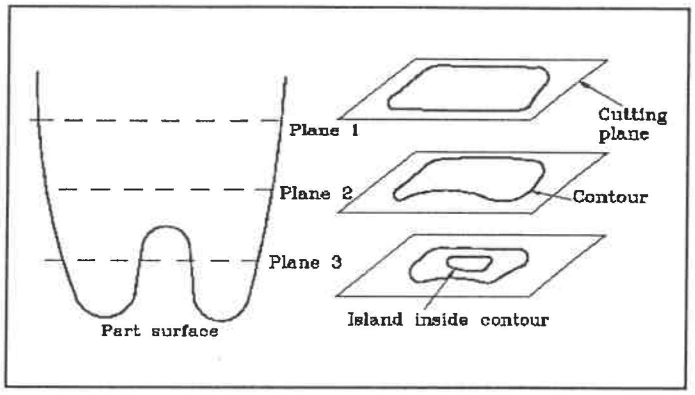


**Figure 21** Different contours on intersecting planes

A standard task in machining die cavities or mechanical parts is the removal of material within a given boundary. A procedure developed in this paper, is needed to cut away the bounded area, which is called a pocket, in order to use efficiently a numerically controlled (NC) machine. To find this area, boundary curves are defined using the intersection between a series of horizontal cutting planes and the sculptured surface (see Figure 21). The normal vectors of all the intersection points are projected on the cutting plane and their offset points are calculated. The *CC* points which give the chordal deviation and do not have any interference with the surface are selected. With these points, the equation of the cutting curve can be defined using kriging.

A pocket can be machined in a zigzag or spiral fashion. Tool path generation for these two methods are discussed in this section.

### 2.6.1 Zigzag fashion

The intersection points between the cutting curve and a series of lines that lie on the cutting plane are obtained (see Figure 22). The distance between lines are less than the tool diameter. The tool is passed from these points and then, the primary

*CC* points on the cutting curve is machined.

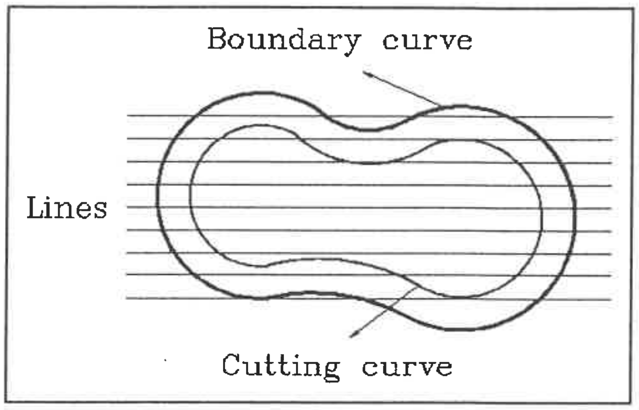


**Figure 22** Zigzag machining

### 2.6.2 Spiral fashion

The spiral machining is often used in uniform pocket cutting and requires more difficult cutter path calculations than zigzag machining. In this method, after having determined the cutting curve, its center is found. The equation of a surface can then be defined using the method for kriging surfaces. This surface is composed of two curves. One is a cutting curve and another is a curve where all its points are equal to the value of the point at the center of the cutting curve.

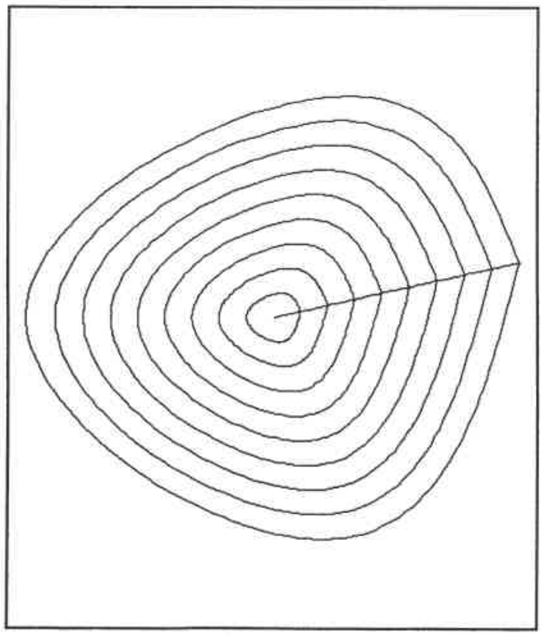


**Figure 23** Spiral fashion

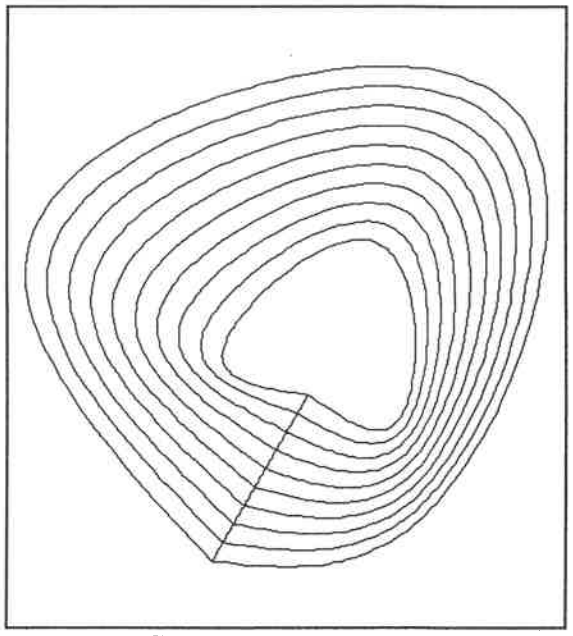


**Figure 24** Spiral fashion with an island

In order to select the *CC* points, a series of points on the curves are selected using the surface parameters. The maximum distance between curves is less than the tool diameter (see Figure 23). If there is an island inside the cutting curve, the surface can be defined using the cutting curve and the island curve ( see Figure 24).

## 2.7 Tool-path generation for RFS

The tool path generation algorithm developed for free-form surfaces is applicable to the rotation-free surface (RFS), because RFS can be easily generated by dual kriging. The cutting curves can be defined by the intersection of a set of parallel planes as shown in Figure 25. Then, the offset points which are not gouging with the surface are selected with the previous method.

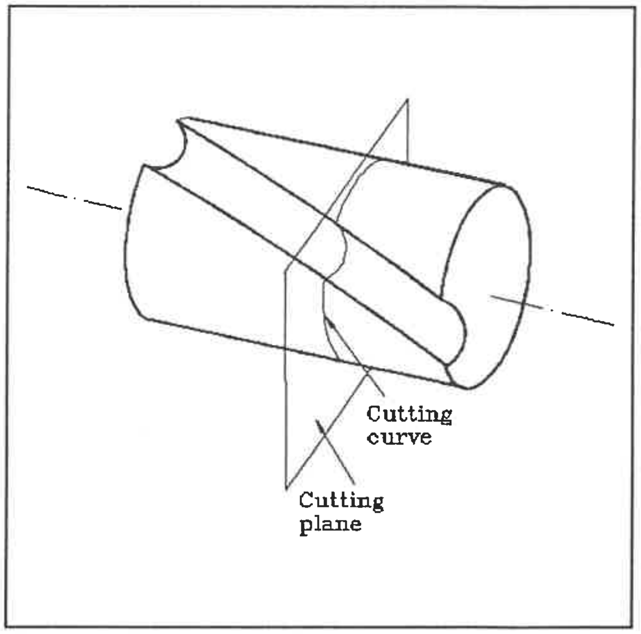


**Figure 25** Cutting plane and cutting curve definitions

## 2.8 Comparing tools and selecting the tool axis vector

When machining curved surfaces, spherical ball-mills are inevitably used. The historical reasons are that ball-mills are easy to position in relation to curved surfaces, generate simple and short numerically controlled machining programs and often only require two-dimensional cutter compensation.

However, when machining plane surfaces, face of end-mills are used. The reasons are that the flat ended cylindrical endmills match exactly the geometry of the required surface and that they are readily available with carbide inserts for faster machining. If a ball-mill was used to face a plane surface it would require many more passes across the surface to generate the same surface finish as that produced with an end-mill.

The same argument holds when machining a wide class of smooth low curvature surfaces such as those encountered with turbines, propellers, and aircraft structural components. The critical factor for rapid and efficient machining of all these surfaces is that the cutter shape should match the surface shape as closely as possible. The profile of an end-mill can be made to match that of a curved surface by inclining it correctly to the surface normal (see Figure 26) [10]. The effective radius of curvature, $r_{eff}$ , of an end-mill can vary from infinity down to the cutter radius R, as the inclination of the cutter to the surface normal, $\phi$, changes from $0 \leq \phi \leq 90$ deg. For relatively small depths of cut, the effective radius, $r_{eff}$ , is

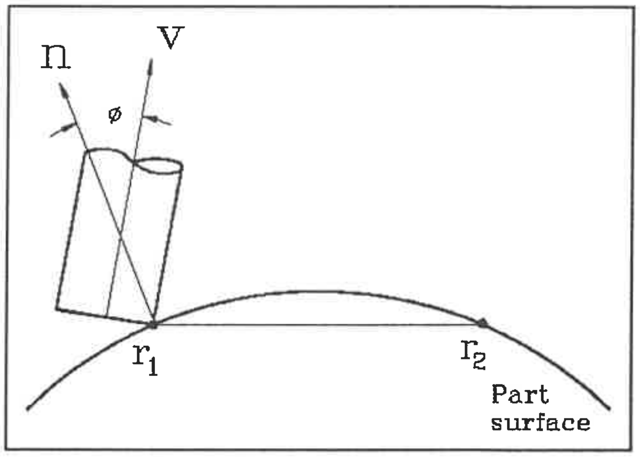


**Figure 26** The effective radius of an end-mill cutter inclined to a surface

given by

$$r_{eff} = \frac{R}{\sin\phi} \tag{69}$$

The effective radius of curvature of a ball-mill is, of course, restricted to the spherical radius of the cutter [10].

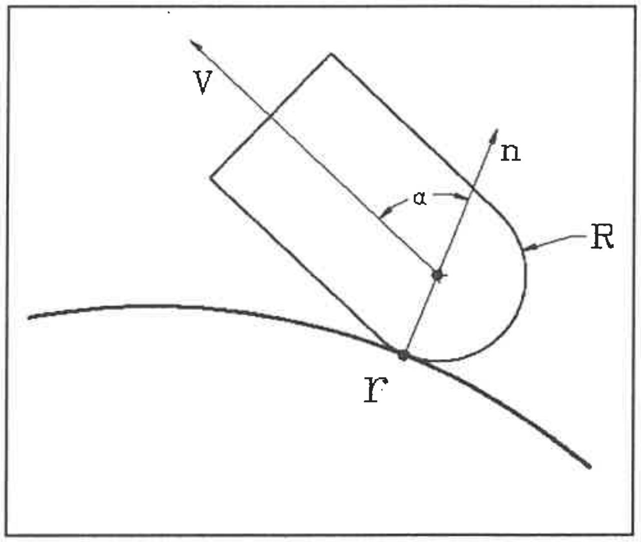


**Figure 27** Change in cutting speed of ball-end mill due to tool attitude

Another important difference between ball-mills and end-mills when machining curved surfaces is the speed of material removal or cutting speed. The tool attitude against the surface partially changes due to the collision avoidance (see Figure 27). The change of tool axis direction causes the variation in the cutting speed though the cutting point is the same. The desired spindle rotation S given in the following expression

$$S = v * 1000/(2 * \pi * R * \sin\alpha) \tag{70}$$

where $v$ and $R$ are the cutting speed and the tool radius respectively. A ball-mill cuts at a portion of the sphere near the axis of rotation and thus has minimal cutting speed. In effect, the cutter is rubbing the material away. With end-mills, however, the material is always cut at the periphery of the cutter at a full and predetermined cutting speed.

A further difference is that ball-mills are usually made from high-speed steels which do not have the machining capability of carbide insert cutters; they require careful spherical resharpening and are only available in relatively small sizes. End mills are readily obtainable in large sizes and with replaceable carbide inserts. Therefore, the selection of tools in machining sculptured surface is dependent to the form of surface and requires good experience.

In three axis machining, the tool can be selected as follows (see Figure 28):

1. Machining surfaces whose slope (with respect to the tool axis $v$) changes substantially, i.e. $0° < \alpha < 90°$. A spherical-end mill may be used, but one should remember that in horizontal surface regions ($\alpha < 10°$) the cutting speed decreases drastically.

2. Machining of nearly horizontal surfaces ($\alpha \approx 30$). The largest machined strip width (among tools with the same diameter) is obtained if a toroidal miller with the small torus radius ($r<0.2$) is applied. Cutting speeds may be kept within a reasonable interval but in the case of concave surfaces one has to avoid undercutting.

When machining sculptured surfaces on a 5-axis CNC milling machine with an end mill, the direction vector of the milling cutter must be determined. The direction vector consists of a tilting angle and a rotation angle. In the machining of large

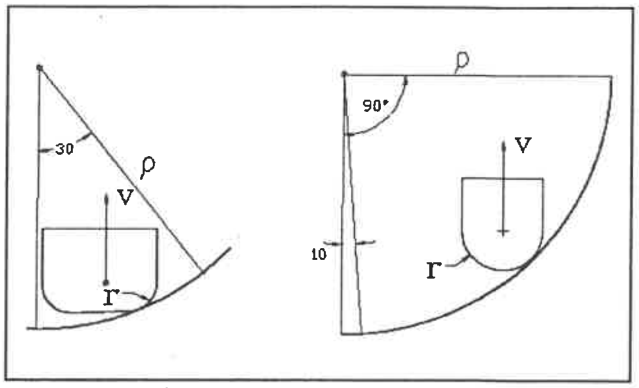


Figure 28 Intervals of $\alpha$ for different tool shapes.

sculptured surfaces (having large curvature radius), the direction vector may be based upon the curvatures at the cutting point (*CC-point*) [10]. But, for smaller free-form surfaces or precise parts, this method is inadequate since the outer region of the end mill interferes with the sculptured surface. Any small error in the direction vector induces high cusp or deep overcuts in the 5-axis machining process. This is why, machining surfaces by 5-axis CNC milling with an end mill has been mainly confined to convex or large concave surfaces [11].

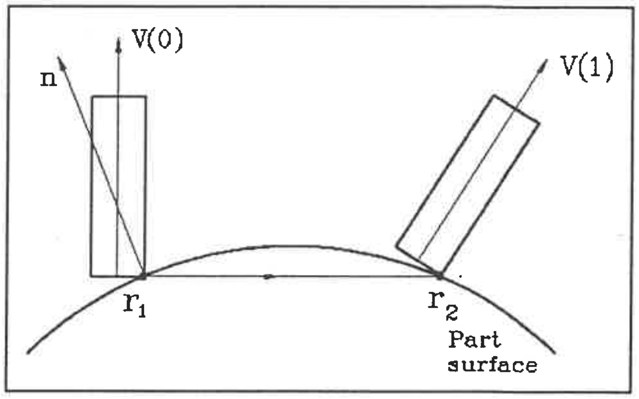


Figure 29 Tool axis vector

The direction vector of the milling cutter must be determined to produce minimum cusp heights on the machined surfaces. The minimum cusp height can be obtained, if the tool axis vector is in the direction of the surface normal vector. But to eliminate the concave interference and the convex interference that is described in section 4.3, it is possible to determine the tool axis vector so that it is normal to the vector $r_1r_2$(see Figure 29).

$$\overline{r_1 r_2} = \frac{r_1 - r_2}{|r_1 - r_2|} \tag{71}$$

$$A = \frac{\overline{r_1 r_2} \times n}{|\overline{r_1 r_2} \times n|} \tag{72}$$

$$V = \frac{A \times \overline{r_1 r_2}}{|A \times \overline{r_1 r_2}|} \tag{73}$$

where $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ are two *CC* points and $n$ and $v$ are normal vector to the surface and tool axis vector respectively.

# CHAPTER 3

# Multi-axis machining

Due to the development of CNC machines and software packages, sculptured surfaces usually are machined with a three-axis milling machine using a ball-end mill cutter. By using 3-axes milling with ball end mills, the tool engagement modes change along a given contour, because permanently different parts of the cutting edge are active. Different cutting speeds appear along the radius. The maximal cutting speed is reached on the tool diameter, the minimal at the top. However, machinability at the bottom of the ball-end mill is poor, and sometimes, workpieces of complex geometry cannot be machined by three-axis milling. In addition, the ball-end mill always produces a cusp on the machined surface. In order to decrease cusp heights when machining sculptured surfaces with a ball-end mill cutter, the tool path interval must be adjusted in consideration of the cusp height. This method requires a long machining time, and manual polishing is required to remove the cusp. Recently, robots have been used in the polishing process in order to enhance the efficiency of die mould production. But robot polishing requires high cost equipment and is a complex geometry. For these reasons, 5-axis CNC milling has been recommended for machining free-form surfaces. When using by 5-axes milling and ball mills, unfavourable cutting condition ($v = 0$ and chip space) can be avoided by using a tilt angle. The tilt should be chosen under consideration of possible collisions. The application of ball end mills in 5 axes milling does not offer any geometrical advantage. It is just important to machine deep and complex cavities. Generally, 5-axis milling yields smaller cusp heights than three-axis milling. Cutting and polishing times in this machining process are shorter than those in the three-axis machining process.

## 3.1 Multi axis machining problems

Collision and linearisation are two critical problems in multi axis machining that are discussed in this section.

### 3.1.1 Collision avoidance

Tool interference problem is one of the most critical problems in the NC machining of free-form surfaces. In 3-axis machining, the tool axis is always z axis, but in multi-axis machining, an offset must be given to the tool along a normal vector to the surface. The tool and the surface come in contact only at desired surface locations and the tool axis varies as the part rotates. By this configuration, some complex surfaces impose an additional source of interference called tool axis interference (TAI) or collision, that cannot be avoided in three-axis machining even if detected. In other words, an implicit assumption in three-axis machining is that the surface geometry is free of such interference. However, TAI is very critical in multi-axis machining, and may be avoided by rotating the part. Some methods of overcoming the collision problem are presented in this section.

**< method 1 >**

In this method, the tool shape is approximated by designating a set of check points in the circumferential and longitudinal direction of the tool as mentioned by Yankeuchi and al [18]. The check points are automatically determined and stored as tool shape data. The collision check is carried out by examining whether or not the check points exist within the workpiece. When a collision takes place, the tool orientation at the cutting point must be changed by finding the collision-free tool axis vector. This method is practical, if the workpiece is defined in constructive solid geometry (CSG).

**< method 2 >**

In this method, the intersection between two surfaces are checked. One is the free-form of the surface and other is the surface generated by the shape of the tool. If there is an intersection, the appropriate corrective action must be taken to avoid it. This method is time-consuming.

**< method 3 >**

In this method, a line in the direction of the tool axis is generated. The original point of this line is the tool center and the final point is the length of the tool in direction its axis. If there are two intersections between the part surface and the line, a collision has occurred. This method can also be used with 2 or 4 lines that represent the tool volume. In order to eliminate the collision, the tool axis is changed toward the z axis until the collision is eliminated. This method is time-consuming.

**Adaptive collision avoidance method**

The collision avoidance procedure developed in this project is explained as follows:

Tool-axis-interference (TAI) is present if there exists a surface point **r'** satisfying the following conditions (see Figure 30):

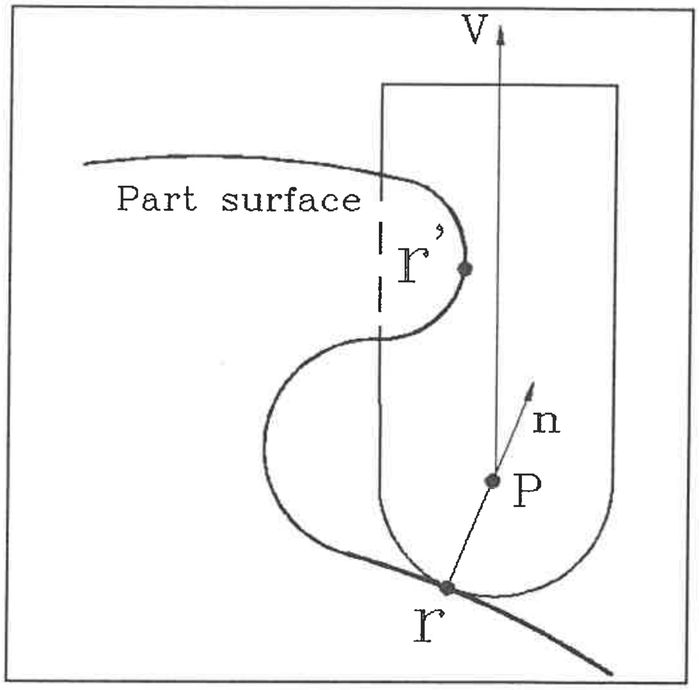


**Figure 30** Tool axis interference

$$\overline{Pr'} \cdot V \geq 0 \tag{74}$$

$$|\overline{Pr'}|^2 - (\overline{Pr'} \cdot V)^2 < R^2 \tag{75}$$

where **V** is the tool axis and R is tool radius.

If there is a collision, then the vector of the milling cutter is changed in the manner that cause the smallest cusp height (when machining with an end-mill) and no interference. Comparing to others methods, this method is fast and precise. This method is selected in the program.

### 3.1.2 Linearization of tool path

When machining between two positions which have a differential in the tool axis direction vectors, overcuts are produced by swivel movements around the pivot point. In order to solve this problem, linearization of the tool path is applied [11].

Figure 31 shows cutting positions with the total interval of the linear interpolation. The starting point is known, and the end point is obtained from linear interpolation which sets maximum error within the total interval to be $\epsilon_{max}$ as in three axis CNC ball-end milling. At the two positions, direction vectors of the milling cutter can be obtained from the equations discussed in the next chapter. Cutting axis direction vectors vary linearly between two positions. Thus, a finite interval of interpolation can be obtained on the condition that the predicted error is constrained within the allowable machining error $\epsilon_{max}$. If the maximum error in a finite interval of the *kth* step is $\epsilon(k)$ and the error due to the influence of rotational axis in a finite interval of the *kth* step is $\epsilon_{ir}(k)$ as shown in

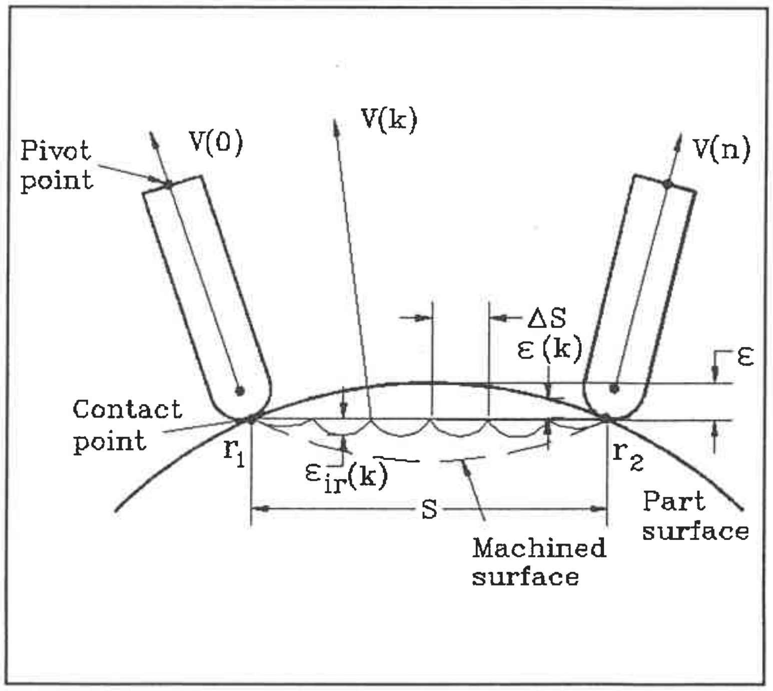


**Figure 31** Linearization for the 5-axis CNC milling process.

Figure 31, its relation is represented by

$$\epsilon_{ir}(k)+\epsilon(k)\leq\epsilon_{\max} \tag{76}$$

and for a convex surface, we have

$$\epsilon_{ir}(k)\leq\epsilon_{\max} \tag{77}$$

The error in a finite interval of *kth* step $\epsilon(k)$ is approximately described by (see Figure 31)

$$\epsilon(k)\approx\frac{\epsilon}{0.25}((-\sum_{i=0}^{k}\frac{\Delta S^{(i)}}{S})^{2}+\sum_{i=0}^{k}\frac{\Delta S^{(i)}}{S}) \tag{78}$$

The equations 76 and 77 are changed, depending of the position of the pivot point in different machines.

The vector $V^{(k)}$, from the *CC* point to the pivot point in the *kth* step, is given by

$$V^{(k)} = \frac{U^{(k)}}{|U^{(k)}|} \tag{79}$$

where

$$U^{(k)}=V^{(0)}+(V^{(n)}-V^{(0)})\sum_{i=0}^{k}\frac{\Delta S^{(i)}}{S} \tag{80}$$

Then, the relation between the overcut error $\epsilon_{ir}(k)$ and the angle variation $\Delta\theta^{(k)}$ at the *kth* step is approximately described by

$$\epsilon_{ir}(k)\cong L(1-\cos\frac{\Delta\theta^{(k)}}{2}) \tag{81}$$

where $\Delta\theta(k)$ and $L$ are $\cos^{-1}(V^{(k-1)}.V^{(k)})$ and the distance between the pivot point and the contact point respectively.

Thus, the interval between two points is divided into several small intervals according to the tolerance. The tool center point and tool axis vector at each divided point are calculated. It is noted that this correction must be made in a postprocessor because the parameters of these equations are dependent on the type of the machine tool. If the new points are calculated on the surface, it is possible to solve this problem with minimum number of added point (see Figure 32)

Therefore, in multi-axis machining if there is a large variation in the cutting axis direction vectors between points, increasing the number of points can reduce this problem.

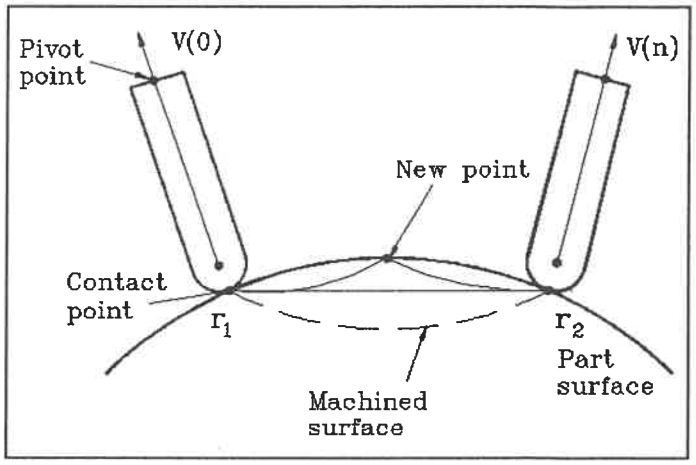


**Figure 32** Linearization for the 5-axis CNC milling process.

## 3.2 Multi axis control machining

The post-processor is a section of the program that converts cutting location ($CL$) to the coordinate system of the machine. It uses the machine data and the cutting condition data to calculate the rotation angle of the rotational axis (R axis) and the tilting axis ( I axis ) on the basis of the tool axis vector. It also determines the coordinates of the tool center after the rotation.

### 3.2.1 4-axis control machining

Machining a rotational-free surface RFS requires the motion of part rotation as well as tool interpolation. Some RFS parts may be machined by sequential operations (turning followed by milling), but most cannot since they require synchronized part rotation and tool interpolation. Even if RFS parts can be machined by a sequential operation, 4-axis operation is desirable because sequential operation requires more loading/unloading, which is usually done manually and takes a considerable amount of time. In the 4-axis operation, the whole surface can be machined with one setup (see Figure 33). Thus, 4-axis operation is a powerful alternative to automation of the machining process and the reduction of the cycle time [7].

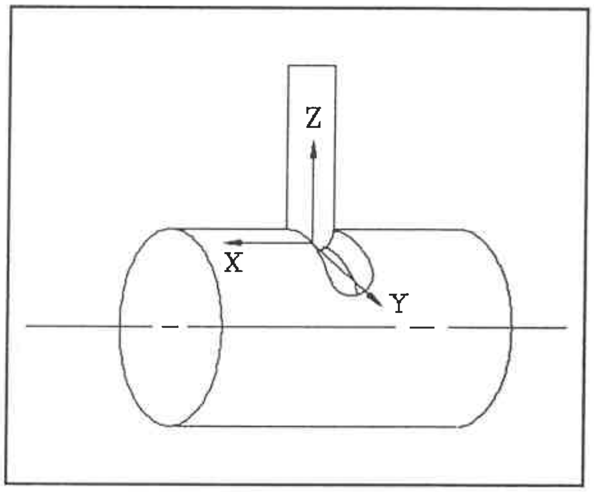


**Figure 33** 4-axis configuration

The next section described the equations are developed to find the angle of rotation and the position of tool center.

### 3.2.1.1 Four-axis control mode

In the configuration shown in Figure 34 three Cartesian axes are used for the tool motion and the fourth axis rotates the workpiece. In practice, the configuration is often achieved by attaching a fourth axis to the original three-axis machine, giving an additional 4-axis machine. Because this simple automation method is economically attractive, it is increasingly used in machine shops.

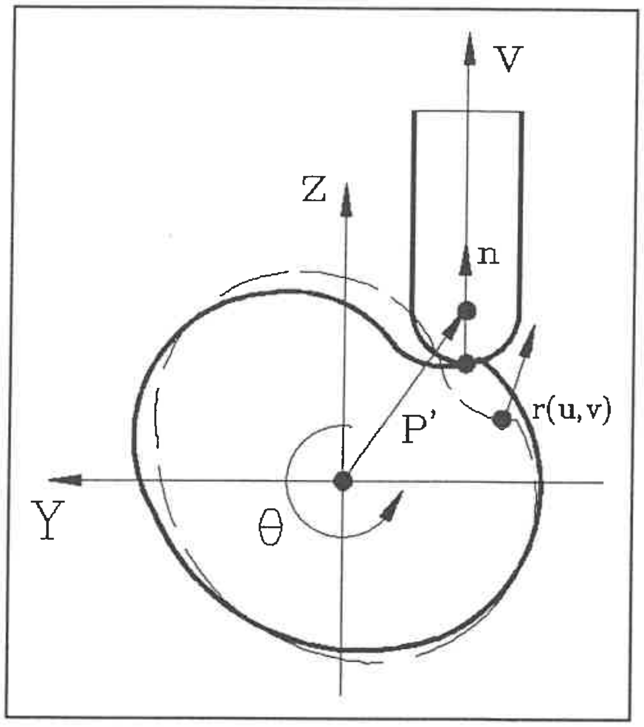


**Figure 34** 4-axis scheme

The rotation angle used to access one offset point ***P(u,v)*** on the surface is determined such that the normal vector ***n(u,v)*** is parallel to the XZ plane. Decomposing the normal vector ***n*** into $P_{ax}$, $P_{ay}$, $P_{az}$, we have

$$I = E^{x}(\theta).P \tag{82}$$

or

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C\theta & -S\theta \\ 0 & S\theta & C\theta \end{bmatrix} \cdot \begin{bmatrix} P_{nx} & P_{ox} & P_{ax} \\ P_{ny} & P_{oy} & P_{ay} \\ P_{nz} & P_{oz} & P_{aZ} \end{bmatrix} \qquad \textbf{(83)}$$

by inverting the matrix $\mathbf{E}^x(\theta)$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & C\theta & S\theta \\ 0 & -S\theta & C\theta \end{bmatrix} = \begin{bmatrix} P_{nx} & P_{ox} & P_{ax} \\ P_{ny} & P_{oy} & P_{ay} \\ P_{nz} & P_{oz} & P_{aZ} \end{bmatrix} \qquad \textbf{(84)}$$

where $\boldsymbol{P}$ is a homogenous matrix of offset points, $I$ is the identity matrix and $\mathbf{E}^x(\theta)$ is the rotation matrix about $x$ .

$$\theta = ATANG2(P_{ay}, P_{az}) \qquad \textbf{(85)}$$

When rotating the workpiece by an angle $\theta$, the new coordinate of point $P$ becomes

$$P' = E^x(\theta) \cdot P \qquad \textbf{(86)}$$

The angle $\theta$ and the point $\boldsymbol{P'}$ transfer to the machine coordinate.

## 3.2.2 Five-axis control machining

The structure of 5-axis control machining centers is composed of three translational movements along the x, y and z axes and two rotational movements of rotation and tilt. According to the type of rotational movements, the machine is usually classified into one of three categories : table-tilting with two degrees of freedom on the table, spindle-tilting with two degrees of freedom on the spindle, and table and spindle having each one degree of freedom [8].

In this paper, we use the Pual's method [5] and develop the necessary equations for finding the angles of rotation and coordinates of tool center.

### 3.2.2.1 Table-tilting types

Table-tilting type machining centers are divided into two types, as illustrated in Figure 35 and 36:

-type 1 with a rotational table on a tilting one.
-type 2 with a tilting table on a rotational one.

With regards to type 1 and 2, let us explain the conversion of CL Data to the machine coordinate system (MCS). It is supposed that the direction of the x,y and z axes in a work coordinate system (WCS) correspond to that in the MCS and that the direction of the z axes is equal to that of the main spindle.

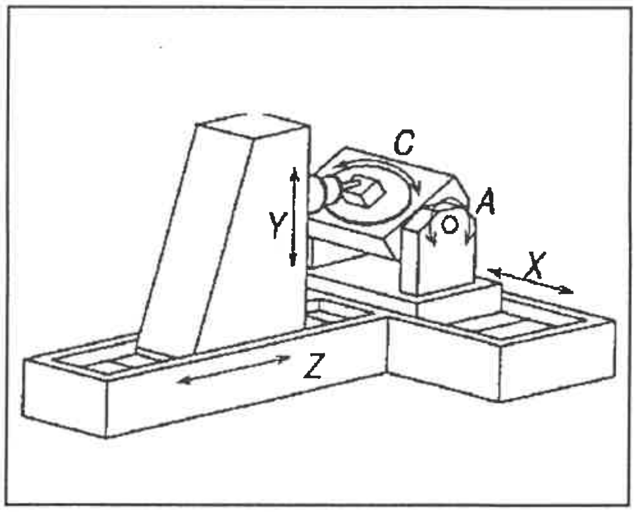


**Figure 35** Type 1: table-tilting machining centers [8]

Namely, the workpiece is mounted on the table so that the x, y and z directions in WCS may correspond to those in MCS when the table of a machining center is faced with the spindle.

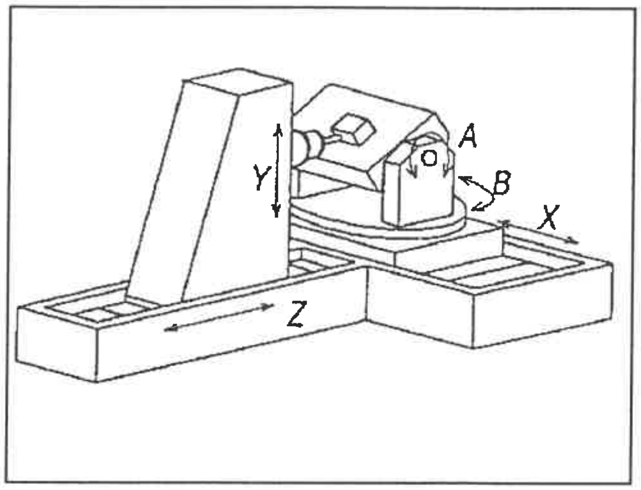


**Figure 36** Type 2: table-tilting machining centers [8]

**< Type 1 >**

As an example of Type 1, let us consider such a structure as illustrated in Figure 37(a). The rotation angles of the table, $\theta$ and $\phi$, are determined in order to make tool axis correspond to the z axis. The motion can be expressed in terms of a drive transformation matrix $D$. We will then represent $D$ as

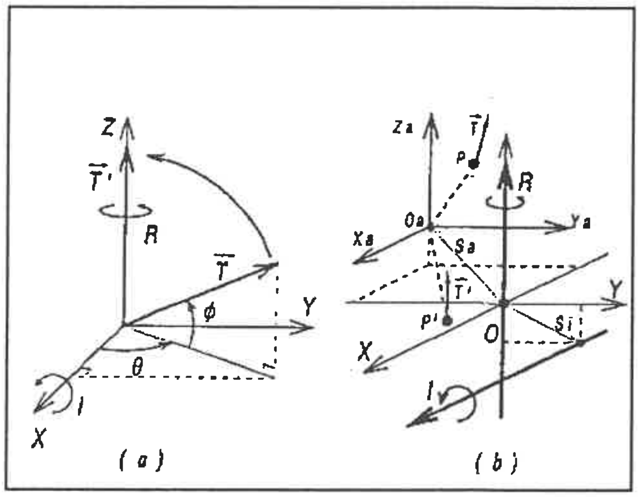


**Figure 37** Table-tilting type with a rotational table on a tilting one [8].

$$D = E^{x}(\phi)\,.\,E^{z}(\theta) \tag{87}$$

$$I = D\cdot P \tag{88}$$

$$D^{-1} = P \tag{89}$$

where P is a homogenous matric of offset points, *I* is the identity transform, and $\mathbf{E}^{x}(\phi)$ and $\mathbf{E}^{z}(\theta)$ are rotational matrices about *x* and *z* axis by an angle $\phi$ and $\theta$ respectively

$$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} = \begin{bmatrix}1&0&0\\0&C\phi&-S\phi\\0&S\phi&C\phi\end{bmatrix}\cdot\begin{bmatrix}C\theta&-S\theta&0\\S\theta&C\theta&0\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}P_{nx}&P_{ox}&P_{ax}\\P_{ny}&P_{oy}&P_{ay}\\P_{nz}&P_{oz}&P_{aZ}\end{bmatrix} \tag{90}$$

$$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} = \begin{bmatrix}C\theta&-S\theta&0\\S\theta C\phi&C\theta C\phi&-S\phi\\S\theta S\phi&C\theta S\phi&C\phi\end{bmatrix}\cdot\begin{bmatrix}P_{nx}&P_{ox}&P_{ax}\\P_{ny}&P_{oy}&P_{ay}\\P_{nz}&P_{oz}&P_{aZ}\end{bmatrix} \tag{91}$$

$$\begin{bmatrix}C\theta&S\theta C\phi&S\theta S\phi\\-S\theta&C\theta C\phi&C\theta S\phi\\0&-S\phi&C\phi\end{bmatrix} = \begin{bmatrix}P_{nx}&P_{ox}&P_{ax}\\P_{ny}&P_{oy}&P_{ay}\\P_{nz}&P_{oz}&P_{aZ}\end{bmatrix} \tag{92}$$

In order to make tool axis correspond to the z axis, we will have

$$(P_{ax}\ , P_{ay}\ , P_{az}) = (S\phi S\theta\ ,\ S\phi C\theta\ ,\ C\phi) \tag{93}$$

Equation (93) can be solved by using

$$\theta = ATANG2(P_{ax}\ ,\ P_{ay}) \qquad 0 \le \theta \le 2\pi \tag{94}$$

$$\phi = ATANG2(SIGN(P_{ay})\ \sqrt{P_{ax}^2 + P_{ay}^2}\ ,\ P_{az}) \qquad -\frac{\pi}{2} \le \phi \le \frac{\pi}{2} \tag{95}$$

As the rotation axis does not always intersect the $I$ axis, let us introduce the temporary coordinate system (TCS), as shown in Figure 37(b), where the origin is the crossing point between $R$ axis and table plane. In TCS, suppose that the difference vector of $I$ axis from the origin within yz plane and the position vector of the origin in the absolute coordinate system (ACS) are $S_i$ and $S_a$ respectively [8].

The coordinate values of the tool center point after the rotational movement, $P'$, in ACS is described as follows:

$$P' = M(S_i - S_a)E^x(\phi)M(-S_i)E^z(\theta)M(S_a)P \tag{96}$$

where M(a) is a matrix concerning the translational movement by a vector $a$ and $E^x(\phi)$ and $E^z(\theta)$ are a matrix concerning the rotational movements around $x$ and $z$ axis, by the angles $\phi$ and $\theta$ respectively.

**< Type 2 >**

As an example of type 2, let us consider such a structure as illustrated in Figure 38 (a). At first, the rotation angles, $\theta$ and $\phi$, are determined in the similar manner as Type 1. The motion can be expressed in terms of a drive transformation matrix $D$. We will then represent D as

$$I = D \cdot P \qquad (98)$$

$$D^{-1} = P \qquad (99)$$

where $P$ is homogeneous matrix of offset points, $I$ is the identity transform, and $\mathbf{E}^y(\phi)$ and $\mathbf{E}^x(\theta)$ are matrix concerning to rotation movement around $Y$ and $X$ axis by an angle $\phi$ and $\theta$ respectively.

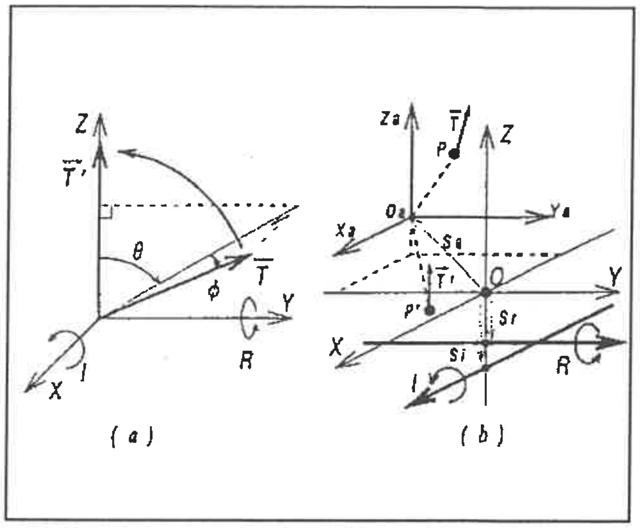


**Figure 38** Table-tilting type with a tilting table on a rotation one [8]

$$D = E^y(\phi)E^x(\theta) \qquad (97)$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C\phi & 0 & S\phi \\ 0 & 1 & 0 \\ -S\phi & 0 & C\phi \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & C\theta & -S\theta \\ 0 & S\theta & C\theta \end{bmatrix} \cdot \begin{bmatrix} P_{nx} & P_{ox} & P_{ax} \\ P_{ny} & P_{oy} & P_{ay} \\ P_{nz} & P_{oz} & P_{aZ} \end{bmatrix} \qquad (100)$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C\phi & S\theta S\phi & C\theta S\phi \\ 0 & C\theta & -S\theta \\ -S\phi & S\theta C\phi & C\theta C\phi \end{bmatrix} \cdot \begin{bmatrix} P_{nx} & P_{ox} & P_{ax} \\ P_{ny} & P_{oy} & P_{ay} \\ P_{nz} & P_{oz} & P_{aZ} \end{bmatrix} \qquad (101)$$

To make tool axis correspondent to $Z$ axis, we will have

$$(P_{ax}\ ,\ P_{ay}\ ,\ P_{az}) = (-S\phi\ ,\ C\phi S\theta\ ,\ C\phi C\theta) \qquad (103)$$

Equation (103) can be solved by using

$$\begin{bmatrix} C\phi & 0 & -S\phi \\ S\phi S\theta & C\theta & C\phi S\theta \\ S\phi C\theta & -S\theta & C\phi C\theta \end{bmatrix} = \begin{bmatrix} P_{nx} & P_{ox} & P_{ax} \\ P_{ny} & P_{oy} & P_{ay} \\ P_{nz} & P_{oz} & P_{aZ} \end{bmatrix} \qquad \textbf{(102)}$$

$$\theta = ATANG2(P_{ay}\ ,\ P_{az}) \qquad -\frac{\pi}{2} \le \theta \le \frac{\pi}{2} \qquad \textbf{(104)}$$

$$\phi = ATANG2(-P_{ax}\ ,\ \sqrt{P_{ay}^2 + P_{az}^2}\ ) \qquad 0 \le \phi \le 2\pi \qquad \textbf{(105)}$$

The temporary coordinate system *xyz* is set, whom the intersection of the two rotation axes is the origin of the *xy* plane, and the table plane is the origin of *Z* axis. Supposing that $S_r$ is the position vector of the *R* axis in the *Z* direction, $S_i$ the position vector of the *I* axis, and $S_a$ the position vector of the origin in ACS, ***P'*** is described as follows [8]:

$$P' = M(S_r - S_a)E^y(\phi)M(S_i - S_r)E^x(\theta)M(S_a - S_i)P \qquad \textbf{(106)}$$

### 3.2.2.2 Spindle-tilting types

The spindle-tilting type machining centers are divided into two types:

Type 1: having a rotational axis around *Z* and *Y* axis

Type 2: having a rotational axis around *Z* and *X* axis(see Figure 39)

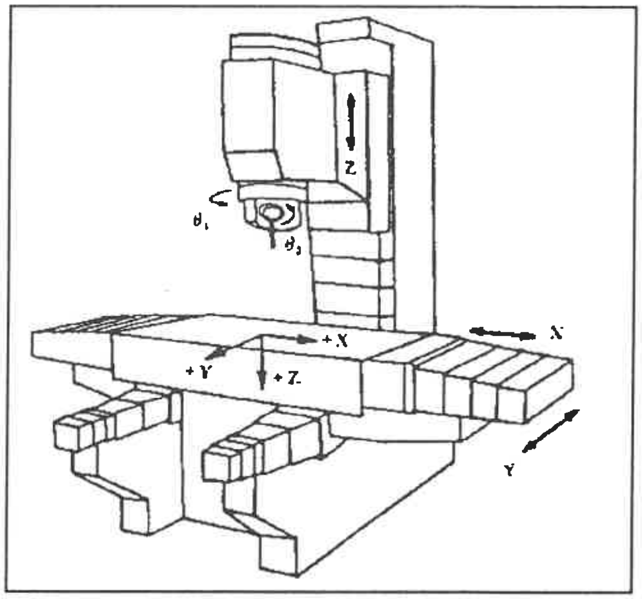


**Figure 39** Fixed bed type of machine [8]

The control methods of this 5-axis CNC machine tools are very similar to those of industrial robots. One of the simplest way to change from one transform to other is by a straight line translation and a rotation about same fixed axis in space. If we can find such a line and axis then we can produce a motion of controlled linear and angular velocity. We will, however, develop a system in which the motion is made in terms of a translation and two rotations. The first rotation will serve to align the tool in the required final direction and the second rotation will control the orientation of the tool about the tool axis. As all manipulators end in a rotary joint, this second rotation in space corresponds to a rotation of the final joint of the manipulator [5].

If A and B (i.e. $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$) are homogeneous matrices representing starting and ending configurations of a machine tool with respect to a reference coordinate system, the straight line motion from the starting configuration to the ending configuration is expressed in terms of a drive transformation matrix D(h), $0 \leq h \leq 1$ as

$$T = A \cdot D(h) \tag{107}$$

A and B express the columns of A and B as vectors $A_n$, $A_o$, $A_a$, $A_P$, and $B_n$, $B_o$, $B_a$, $B_P$, such that

$$A = \begin{bmatrix} A_{nx} & A_{ox} & A_{ax} & A_{Px} \\ A_{ny} & A_{oy} & A_{ay} & A_{Py} \\ A_{nz} & A_{oz} & A_{aZ} & A_{Pz} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{108}$$

$$B = \begin{bmatrix} B_{nx} & B_{ox} & B_{ax} & B_{Px} \\ B_{ny} & B_{oy} & B_{ay} & B_{Py} \\ B_{nz} & B_{oz} & B_{aZ} & B_{Pz} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{109}$$

At the start of motion T=A. The motion D(0) is obtained as D(0)=*I* and the end of the motion as T=B. The matrix D(1) is obtained as

$$D(1) = A^{-1} \cdot B \tag{110}$$

or

$$D(1) = \begin{bmatrix} A_n.B_n & A_n.B_o & A_n.B_a & A_n.(B_p - A_p) \\ A_o.B_n & A_o.B_o & A_o.B_a & A_o.(B_p - A_p) \\ A_a.B_n & A_a.B_o & A_a.B_a & A_a.(B_p - A_p) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{111}$$

We will choose intermediate values of *D* representing a translation and two rotations. Both the translation and rotations will be directly proportional to $h$ so that if $h$ varies linearly with respect to time the motion represented by D will correspond to a constant linear and two constant angular velocities. The translation will be along the line joining A and B and will be represented by the transformation T(h). The first rotation will serve to rotate the approach vector, the direction in which the tool is pointing from A into the approach vector at B. This rotation will be represented by $E^{kz}(\theta_1,h\theta_2)$. The second rotation will rotate the orientation vector representing the orientation of the tool, from A into the orientation vector at B, $E^z(h\phi)$. We will then represent D(h) as

$$D(h) = T(h) \times E^{kz}(\theta_1,h\theta_2) \times E^z(h\phi) \tag{112}$$

where T(h) interpolates the position of the tool end from $A_p$ to $B_p$ and as h varies linearly from 0 to 1.

$$T(h) = \begin{bmatrix} 1 & 0 & 0 & hx \\ 0 & 1 & 0 & hy \\ 0 & 0 & 1 & hz \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{113}$$

where

$$(x, y, z) = [B_p - A_p]^T \tag{114}$$

**< type 1 >**

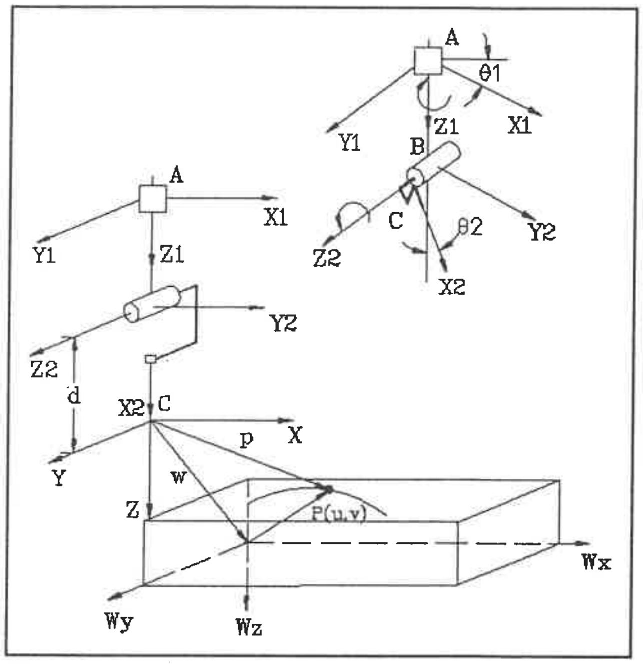


**Figure 40** Spindle-tilting type with a rotation axis around z and y axis

In this type of 5-axis machine tool, we have a rotational axis around z and y axis. $\mathbf{E}^{kz}(\theta_1,h\theta_2)$ represents a rotation of $\theta_2$ (y axis) about a unit vector $\mathbf{k}=(\sin\theta_1,\cos\theta_1,0)^T$ to change its z-axis from A to B (see Appendix A). where $\mathbf{B_p} = \mathbf{p} = \mathbf{W} + \mathbf{P}(u,v)$, $W$ is translation displacement of the workpiece with respect to a fixed coordinate system, $P(u,v)$ is the desired path with respect to the fixed coordinate system, and $\mathbf{A_p}$ is the starting configuration of the machine tool (see Figure 40) :

$$E^{kz}(\theta_1,h\theta_2) = \begin{bmatrix} S\theta_1^2V(h\theta_2)+C(h\theta_2) & -S\theta_1C\theta_1V(h\theta_2) & C\theta_1S(h\theta_2) & 0 \\ -S\theta_1C\theta_1V(h\theta_2) & C\theta_1^2V(h\theta_2)+C(h\theta_2) & S\theta_1S(h\theta_2) & 0 \\ -C\theta_1S(h\theta_2) & -S\theta_1S(h\theta_2) & C(h\theta_2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(115)}$$

where

$$V(h\theta_2) = Versine(h\theta_2) = 1-\cos(h\theta_2) \quad \textbf{(116)}$$

$$C(h\theta_2) = \cos(h\theta_2) \quad \textbf{(117)}$$

$$S(h\theta_2) = \sin(h\theta_2) \qquad \textbf{(118)}$$

$E^z(h\phi)$ twists the tool about its z-axis vector to change the y-axis vector from Ao to Bo.

$$E^z(h\phi) = \begin{bmatrix} C(h\phi) & -S(h\phi) & 0 & 0 \\ S(h\phi) & C(h\phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(119)}$$

where

$$S(h\phi) = \sin(h\phi) \qquad \textbf{(120)}$$

$$C(h\phi) = \cos(h\phi) \qquad \textbf{(121)}$$

The right two columns of D(h) are

$$D(h) = \begin{bmatrix} ? & ? & C\theta_1 S(h\theta_2) & hx \\ ? & ? & S\theta_1 S(h\theta_2) & hy \\ ? & ? & C(h\theta_2) & hz \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(122)}$$

If we set h = 1, we may solve for x, y, z, $\theta_1$, $\theta_2$, and $\phi$ as follows:

$$D(1) \times E^z(\phi)^{-1} \times E^{kz}(\theta_1, \theta_2)^{-1} = T(1) \qquad \textbf{(123)}$$

$$x = A_n \cdot (B_p - A_p) \qquad \textbf{(124)}$$

$$y = A_o \cdot (B_p - A_p) \qquad \textbf{(125)}$$

$$z = A_a \cdot (B_p - A_p) \qquad \textbf{(126)}$$

and

$$T(1)^{-1} \times D(1) \times E^z(\phi)^{-1} = E^{kz}(\theta_1, \theta_2) \qquad \textbf{(127)}$$

$$C\theta_1 \cdot S\theta_2 = A_n \cdot B_a \qquad \textbf{(128)}$$

$$S\theta_1 \cdot S\theta_2 = A_o \cdot B_a \qquad \textbf{(129)}$$

$$C\theta_2 = A_a \cdot B_a \qquad \textbf{(130)}$$

Thus

$$\theta_1 = ATANG2((A_o \cdot B_a)\ ,\ (A_n \cdot B_a)) \qquad 0 \le \theta_1 \le 2\pi \qquad \textbf{(131)}$$

and

$$\theta_2 = ATANG2(\sqrt{(A_n \cdot B_A)^2 + (A_O \cdot B_a)^2}\ ,\ (A_a \cdot B_a)) \qquad -\frac{\pi}{2} \le \theta_2 \le \frac{\pi}{2} \qquad \textbf{(132)}$$

and finally

$$E^{kz}(\theta_1, \theta_2)^{-1} * T(1)^{-1} * D(1) = E^Z(\phi) \qquad \textbf{(133)}$$

or

$$\begin{bmatrix} S\theta_1^2V\theta_2+C\theta_2 & -S\theta_1C\theta_1V\theta_2 & -C\theta_1S\theta_2 & 0 \\ -S\theta_1C\theta_1V\theta_2 & C\theta_1^2V\theta_2+C\theta_2 & -S\theta_1S\theta_2 & 0 \\ C\theta_1S\theta_2 & S\theta_1S\theta_2 & C\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times D(1) = \begin{bmatrix} C\phi & -S\phi & 0 & 0 \\ S\phi & C\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(134)}$$

then

$$S\phi = -S\theta_1C\theta_1V\theta_2(A_n \cdot B_n)+(C\theta_1^2V\theta_2+C\theta_2)(A_o \cdot B_n)-S\theta_1S\theta_2(A_a \cdot B_n) \qquad \textbf{(135)}$$

$$C\phi = -S\theta_1C\theta_1V\theta_2(A_n.B_o)+(C\theta_1^2V\theta_2+C\theta_2)(A_o.B_o)-S\theta_1S\theta_2(A_a.B_o) \qquad \textbf{(136)}$$

and

$$\tan\phi = \frac{S\phi}{C\phi} \qquad -\pi \leq \phi < \pi \qquad \textbf{(137)}$$

In mode absolute, where the point coordinates and orientations define relation to coordinate system of machine (not relation to last point), we have

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(138)}$$

If d is the length of tool, we will have (see Figure 40),

$$x = -B_{px}+d \cdot C\theta_1 \cdot S\theta_2 \qquad \textbf{(139)}$$

$$y = -B_{py} + d \cdot S\theta_1 \cdot C\theta_2 \tag{140}$$

$$z = B_{pz} - d \cdot C\theta_2 + d \tag{141}$$

and

$$\theta_1 = ATANG2(B_{ay}, B_{ax}) \qquad 0 \le \theta_1 \le 2\pi \tag{142}$$

and

$$\theta_2 = ATANG2(-SIGN(B_{ax})\sqrt{(B_{ax})^2 + B_{ay})^2}\ ,\ B_{az}) \qquad -\frac{\pi}{2} \le \theta_2 \le \frac{\pi}{2} \tag{143}$$

**< Type 2 >**

In this type of 5-axis machine tool, we have a rotational axis around z and x axis. $E^{kz}(\theta_1, h\theta_2)$ represents a rotation of $\theta_2$ (x axis) about a unit vector $\mathbf{k}=(\mathrm{con}\theta_1, \cos\theta_1, 0)^T$ to change its z-axis from A to B (see Appendix A). Thus $E^{kz}(\theta, h\theta_2)$ is given by

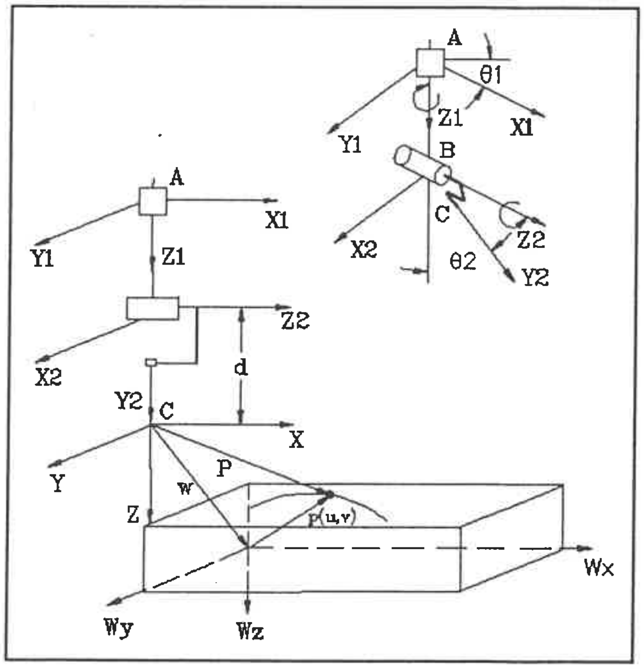


**Figure 41** Spindle-tilting type with a rotation axis around z and x axis

$$E^{kz}(\theta,h\theta_2)=\begin{bmatrix} S\theta_1^2V(h\theta_2)+C(h\theta_2) & C\theta_1C\theta_1V(h\theta_2) & S\theta_1S(h\theta_2) & 0 \\ -S\theta_1C\theta_1V(h\theta_2) & S\theta_1^2V(h\theta_2)+C(h\theta_2) & -C\theta_1S(h\theta2) & 0 \\ -S\theta_1S(h\theta_2) & C\theta1S(h\theta2) & C(h\theta_2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(144)}$$

and with the same method as type 1 we obtain values for x, y and z (see Figure 41)

$$x = A_n \cdot (B_p - A_p) \qquad \textbf{(145)}$$

$$y = A_o \cdot (B_p - A_p) \qquad \textbf{(146)}$$

and

$$S\theta_1 S\theta_2 = A_n \cdot B_a \qquad \textbf{(148)}$$

$$-C\theta_1 S\theta_2 = A_o \cdot B_a \qquad \textbf{(149)}$$

$$C\theta_2 = A_a \cdot B_a \qquad \textbf{(150)}$$

Thus

$$\theta_1 = ATANG2((A_n \cdot B_a), -(A_o \cdot B_a)) \qquad 0 \leq \theta_1 < 2\pi \qquad \textbf{(151)}$$

and

$$\theta_2 = ATANG2(-SIGN(A_o \cdot B_a)\sqrt{(A_o \cdot B_a)^2 + (A_a \cdot B_a)^2}\ , A_o \cdot B_a) \quad -\frac{\pi}{2} \le \theta_2 \le \frac{\pi}{2} \qquad \textbf{(152)}$$

and in mode absolute

$$x = -B_{px} + d \cdot S\theta_1 S\theta_2 \qquad \textbf{(153)}$$

$$y = -B_{py} - d \cdot C\theta_1 S\theta_2 \qquad \textbf{(154)}$$

$$z = B_{pz} - d \cdot C\theta_2 + d \qquad \textbf{(155)}$$

and

$$\theta_1 = ATANG2(B_{ax}, -B_{ay}) \qquad 0 \le \theta_1 < 2\pi \qquad \textbf{(156)}$$

and

$$\theta_2 = ATANG2(-SIGN(B_{ay})\sqrt{(B_{ax})^2 + B_{ay})^2}\ , B_{az}) \qquad -\frac{\pi}{2} \le \theta_2 \le \frac{\pi}{2} \qquad \textbf{(157)}$$

### 3.2.3 5-axis rough cutting

Although three axis is desirable for rough cutting, 5-axis control is often required for workpiece shapes with multi-values relating $z$ direction. The rough cutting tool path is generated by shifting the interference-free finishing tool path by an amount of depth of cut in the direction of tool axis vector. The shift operation is repeated times with the number depth of cut. The tool center location in rough cutting *P'* is given as follows:

$$P' = P + i \cdot d \cdot T \quad (i=1,\ldots,f) \tag{158}$$

where ***P***, ***T***, *d* and *f* are the tool center location of finishing tool path, the tool axis vector, depth of cut and the number of depth of cut respectively.

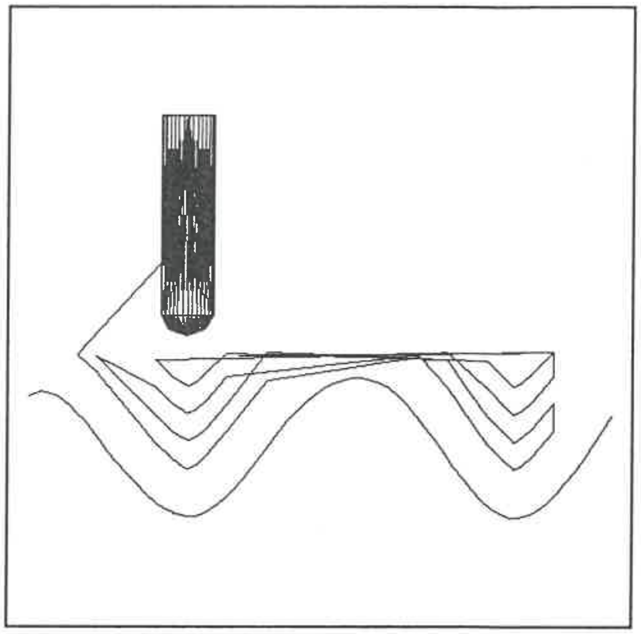


**Figure 42** Number of depth of cut in rough cutting

Another method developed in this project consist of the possibility to shift the tool path by an amount of depth of cut in the direction of *z* axis vector. if there is no collision between surface and tool. In this case the points that their *z* value is more than $z_{max}$ are eliminated (see Figure 42).

$$f = \frac{z_{\max} - z_{\min}}{d} \tag{159}$$

$$P' = P + i \cdot d \cdot z \tag{160}$$

In this case, it is possible to use a tool with greater diameter to decrease the time of machining.

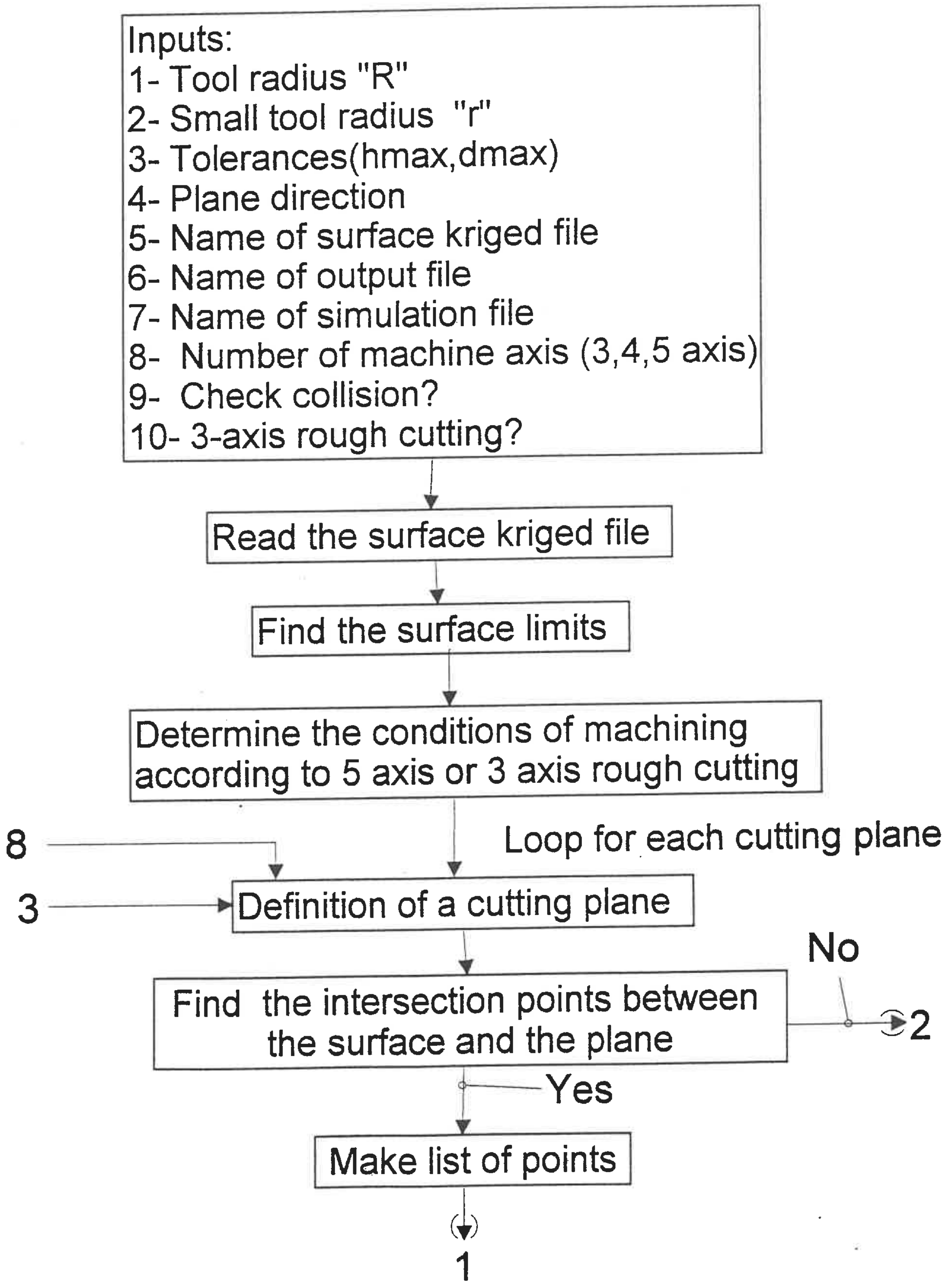
## ALGORITHM

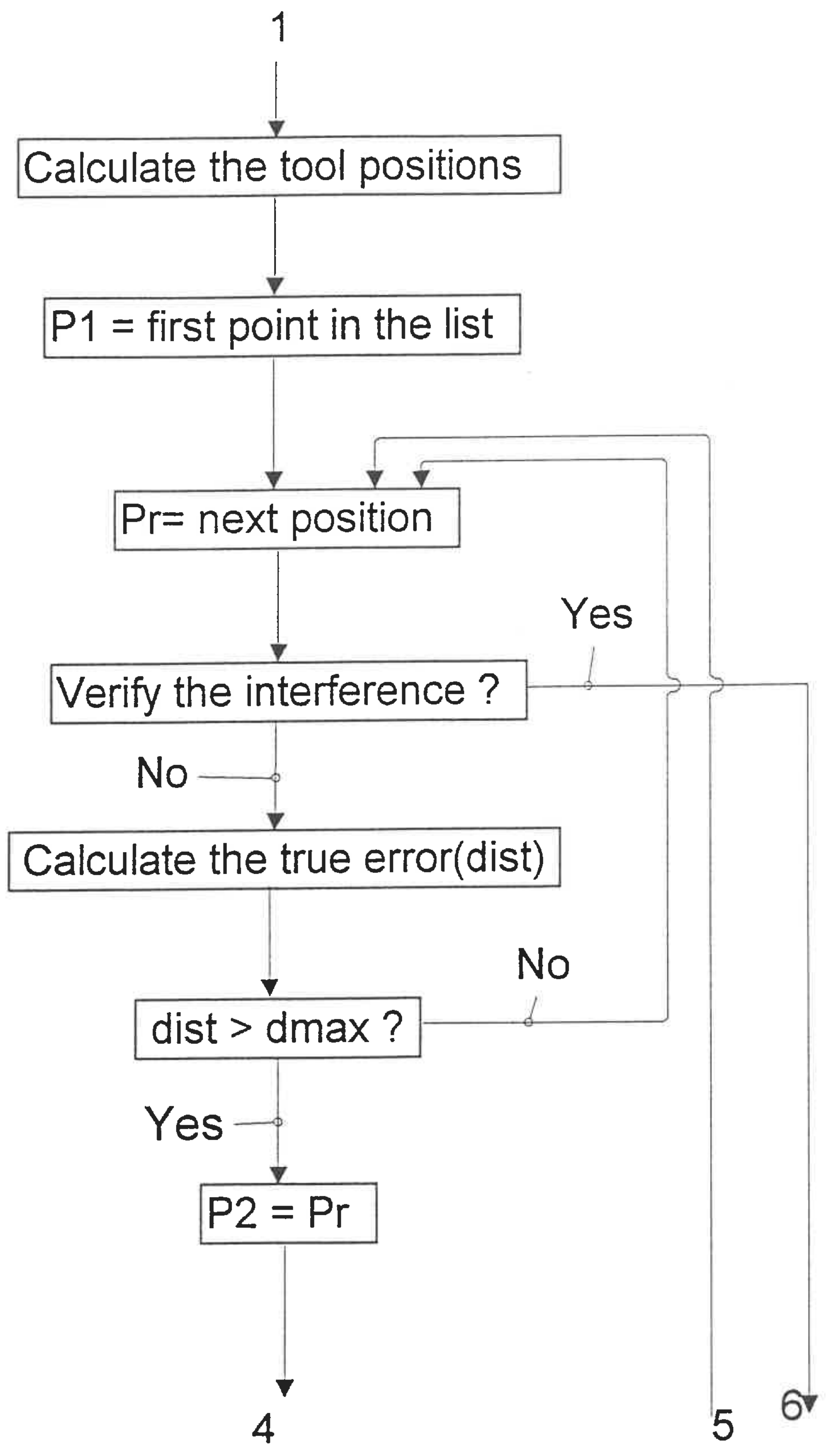
The algorithm shown in the following pages is developed in the program NC_KRIGE (nckrige.c). The krigeag subroutine, KG_IntersectPlaneSurf is used to generate a list of initial points in each cutting curve. The number of intersection points is specified in the intps.h file with the variable #defined Max_iso . This variable has to be determined depending of the given tolerance and the area of the surface.

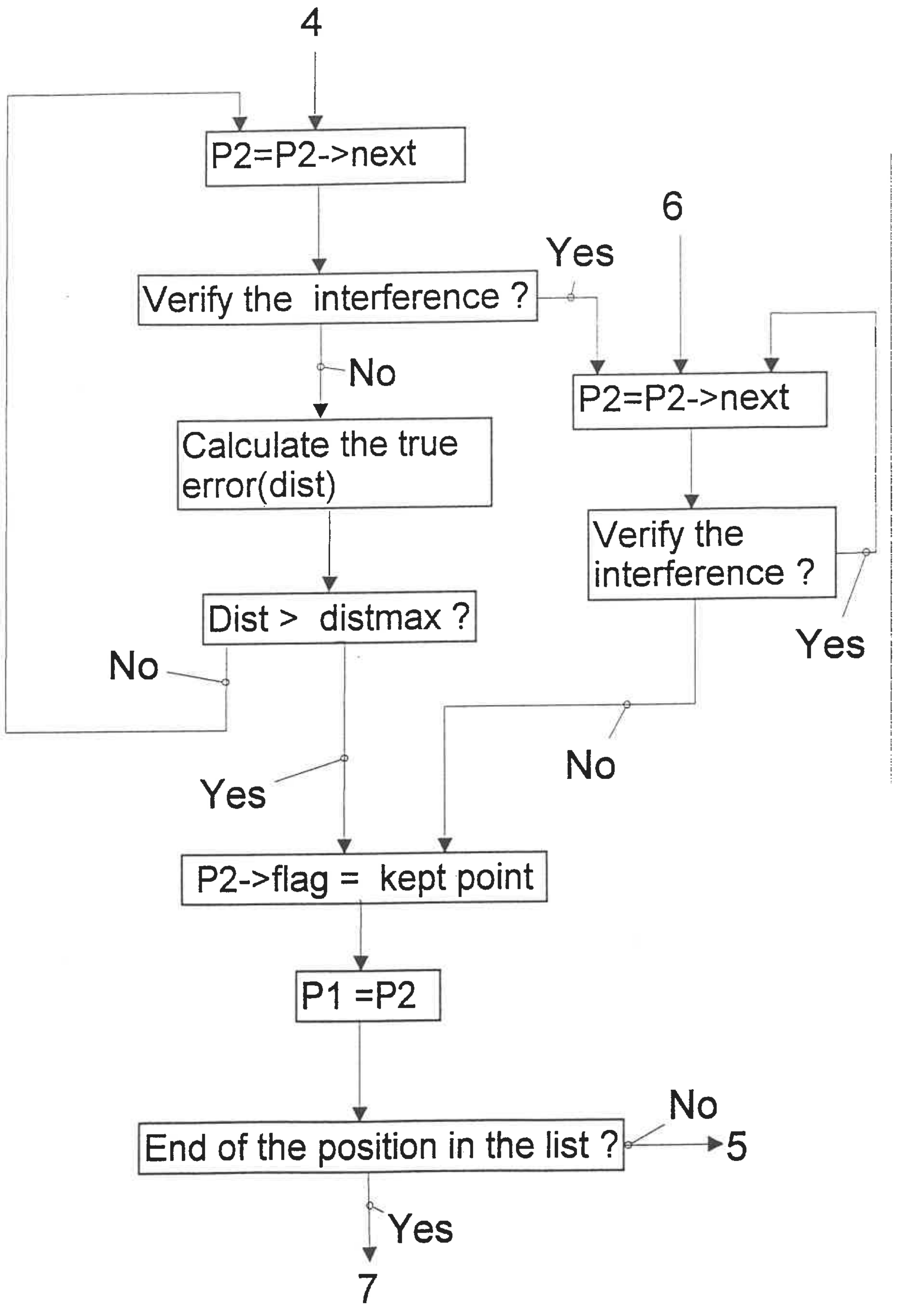
A series of functions are used to create a list of the tool positions. For more details on these functions refer to Appendix B (reference manual of LIBKNC library). These functions are used to create new path generation algorithms.

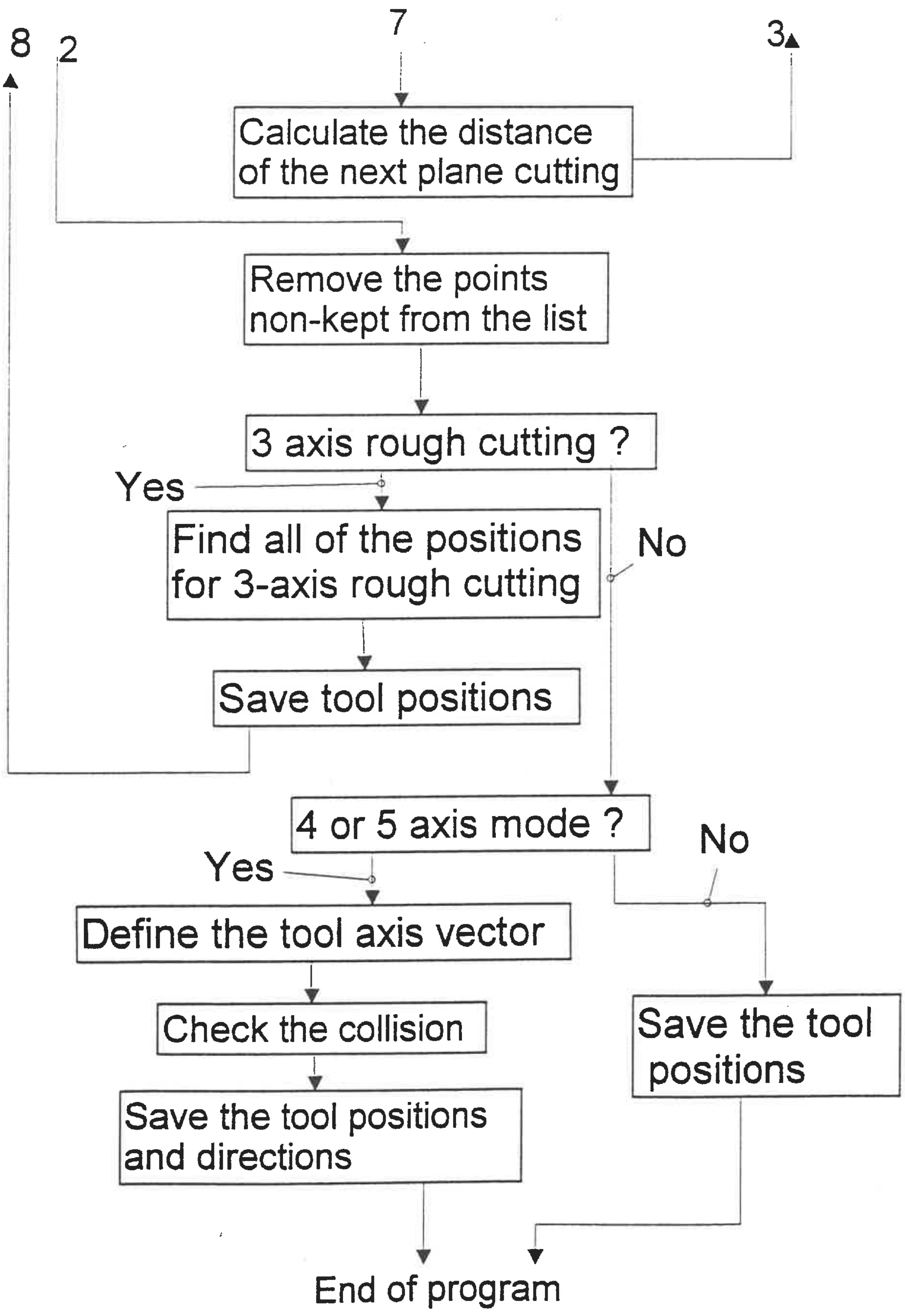
The program calculates a tool path on a kriged surface (see Appendix C). The surface finish is dependant of the given tolerance ($h_{max}$ and $d_{max}$). The program uses method 2; section 2.2 and the method discussed in section 2.5 (CalcCuspHeight) to calculate $d_{max}$ and $h_{max}$ respectively. The program verifies simultaneously all interferences with the subroutine CheckInterference. In the case of 4 or 5 axis machining, after determining the tool axis vectors by the method described in section 2.8, the subroutine CheckCollision is used to verify the collision between the tool and the surface by method 4 described in section 3.1.1. If necessary, the program uses the subroutine 3_AxisRougCutting to generate a path for rough cutting in three axis machining.

The results of NC_krig program are saved in an ASCII file that is used by the post-processor. Finally, the Finallist of the program is also saved to be used in the simulation program.


Inputs:
1- Tool radius "R"
2- Small tool radius "r"
3- Tolerances(hmax,dmax)
4- Plane direction
5- Name of surface kriged file
6- Name of output file
7- Name of simulation file
8- Number of machine axis (3,4,5 axis)
9- Check collision?
10- 3-axis rough cutting?
Read the surface kriged file
Find the surface limits
Determine the conditions of machining according to 5 axis or 3 axis rough cutting
Loop for each cutting plane
8
3
Definition of a cutting plane
No
Find the intersection points between the surface and the plane
2
Yes
Make list of points
1


1
Calculate the tool positions
P1 = first point in the list
Pr= next position
Verify the interference ?
Yes
No
Calculate the true error(dist)
dist > dmax ?
No
Yes
P2 = Pr
4
5
6


4
P2=P2->next
Verify the interference ?
Yes
No
6
Calculate the true error(dist)
P2=P2->next
Verify the interference ?
Dist > distmax ?
No
Yes
Yes
No
P2->flag = kept point
P1 =P2
End of the position in the list ?
No
5
Yes
7



8
2
7
3
Calculate the distance of the next plane cutting
Remove the points non-kept from the list
3 axis rough cutting ?
Yes
No
Find all of the positions for 3-axis rough cutting
Save tool positions
4 or 5 axis mode ?
Yes
No
Define the tool axis vector
Check the collision
Save the tool positions and directions
Save the tool positions
End of program

## SIMULATION

It is very important to ensure the correctness of NC programs before they are used in regular production, to avoid errors that can cause damage to the machine tool or workpiece. The traditional verification for NC program has been direct 'proofing runs' on soft materials, which can be expensive and time consuming. So regardless of the method used to generate the NC program, it is generally checked before final milling. Here, the simulation program developed in this project, is executed by using two free-form surfaces and a tool surface. In the case of 3-axis machining the tool axis is constant and we can verify the path of tool center and CC points. In the case of 5-axis machining, the tool surface or free-form surface, depending on the type of machine tool, is translated and rotated according to the *CC* points and tool axis using the equations that described previously. The simulation program (ncsimul.c see Appendix C) permit to see the tool path and the moving of tool during machining in 3-rough cutting and 3 , 4 and 5 axis.

## EXPERIMENTS

Some kriged surfaces are used to validate the algorithms: 1) the complex surface shown in Figure 5, 2) the turbine blade shown in Figure 43, 3) a convex and a concave surface for measuring the precision of surface for machining in 3 axis, the convex and the concave surfaces shown in Figure 44 and 45 for machining in 5 axis, and a wavy surface for verification of interference and collision.

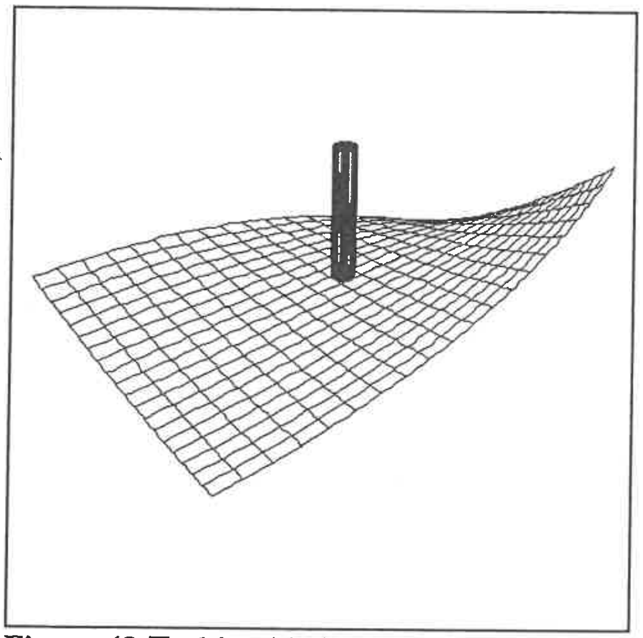


**Figure 43** Turbine blade

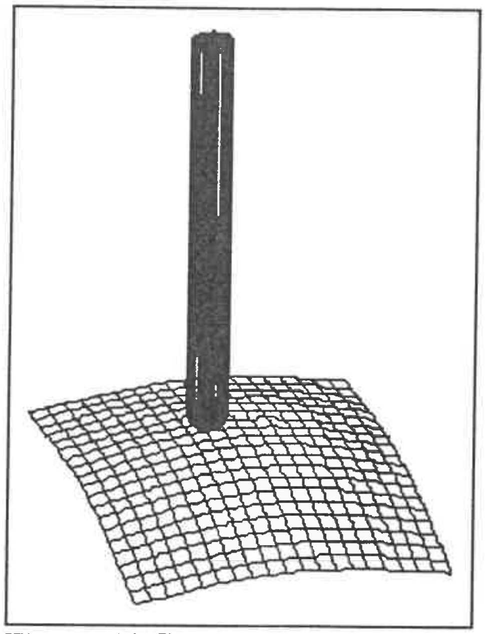


**Figure 44** Convex surface

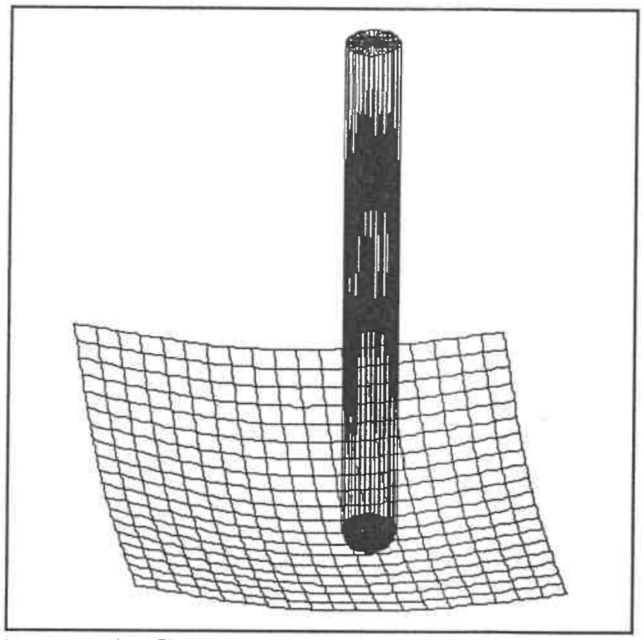


**Figure 45** Concave surface

Figures 46 and 47 show the results of the interference algorithm. The curve above the surface indicates the path of the tool center. In the first case, when the tool follows this curve, there is interference between the tool and the surface. This situation is corrected by the interference algorithm (see Figure 47). It is noted that some regions of the surface are not machined within the given tolerance. These regions must be machined with a smaller tool.

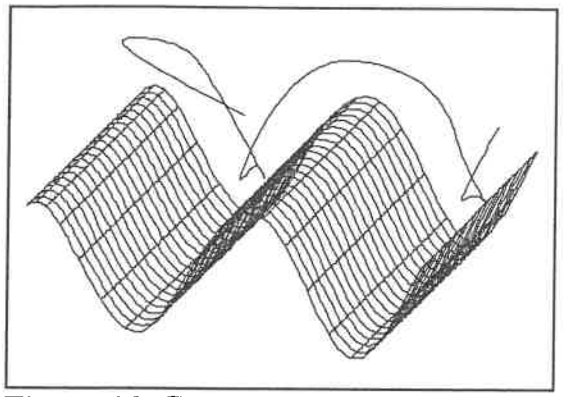


**Figure 46** Curve relative to the tool positions without the interference algorithm.

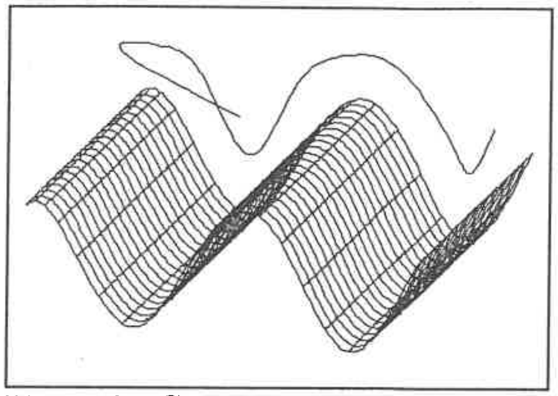


**Figure 47** Curve relative to the tool positions with the interference algorithm.

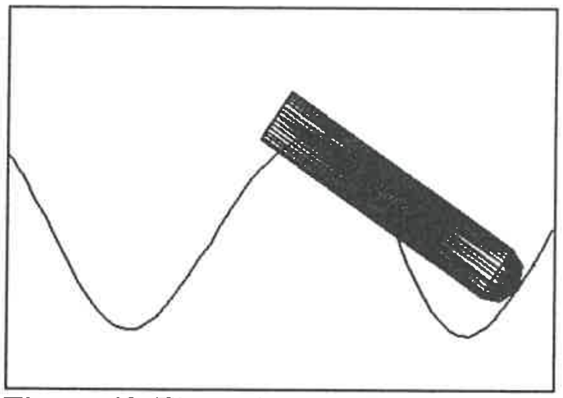


**Figure 48** Five-axis machining without the collision algorithm.

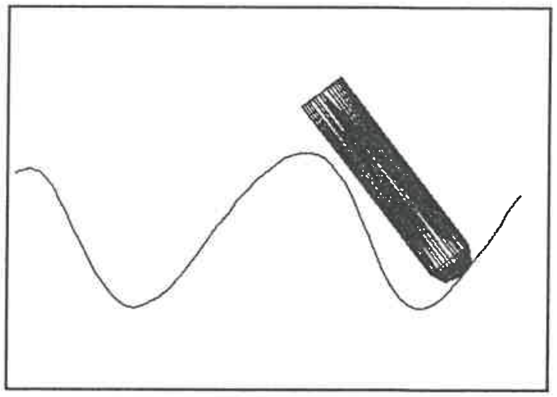


**Figure 49** Five-axis machining with the collision algorithm.

Figure 48 and 49 show the results of the collision algorithm. The surface is machined in 5-axis mode where the tool is in the direction of the surface normal vector of surface. Figure 49 shows that the algorithm can correct the tool axis vector to avoid the collision.

Figure 50 and 51 show the results of the interference algorithm because of the tool shape. Because of the gouging effect, the selected points on a convex surface are different than for a concave surface and we need more points in the case of a convex surface to obtain the same tolerance.

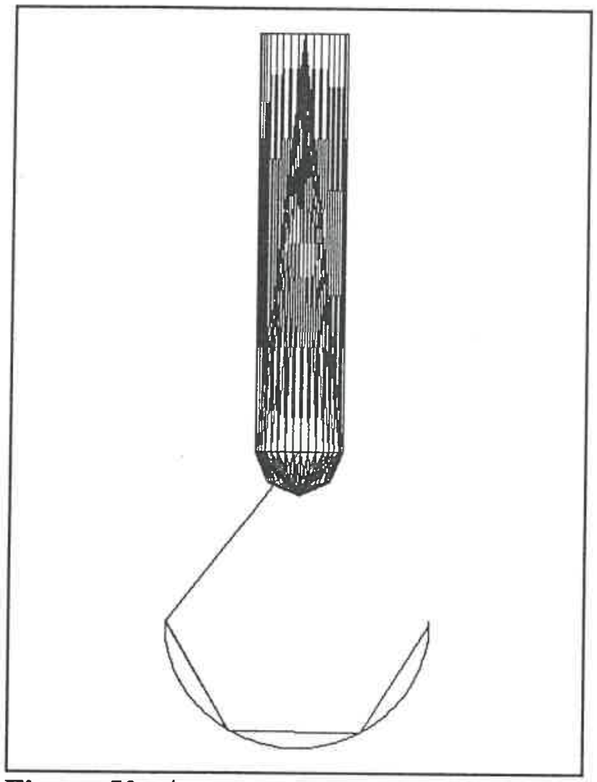


**Figure 50** A concave surface with d= 0.268 and a tool radius R=r=1.0

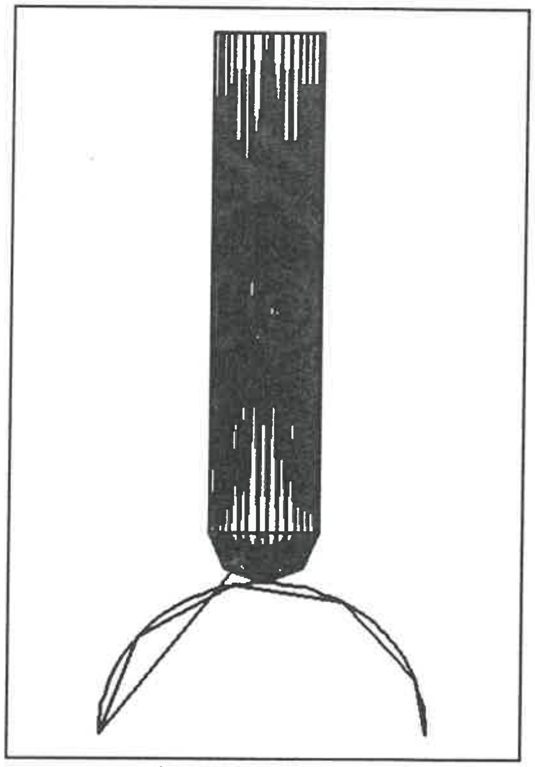


**Figure 51** A convex surface with d = 0.268 and a tool radius R=r=1.0

# CONCLUSIONS AND RECOMMENDATIONS

The CAD/CAM system is developed in this paper to solve the problems of manual programming and controlling the tolerances, when a kriged surface complex is machined in 3, 4, or 5 axis. The cartesian machining method was selected for finding the intersection points using the intersection between the surface and a series of parallel planes. This method decrease the machine time specially for irregular and complex surfaces. The new method is developed to select CC points from intersection points that provide the given tolerances while the interference between the tool and the surface is avoided simultaneously. The two method "zigzag" and "spiral" are developed for rough machining the die cavity. The end-mill and ball-mill are compared according to the tolerances of machining, cutting speed and the accessibility.

In the case of 4 and 5 axis, the new method for avoidance the collision is presented and some suggestions for solving the linearization problem are given. The equations to find the rotary angles of machine axes and the coordinates of tool center are developed according to the type of machine tool. Five axis rough cutting is developed in order to machine a surface with a small depth of cavity.

The new algorithm is executed in the program NC_Krig. This program defines a tool path on a complex surface defined by kriging. This path has the following property:

1. The smoothness of the surface is equal given tolerance $h_{max}$ and $d_{max}$.

2. The points on the path avoid all the interference between the tool and the surface.
3. The collision is checked at each point.
4. The algorithm is valid for the end-mill, the ball-mill, and the torus-mill.
5. Rough cutting in 3 axis can be done with a specific depth.
6. The algorithm is valid for machining in 3, 4 and 5 axis.
7. The simulation program shows the tool path and movements.
8. A post-processor can convert new positions and orientations of the tool to the machine.

This program can be also used to

1. Find an offset surface without interference.
2. Determine the region that was not machined because of the interference.

## Recommendations

1. Study the effects of the tool axis and the type of tool surface on the machined region.
2. Further study on linearization is needed.
3. Use an expert system to select the proper tool size and type.
4. Simulate a workpiece and visualise the final shape and tolerances.
5. Use an environment to make testing easier.

## BIBLIOGRAPHY

1. CHOI. B. K. , JUN. C. S , july/august 1989,"Ball-end Cutter Interference Avoidance in NC Machining of Sculptured Surfaces", Computer-aided design, vol 21 No 6, 371-378.

2. HUANG. Yunching, OLIVER. James H, 1992,"Non-constant Parameter Tool Path Generation on Sculpture Surfaces", Computers in engineering, ASME, Vol. 1, 411-419.

3. KRIGE. D. G, 1951,"A Statistical Approach to Some Basic Mine Valuation Problems on the Witwatersrand", J. Chem. Metal. Min. Soc. S. Afr., S2, 119-139.

4. MATHERON. G, 1973,"The intrinsic random functions and their applications", Adv. Appl, Prob.5, 439-468.

5. PAUL. Richard, November 1979,"Manipulator Cartesian Path Control", IEEE Translations on systems, man, and cybernetics, vol. smc-9, no 11.

6. SUH. Suk-Hwan, LEE. Kee-Sang," Solving Tool-Interference Problem for Four-Axis NC ", Machining Processing of the 1992 IEEE international Conference on Robotics and Automation Nice,France-May 1992 Computer Automated Manufacturing Lab, 790-600.

7. SUH. Suk-Hwan, LEE. Kee-Sang ,"A Prototype CAM System for Four-Axis NC

Machining of Rotational-Free-Surfaces", Journal of manufacturing systems, vol 10, No.4.

8. TAKEUCHI. Yoshimi, WATANABE. Takahiro, 1992,"Generation of 5-Axis Control Collision-Free Tool Path and Postprocessing for NC Data" ,Annals of the CIRP vol 41/1/1992, 539-542.

9. TROCHU. F, 1993,"Presentation of a Contouring Program Based on Dual Kriging Interpolation", Engineering with computers, Vol 9, 160-177

10. VIKERS. G. W., QUAN. K. W., 1989," Ball-Mills Versus End-Mills for Curved Surface MaChining ", Transaction of the ASME, Vol. 111, 22-26.

11. CHO, H. D., JUN, Y. T. and YANG, Y.,1993,"Five_axis CNC Milling for Effective Machining of Sculptured Surface", INT. J. PROD. RES., Vol. 31, No.11, 2559-2573.

12. CHOPU, Jui-jen and YANG, D. C. H, 1991,"Coordinated Motion Generation of Five-axis CNC/CMM Machines in CAD/CAM Integration", Sensors, Controls, and Quality Issues in Manufacturing, ASME 1991, 151-161.

13. TöNSHOFF, H. K., HERNANDEZ-CAMACHO, J.,1989," Die Manufaturing by 5-axes Milling", Journal of Mechanical Working Technology, 20, 105-119.

14. PAUL, R. P, 1981,"Robot Manipulators: Matematics, Programming, and Control" MIT Press. Cambridg, Massachuestts.

15. PAUL. Richard, November 1975,"Manipulator Path Control", Proc. IEEE Int. Conf. cybernetics and Society, 147-152.

16. CHOU, Jui-jeu and DYANG, D. C. H., February 1992,"On the Generation of Coordinates Motion of Five-Axis CNC/CMM Machinee", Journal Engineering for industry, Vol. 114, 15-22.

17. CHEN, Y. J.,and RAVANI, B.,1987,"Offset Surface Generation and Contouring Computer-aided Designe", ASME Journal of Mechanisms, Transmissions, and Automation in Designe, 109, 133-142.

18. YAKEUCHI, Y. and IDEMURA, T. 1991,"5-Axis Control Machining and Grinding Based on solid model", CIRP Annals, 40/1/1992, 455-458.

19. OLIVER, J. H.,WYSOKI, D. A. and GOODMAN, E. D., February 1993,"Gouge Detection Algorithms for Sculptured Surface NC Generation", Journal of Engineering for Industry, Vol. 115, 139-144.

20. CHI, B. K., PARK, J. W. and JUN, C. S., june 1993,"Cutter location Data Optimization in 5-axis Surface Machining", Computer-aided design, Vol. 25, No. 6, 377-386.

21. LEE, An-chen, CHEN, Da-pan, and LIN, Chinh-lung, 1990,"A CAD/CAM system from 3D coordinate measuring data", INT. J. PROD. DES., Vol. 28, No 12, 2353-2371.

22. SUH, Suk-Hwan and LEE, Kee-sang, 1992," Solving Tool-Interference Problem for Four-Axis Nc Machining", Proceeding of the 1992 IEEE International Conference on Robatics and Automation Nine, France.

23. SUH, Yong Seok and LEE, Kunwoo, 1990,"NC Milling Tool Path Generation for Arbitry Pockets Defined by Sculptured Surface", Computer-Aided Design, Vol. 22 , No. 5, 273-284.

24. CHOI, B. K., LEE, C. S., HWANG, J. S. and JUN, C. S., 1988,"Compound Surface Modeling and Machining", Computer-Aided Design, VOL. 20, No 3, 127-136.

25. LEE, Y. S., CHOI, B. K., and CHANG, T. C., 1991,"Cut Ditribution and Cutting Selection for Sculptured Surface Cavity Machining", INT. J. PROD. RES., Vol 30, No 6, 1447-1470.

26. LONEY, Gregory C., and OZSOY, Tulga M., 1987," NC Machining of Free Form Surfaces", Cmputer-Ainded Designe, VOL. 19, No 2. 85-90.

27. BOBROW, James E, 1985,"NC Machine Tool Path Generation from CSG Part Representations", Computer-Aided Design, Nol 17, No 2, 69-76.

28. LEE, Yuan-Shin and CHANG, Tien-Chien, 1991," CASCAM - An Automated System for Scculptured Surface Cavity Machining", Computers in Industry, 16, 321-342.

29. SASTRI, J. P., and KUMAR, S. Vasantha, 1986," Computer Aided Manufacturing of Sculptured Surfaces of Impeller Blades on 5 Axes CNC machins", 12th AIMTDR Conference, IIT Dehl, 120-123.

30. KRAMER, Thomas R.," Poket Milling with Tool Engagement Detection", Journal of Manufacturing Systems, Vol 11, No 2, 114-123.

32. YAU, Hong-Tzong, Menq, Chia-Hsiang, 1991," Concurrent Process Planning for Machining and Inspection of Sculptured Surfaces", Transactions of NAMRI/SME,320-326.

33. MARCINAK, Krzysztof, 1991,"Geometric Modelling for Numerically Controlled Machining", Oxford Science Publications.

34. MORTENSON, Michael E., 1985,"Geometric Modeling",John Wiley & Sons.

35.FAUX, I. D., and PRATT, M. J.,1980,"Computational Geometry for Design and Manufacturing", Ellis Horwood Publishers.

36. VIKERS, G. W. ,LY, M. H., and OETTER, R. G.," Numerically Controlled

Machine Tools", Ellis Horwood Publishers.

37. KIM, Kwangsoo and BIEGEL, John E., 1988,"A Path Generation Method for Sculptured Surface Manufature", Computes ind. Engng, Vol. 14, No 2, 95-101.

38. KIM, Kwangsoo and BIEGEL, John E., 1988,"An Integrated Approach to Sculptured Surface Design and Manufacture", Computes ind. Engng, Vol. 14, No 3, 271-280.

39. TANG, K., WOO, T., and GAN, J., 1992,"Maximum Intersection of Spherical Polygons and Workpiece Orientation for 4- and 5- Axis Machining", Journal of Mechanical Design, Vol 114, 477-485.

40. ZHU, Cui, 1990,"Avoiding Iterference in Manufacturing a Free-Formed Surface with a Cylindrical End Milling Cutter", Computers in Industry, 14, 367-371.

41. ZHU, Cui, 1991,"Tool-path Generation in Manufacturing Sculptured Surface with a Cylindrical End Milling Cutter", Computers in Industry, 17, 385-389.

42. MARCINAK, Krzysztof, 1987,"Influence of Surface Sahape on Admissible Tool Positions in 5-Axis Face Milling", Computer-Aided Design, Vol 19, No. 5, 233-236.

43. BROOMHEAD, P. and EDKINS, M., 1986,"Generating NC data at the Machine Tool for the Manufacture of Free-Form Surfces", INT. J. PROD. RES., Vol 24, No.

1, 1-14.

44. MASON, Frederick, November 1991,"5 x 5 for High-productivity Airfoil Milling", American machinist, 37-39.

45. KIMMEL, R. and BRUCKSTEIN, M. A., 1993,"Shape Offsets Via Level Sets", Computer-Aided Design, Vol. 25, No. 3, 154-162.

46. February 1987,"5-axis Model Making Saves Time and Money", Production Engineer.

47. BALA, M. and CHANG, T.-C., 1991,"Automatic Cutting Selection and Optimal Cutting Path Generation for Prismatic Parts", INT. J. PROD. RES., Vol. 29, No 11, 2163-2176.

48. RAMARAJ, T. C., ELEFTHERIOU, Eleftheriou, and RAMARAJ, Rama, 1989,"Integration of Design and Manufacture of Complex Geometries Through Solid and Surface Modeling Techniques", Journal of Mechanical Working Technology, 20, 141-152.

49. TROCHU, F. and LAROCQUE, S., 1992,"Présentation d'un Logiciel de Modélisation Géométrique par Krigeag Dual", Huitiéme Congrés Canadien de L'education en Ingénierie, 278-285.

## APPENDIX A: Rotate a frame about a vector

In order to rotate a frame about a vector rotated of an angle $\theta_1$ from the y or orientation vector of a coordinate frame. we will first develop a transform function H(K, $\theta_2$) to rotate any transform X about a vector k and angle $\theta_2$ [5].

$$Y = H(K \cdot \theta_2) \times X \tag{161}$$

We will perform the rotation about the vector k by constructing a coordinate frame C with k as the z unit vector, and then rotate the angle $\theta_2$ about the z axis of C.

The frame represented by X with respect to base coordinates is equal to a frame Z with respect to coordinate frame C. Thus we have

$$X = C * X \tag{162}$$

or

$$Z = C^{-1} * X \tag{163}$$

The frame Z rotated about the z axis and angle $\theta_i$ is given by

$$Z = E^{z}(\theta_2) \tag{164}$$

where

$$E^{z}(\theta_2) = \begin{bmatrix} C\theta_2 & -S\theta_2 & 0 & 0 \\ S\theta_2 & C\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{165}$$

The rotated Z frame is with respect to the coordinate frame C. In order to relate it to base coordinates we must pre-multiply by C

$$Y = C * E^{z}(\theta_2) * Z \tag{166}$$

and substituting from (A3)

$$Y = C * E^{z}(\theta_2) * C^{-1} * X \tag{167}$$

The transform function **H(k,** $\theta_2$**)** is then

$$H(K,\theta_2) = C * E^{z}(\theta_2) * C^{-1} \tag{168}$$

where C is a coordinate frame with k as the z axis. Evaluating the expression for **H(k,** $\theta_2$**)** symbolically yields

$$H(K,\theta_2) = \begin{bmatrix} K_x K_x V\theta_2 + C\theta_2 & K_y K_x V\theta_2 - K_z S\theta_2 & K_z K_x V\theta_2 + K_y S\theta_2 & 0 \\ K_x K_y V\theta_2 + K_z S\theta_2 & K_y K_y V\theta_2 + C\theta_2 & K_z K_y V\theta_2 - K_x S\theta_2 & 0 \\ K_x K_z V\theta_2 - K_y S\theta_2 & Ky_z V\theta_2 + K_x S\theta_2 & K_z K_z V\theta_2 + C\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{169}$$

In the first case the vector **k** is obtained by rotating the y axis of a coordinate frame an angle $\theta_2$ about the z axis. Thus **k** is given by

$$\begin{bmatrix} -S\theta_1 \\ C\theta_1 \\ 0 \end{bmatrix} = \begin{bmatrix} C\theta_1 & -S\theta_1 & 0 \\ S\theta_1 & C\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{170}$$

and **H(k,** $\theta_2$**)** simplifies to

$$H(K,\theta_2) = \begin{bmatrix} S\theta_1^2V\theta_2+C\theta_2 & -S\theta_1C\theta_1V\theta_2 & C\theta_1S\theta_2 & 0 \\ -S\theta_1C\theta_1V\theta_2 & C^2\theta_1+C\theta_2 & S\theta_1S\theta_2 & 0 \\ -C\theta_1S\theta_2 & -S\theta_1S\theta_2 & C\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(171)}$$

In the second case the vector **k** is obtained by rotating the x axis of a coordinate frame an angle $\theta_2$ about the z axis. Thus **k** is given by

$$\begin{bmatrix} C\theta_1 \\ S\theta_1 \\ 0 \end{bmatrix} = \begin{bmatrix} C\theta_1 & -S\theta_1 & 0 \\ S\theta_1 & C\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad \textbf{(172)}$$

and **H(k,** $\theta_2$**)** simplifies to

$$H(K,\theta_2) = \begin{bmatrix} c\theta_1^2V\theta_2+C\theta_2 & S\theta_1C\theta_1V\theta_2 & S\theta_1S\theta_2 & 0 \\ S\theta_1C\theta_1V\theta_2 & S^2\theta_1+C\theta_2 & -C\theta_1S\theta_2 & 0 \\ -S\theta_1S\theta_2 & C\theta_1S\theta_2 & C\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(173)}$$

# Appendix B: Reference manual of the LIBKNC library

## MANIPULATION OF DATA OF PATH

The algorithm for selection the *CC* points and the algorithms that foresee the interference and collision use repeatedly the normal and the offset of the tool center in the calculations. For this reason, a list of positions is created that contains all the necessary information during the program. The information of the list is also used for validation and visualisation algorithms.

A doubly linked list is defined with the following structure :

```
typedef struct ncpos_c NCPos_c;

struct ncpos_c                  /* STRUCTURE for a position on the surface */
{
        double    puv[2];       /* parameters (u,v) of the position */
        Precis_t  pxyz[3];      /* coordinates (x,y,z) */
        Precis_t  vnrml[3];     /* unit vector of normal */
        Precis_t  vaxis[3];     /* unit vector of tool axis orientation */
        Precis_t  poffset[3];           /* coordinates of the tool-center offset */
        NCPos_c   *prev;                /* previous position in the list */
        NCPos_c   *next;                /* next position in the list */
        PosFlag   flag;         /* position is ON or OFF */
};
```

Each position first contains the parameters *(u,v)* of the position on the surface and the correspondent cartesian coordinates *(x,y,z)*. The unit vector of the normal, the unit values that define the orientation of tool axis and the relative positions of tool center are also recorded in each position. Two pointers, **prev* and **next*, are used to move through the list. Finally, a index *flag* is needed in the algorithms for define the points that must be removed from the list.

A group of procedures are used to create, modify, manipulate and record the lists of positions. These procedures are described in the Appendix D in the programme *nc.c*. The following examples show how it can be used:

```
NCPos_c        *list1;
Precis_t
.
.
.
KG_IntersectSurfPlane ( surf, a, b, c, d, npoints, int_points );
list1 = NC_MakeListFromXYZ ( surf, int_points, npoints );
```

The procedure *NC_MakeListFromXYZ* will create a list of position from the coordinates *(x,y,z)* which from the intersection between the plane and surface. The normals will be calculated at each position. Then, we calculate the positions relative of the tool and change the orientation of the tool axis for 5 axis machining:

```
NC_CalcOffset(list1, R, r);
NC_Set5axisInfo(list1);
```

The information of each position can be modified very easily. For example, to change the index *flag* of each position in the list, one can write:

```
pos = list1;                    /* began from the appointed position by list1 */
do {
        pos->flag = PON;                /* give the value PON to flag in the position pos */
        pos = pos->next;                /* go to the next position in the list */
} while (pos);                  /* continue until pos=NULL */
```

It is also possible to search for the positions containing a special information. The next example search all the positions with *flag* == *PON* and removes them:

```
pos = list1;            /* began to seek the first position */
while (pos) {           /* loop until pos is no-NULL */
        pos = NC_SearchForFLAG(pos,PON);
                        /* seek the next position where flag=PON, and return
                          this position */
        if (pos) pos = NC_DeletePos(&list, pos);
                        /* if this position is no-NULL, remove it
                          and return the new location of the position */
}
```

Several lists can be created independently to joint with a principal list. For example, if two lists (*list1* and *list2*) were created, the second can be joined to the other in inverse order:

```
NC_AppendListToList(list1, &list2, 1);    /* Note: the 1 appoint the inverse order */
```

The above examples show how it is possible to create and manipulate the list of positions. The information of a list can be extracted and saved for using in the simulator or post-processor. For example,

```
        NC_OutputSampSet("points.bin", list1);
```

save in the binary file, a *sampset* of *offset* points from the list *list1*.

# Appendix C: Lists of programs: nckrige.c, nc.c, ncsimul.c, nc.h

```
/*****************************************************************************
 NC_Krige:
 ~~~~~~~~
        Calculates the tool path for machining complexe surfaces using kriging.

        Written by:     Abbas Vafaeesefat
                        Serge Gravelle

        Ecole Polytechnique, Universite de Montreal, December 15 1993
 *****************************************************************************/

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

#include <libkg.h>
#include <libkgl.h>
#include <libkap.h>
#include <mymacros.h>

#include "nc.h"
#include "intps.h"


/*PROTOTYPES */

NCPos_c*Threeaxisrough(NCPos_c *finallist, Precis_t *homepos,Precis_t dd);
void     Define_vaxis_EM(NCPos_c *finallist);

void      Check_Collision(Handle_t surf, NCPos_c *list, Precis_t R,Precis_t r);
int    CheckInterference(NCPos_c *list, Precis_t R,Precis_t r);
void      Calcule_Cusp_h(Handle_t surf, NCPos_c *list, Precis_t *normal_plan,
     Precis_t R, Precis_t h, Precis_t *point);
void    DefinePlane(Precis_t *normal, Precis_t *point, Precis_t *a,
     Precis_t *b, Precis_t *c, Precis_t *d);
Precis_t CalcD(NCPos_c *pr,  NCPos_c *p1, Precis_t  R);
char*InputName(char *text, char *name);
doubleInputFloat(char *text, double dflt);

/* GLOBAL VARIABLES */


Precis_tplannorm[3]={1.,0.,0.};
Precis_tdistmax2=0.02,distmax,distmax1=0.02;



int main( )
{

/* DEFAULT VARIABLES */

Precis_t
R = 0.1875,
r  = 0.1875,
               h  = 0.02,
RR,
rr,
hh,
theta = 0.,
homepos[3] = {0., 0., 4.};

charname[20] = "bouteil.bin",
outname[20] = "nc_3axis.dat",
listname[20] = "nc_3axis.list",
three_axis_rough[20] = "t_axis_rough.dat",
list_axis_rough[20] = "t_axis_rough.list";

Switch_tfiveaxismode=OFF,
```

```
                                        t_axis_rough=OFF,
                                        CheckCollision=OFF;

/* VARIABLE DEFINITION */

      NCPos_c                           *list,
                                              *lastlist,
                                              *finallist1,*finallist,
                                              *pos, *p1, *p2, *pr;


      Handle_t                          surf,
                                              samp;

      Precis_t                          *result,
                                              point1[3], *point1uv,
                                              a, b, c, d,dd=r,thick,
                                              thick1=0.0,thick2=0.0,
                                              dist,
                                              ui, vi;
      int                               alpha,
                                              i,m,j,
                                              CI = 1,
                                              k  = 0;

      FILE                              *fp;

      float                             input;

      char                              *name2, *name1,*name3,*name4,*name5;


/* INPUT DATA FROM USER (doesn't work yet) */

            printf("\n\n\nPOLYFAB: Usinage automatique de surfaces complexes");
            printf(" par krigeage parametrique\n~~~~~~~\n");
            R = InputFloat("        Radius (R)",R);
            r = InputFloat("        radius (r)",R);
            h = InputFloat("        cusp height (h)",h);
            distmax1 = InputFloat("  chord deviation  distmax = ", distmax1);
            thick1=InputFloat("     thickness =",thick1);
     theta = InputFloat("       normal orientation (in degrees)",theta);
     plannorm[0] = cos(theta*3.1415926/180);
     plannorm[1] = sin(theta*3.1415926/180);
     plannorm[2] = 0.;
     name1 = InputName("        surface filename",name);
     name2 = InputName("        data filename",outname);
     name3 = InputName("        list filename",listname);
            input = InputFloat("     3 axis rough cutting (0-OFF, 1-ON)", t_axis_rough);
     t_axis_rough = (input) ? ON : OFF;
            if(t_axis_rough == ON){
            dd = InputFloat("               depth of cut =",dd);
            thick2=InputFloat("             thickness =",thick2);
            RR = InputFloat("               Radius (R)",R);
            rr = InputFloat("               radius (r)",R);
     distmax2 = InputFloat("            chord deviation  distmax = ", distmax2);

            hh = InputFloat("               cusp height (h)",h);
     name4=InputName("          data filename",three_axis_rough);
     name5=InputName("          list filenamt",list_axis_rough);
                                                                        }
            input = InputFloat("     5 axis mode (0-OFF, 1-ON)",fiveaxismode);
            fiveaxismode = (input) ? ON : OFF;
     if(fiveaxismode==ON){
            input = InputFloat("     check collision (0-OFF, 1-ON)",CheckCollision);
     CheckCollision = (input) ? ON : OFF;}

     printf("\n-----------------------------------------\n");
     printf("SUMMARY:\nTOOL:  R = %f  r = %f  \n",R,r);
```

```
      printf("TOLERANCES:  h = %f  distmax = %f  plane normal = (%f, %f, %f)\n
                                                  ", h, distmax, plannorm[0],plannorm[1],plannorm[2])
 ;
      printf("surface in: %s  output data in: %s\n",name1, name2);
      printf("five axis mode is: %s\n", ((fiveaxismode) ? "ON" : "OFF") );
           printf("check collision is: %s\n", ((CheckCollision) ? "ON" : "OFF"));
           printf("3 axis rough cutting is: %s\n\n", ((t_axis_rough) ? "ON" : "OFF"));
                if(t_axis_rough==ON){

      printf("\nCASE FOR 3 AXIS ROUGH CUTTING:\n\nR = %f  r = %f  ",RR,rr);
      printf("\nh = %f  distmax = %f \n", hh, distmax2);
           printf("Depth of cut is: %d\n",d);

                                                                             }
       printf("----------------------------------------\n\n");

 /* READ SURFACE FROM FILE */

        if((fp = fopen(name1, "rb")) != NULL)
                {
                        KG_SetInputFile(fp);
                        if(!( surf = KG_LoadSurf()))
                        printf("error ");
                fclose(fp);
                }
                else
                printf("error...");


        if(t_axis_rough ==ON)
                                                                      {
                                                                              m=2;
                                                                      }else{m=1;}
        for(j=0;j<m;j++)
                 {

/* MAKE LIST OF STARTING POSITION (HOME POSITION) */

   finallist = NC_CreateList();
   for(i=0;i<3;i++) finallist->pxyz[i] = homepos[i];


/* INTERSECT PLANES WITH SURFACE AND REMOVE UNWANTED POINTS  */

   /* STARTING POINT ON SURFACE AT u=0, v=0 */

   NC_FindLimits( surf, plannorm, 20, &ui, &vi);
   KG_InterpolSurf( surf, ui, vi, 0, 0, point1);

   printf("\n Starting point (%f,%f,%f)\n\n",point1[0],point1[1],point1[2]);

        if(t_axis_rough ==ON && !j){h=hh;R=RR;r=rr; distmax = distmax2;thick=thick2;}
                                                                                       else

{h=h;R=R;r=r;distmax = distmax1;thick=thick1;}


/* LOOP FOR EACH CUTTING PLANE */

        for(i=0;i<111;i++)
                        {


/* DEFINE PLANE AND INTERSECT, PUT RESULT IN A NEW LIST */

                DefinePlane(plannorm,point1,&a,&b,&c,&d);
                printf("\nPlan: %fx + %fy + %fz + %f = 0\n",a, b, c, d);

                if (IntersectPlaneSurf(surf,a,b,c,d,&alpha,&result)
                        != KGSUCCESS) {KG_PrintError( "No Intersection" ) ;break;}
            printf("%d points of intersection between plane and surface\n",alpha);
            if (!alpha) {printf("\n 0 points of intersection ");  break;}

            list = NC_MakeListFromXYZ(surf,result,alpha);

            free (result);


/* CALCULATE OFFSETS IN THIS LIST */

        NC_CalcOffset(list, R, r,thick);

            if (fiveaxismode==ON) NC_Set5axisInfo(list);


/*  TOOL PATH GENERATION */

                p1 = list;

                while(p1->next)
                {
                k = 0;
                pr = p1;
                pr->flag=PON;

                while(pr->next)
                    {
                    pr = pr->next;
                    CI = CheckInterference(pr,R,r);
                    if (!CI) {pr=pr->prev; k=1; p2=pr; break;}
                    dist = Abs(CalcD(pr,p1,r));
                    if (dist > distmax) { pr=pr->prev; break;}
                    }

                if(!(pr->next)) { pr->flag=PON; break;}


                if (CI) {
                    p2 = pr;
                    while(p2->next)
                                    {
                                    p2 = p2->next;
                                    CI = CheckInterference(p2, R,r);
                                    if (!CI) {p2=p2->prev; k=k+1; break;}
                                dist = Abs(CalcD(pr,p2,r));
                                    if (dist > distmax) {p2=p2->prev; break;}
                                    }
                                }
                if (k){
                                do {
                                        p2 = p2->next;
                                        CI = CheckInterference(p2, R,r);
                                        if(CI)  break;
                                    } while(p2->next );
                            }

            p2->flag = PON;
            p1 = p2;
            if(!(p1->next)) break;
            }


/* CALCULATE CUSP HEIGHT TO FIND POINT IN NEXT CUTTING PLANE  */

    Calcule_Cusp_h(surf, list, plannorm, R, h, point1);
    printf ("Point in new plane = (%f,%f,%f)\n",
    point1[0],point1[1],point1[2]);
```

```
/* REMOVE UNEEDED POINTS FROM LIST, ADD TO THE FINAL LIST*/

        pos = list;
   while (pos) {
                                              pos = NC_SearchForFLAG(pos,POFF);
                                              if (pos) pos = NC_DeletePos(&list,pos);
                                              }

        lastlist = finallist;
        while (lastlist->next) lastlist = lastlist->next;

/* last argument changes list direction for zigzag */

        NC_AppendListToList(finallist, &list, (i-(i/2)*2));

                        }

        if(t_axis_rough ==ON && !j){

finallist1=Threeaxisrough(finallist,homepos,dd);
                                                                                  NC_WriteList(name5,
finallist1);

NC_Output3axis(name4, finallist1);
                                                                                  }


                }
/* VERIFY PATH FOR COLLISION */

        if (fiveaxismode==ON){
                         if(r<R)Define_vaxis_EM(finallist);
                      if(CheckCollision==ON)Check_Collision(surf,finallist, R,r);
                           NC_Output5axis(name2, finallist);
                                                                  }
                                                                  else
                                                     {NC_Output3axis(name2, finallist);}
if(r<R)NC_CalcOffset(finallist, R, r,thick1);
NC_WriteList(name3,finallist);
return 0;
}


/************************************************************************/
/* SUBROUTINES */
/************************************************************************/


/************************************************************************

     Function   Verifies if there is interference between the tool
                                 and the surface.

     Syntax   CheckInterference(NCPos_c *list, Precis_t R ,Precis_t r)

     Remarks   'list' expects a list structure
               'R' expects the radius R of the tool
                                  'r' expects the small radius of the tool

               Return value   Returns a 0 if an interference was found.
                   Returns a 1 if no interference exists.

************************************************************************/


int     CheckInterference(NCPos_c *list, Precis_t R ,Precis_t r)
```

```
{
        int                         CI = 1,j,k,n=0;
        Precis_t    distance[3],s,a,b,dist1,
                                            alpha=1,theta,pi=3.141592654;
        NCPos_c     *pos;

for(k=0;k<2;k++)
        {
        if(CI)
                        {
                                pos = list;
                                do
            {
                                        if(k){pos = pos->next;}else{pos = pos->prev;}
            for(j=0;j<3;j++)distance[j] = pos->pxyz[j] - list->poffset[j];
            dist1 = sqrt(Dotprod(distance,distance));
             for(j=0;j<3;j++){distance[j]=distance[j]/dist1;}
            for(j=0;j<3;j++)distance[j]=-1*distance[j];
                                        alpha= acos(Dotprod(distance,list->vaxis));
                                        if(r){theta=atan((R/r)-1.);}else {theta = pi/2;}
                                        if(alpha>=theta && alpha<=pi/2)
                                        {

s=(R-r)*sin(alpha)+sqrt(Sqr(R-r)*Sqr(sin(alpha))-(Sqr(R)-2*R*r));
                                        if(dist1< s){CI = 0;break;}
                                        }
                                        else
                                        if(alpha<theta){
                                   s=r/cos(alpha);
                                        if(dist1<s){CI = 0; break;}
                                                                                    }else
                                        if(alpha>=pi/2){
                                for(j=0;j<3;j++)distance[j] = pos->pxyz[j] - list->poffset[j];
                                        b= Sqr(dist1)-Sqr(Dotprod(distance,list->vaxis));
            if(b< Sqr(R+0.0000001)) {CI = 0; break;}
                                                                                        }
                                        }while(CI,pos && Sqr(list->poffset[0]-pos->pxyz[0])+
                                                    Sqr(list->poffset[1]-pos->pxyz[1])<=Sqr(R));

                        }
                }
return CI;
}
/**********************************************************************

     Function   Calculates the distance 'd' which will be compared
                to 'dmax' in order to find the position P2.

     Syntax    CalcD(NCPos_c *pr,  NCPos_c *p1,  Precis_t r)


     Remarks   'pr' the changing point
               'p1' the starting point
               'r'  the radius of the tool

Note that the distance calculated includes the correction distance
'g' which compensates for the tool radius.

 Return value   Returns the distance d.

 **********************************************************************/


Precis_t        CalcD(NCPos_c *pr,  NCPos_c *p1,  Precis_t r)

{
        Precis_t        *c, s, *cc, d, g;
        int                     i;
```

```
                c  = Crossprod(pr->vnrml,plannorm);
                cc = Crossprod(plannorm,c);

                for(i=0;i<3;i++)  c[i] = p1->pxyz[i] - pr->pxyz[i];

                d = Abs(Dotprod(c,cc));

                g = r * (1 - (Dotprod(pr->vnrml, p1->vnrml)));

                for(i=0;i<3;i++)  c[i] = pr->pxyz[i] - p1->pxyz[i];

                if (Dotprod(c, p1->vnrml) > 0)  g = -g;

        free (c);
        free (cc);
return (d + g);
}

/*******************************************************************

     Function   Calculates the parameters for an algebraic plane given
                                a unit normal and a point.

       Syntax   DefinePlane(Precis_t *normal, Precis_t *point, Precis_t *a,
                           Precis_t *b, Precis_t *c , Precis_t *d)

      Remarks   'normal' is a pointer to a unit normal vector
                'point' is a pointer to the coordinates of a
                                        point on the plane

                                'a', 'b', 'c', 'd'  are pointers to the parameters for the
                                          algebraic equation of a plane:
                         a*x + b*y + c*z + d = 0;


 Return value  On successful completion, the parameters will be
               stored at the addresses given to the procedure.

 *******************************************************************/

void    DefinePlane(Precis_t *normal, Precis_t *point, Precis_t *a, Precis_t *b,
                        Precis_t *c, Precis_t *d)

{
        Precis_t   length;

        length = sqrt(Dotprod(normal,normal));

        *a = normal[0]/length;
        *b = normal[1]/length;
        *c = normal[2]/length;

        *d = -(*a) * point[0] - (*b) * point[1] - (*c) * point[2];
}

/*******************************************************************

     Function   Calculates the distance to the next parallel plane
                 so that the cusp height does not exceed hmax.

       Syntax   CalcCuspHeight(Handle_t surf, NCPos_c *list,
                                 Precis_t *normal_plan, Precis_t R, Precis_t h, Pr ecis_t *point)

      Remarks   'surf' is a handle to the kriged surface
                'list' is a pointer to the beginning of the list of positions
                'normal_plan' is a pointer to the plane normal vector
                'R' is the radius of the tool
                'h' is the maximum cusp height allowed (hmax)
                'point' is the pointer to where the coordinates of the
                point calculated should be stored.

 Return value   Stores the coordinates of the point which will be
                used to define the next plane.

 ******************************************************************/



void  Calcule_Cusp_h(Handle_t surf, NCPos_c *list, Precis_t *normal_plan,
                                        Precis_t R, Precis_t h, Precis_t *point)
{

        Precis_t                lmin = 999.9, l, t = 0.2;
        Precis_t                result[3], ppoint[3], puv[2], *normal_pp;
        Precis_t                theta, p, s,DOC,AOB,pi=3.141592654;
        int                     i, out_flag;
        NCPos_c         *pos, *posmin;

        pos = list;
        pos = pos->next;
        do {

        if(pos->flag == PON)
                {
                for(i=0;i<3;i++) ppoint[i] = t * normal_plan[i] + pos->pxyz[i];
                KG_ProjectPointOnSurf(surf, ppoint, 70, 3, &out_flag, result);
                if(!(out_flag)) printf("\n erreur de projection...Calcule_Cusp...\n");
                KG_xyz2uv(surf, result, &puv[0], &puv[1]);
                normal_pp = KG_InterpolSurfNormalOnUV(surf, puv, 1);
                theta  = acos(normal_pp[0] * pos->vnrml[0]+
                                                normal_pp[1] * pos->vnrml[1]+
                                                normal_pp[2] * pos->vnrml[2]);
                s = sqrt( Sqr(pos->pxyz[0]-result[0]) +
                                        Sqr(pos->pxyz[1]-result[1]) +
                                        Sqr(pos->pxyz[2]-result[2]) );
                p = s / tan(theta);

/* for a concave surface */

                if((result[0]-pos->pxyz[0])*pos->vnrml[0]+
                        (result[1]-pos->pxyz[1])*pos->vnrml[1]+
                        (result[2]-pos->pxyz[2])*pos->vnrml[2]
                        > 0 ){
                                l =  (p/((p-R)*(p-h))) * sqrt(4*Sqr(p-R)*
                                                Sqr(p-h) - Sqr(Sqr(p) - 2*R*p+Sqr(p-h)));
                        DOC = acos(l/(2*p));
                        AOB = acos(Dotprod(pos->vnrml,normal_plan));
                                        l= l * cos(AOB-DOC);
                                        }
/* for a convax surface */

                                 else{
                                l =  (p/((p+R)*(p+h))) * sqrt(4*Sqr(p+R)*Sqr(p+h)
                                                - Sqr(Sqr(p) + 2*R*p+Sqr(p+h)));
              DOC = acos(l/(2*p));
              AOB = acos(Dotprod(pos->vnrml,normal_plan));
                                        l = l * cos(DOC+AOB-pi);
                                        }

        if (l > 2*R || p>599.)  l = 2*sqrt(2*R*h-Sqr(h));

                if ( l <= lmin ) { lmin = l; posmin = pos; }
                }
        } while (pos=pos->next);


        for(i=0;i<3;i++) point[i] = posmin->pxyz[i] + normal_plan[i]*lmin;
        free(normal_pp);
        free(result);

}
/*******************************************************************

     Function   Verifies if there are any collisions between the
                tool and the surface at each position in the list .

       Syntax   CheckCollision(Handle_t surf, NCPos_c *list, Precis_t R)

      Remarks   'surf' is a handle to the kriged surface.
                'list' is a pointer to the beginning of the list of positions
                'R' is the radius of the tool

Note, if a collision is found, corrections are done to the orientation vector at that position.

 Return value   none.

 *******************************************************************/


void  Check_Collision(Handle_t surf, NCPos_c *list, Precis_t R,Precis_t r)
{
        Precis_t   m[3],dist[3],ui,vi,point[3],
                                                  s,a,b,ppoint[50][50][3];
        int         i,j,n=50,k;
        NCPos_c     *pos;


        for(i=0;i<n;i++)
        {
        for(j=0;j<n;j++)
         {
         pos = list;
         ui = i*1.0/(n-1.0);
         vi = j*1.0/(n-1.0);
         KG_InterpolSurf( surf, ui, vi, 0, 0, point);
                         for(k=0;k<3;k++) ppoint[i][j][k]=point[k];

                 }
     }
   do{
     for(i=0;i<n;i++)
     for(j=0;j<n;j++)
          {
        for(k=0;k<3;k++) point[k]=ppoint[i][j][k];
                              if(r<R){
                              for(k=0;k<3;k++)
                                       pos->poffset[k] = pos->pxyz[k] + (pos->vnrml[k])*R;
                                      }

            for(k=0;k<3;k++) dist[k] = point[k] - pos->poffset[k];

            if(Dotprod(dist,pos->vaxis) > 0.){

            b= Sqr(sqrt(Dotprod(dist,dist)))-Sqr(Dotprod(dist,pos->vaxis));
            if(b< Sqr(R+0.0000001))
            {
             if(pos->vaxis[2]<1.) pos->vaxis[2] = pos->vaxis[2] + 0.2;
             s = sqrt(Dotprod(pos->vaxis, pos->vaxis));
             for(k=0;k<3;k++) pos->vaxis[k] =  pos->vaxis[k] / s ;
            i=0;
            break;

            }

       }
          }
       }while(pos=pos->next);

 }


 /*********************************************************************

      Function   Inputs a string of text.  If none are entered, the
                                       default is used.

        Syntax   *InputName(char *text, char *name)


       Remarks   'text' is text of the question to be printed to the screen
                                       'name' is the default string

                 Return value   Returns the string of characters.

 *********************************************************************/


 char  *InputName(char *text, char *name)
 {
         char    *input;
         MALLOC (input, char, 25);

         printf("\n%s [%s] > ",text,name);
         gets(input);
         if (*input==NULL) return name;
         else return input;
 }
 /*********************************************************************

      Function   Inputs a floating point number.  If nonthing is entered,
                 the default is used.

        Syntax   InputFloat(char *text, double dflt)

       Remarks    'text' is the text of the question to be printed to the screen
                  'dflt' is the default number

  Return value   Returns the number.

 *********************************************************************/


 double  InputFloat(char *text, double dflt)
 {
         char    input[25];

         printf("\n%s [%f] > ",text,dflt);
         gets(input);

         if (*input==NULL) {return dflt;}
         else return atof(input);
 }


 /*********************************************************************

      Function   Defines the orientation the tool axis

        Syntax   void  Define_vaxis_EM( NCPos_c  *finallist)

       Remarks   'finallist' expects a list structure


 *********************************************************************/
```

```
void  Define_vaxis_EM( NCPos_c  *finallist)
{

                         Precis_t        pp12[3],l,*A,*p;
                         int             i;
        NCPos_c         *p2,*p1;
   p1=finallist;
   p1= p1->next;
  while(p1->next)
        {
        p2=p1;
        p2 = p2->next;
        for(i=0;i<3;i++){ pp12[i] = p2->pxyz[i] - p1->pxyz[i];}
        l = sqrt(Dotprod(pp12,pp12));
        for(i=0;i<3;i++) pp12[i] = pp12[i] / l;
        A = Crossprod(pp12, p1->vnrml);
        l = sqrt(Dotprod(A,A));
        for(i=0;i<3;i++) A[i] = A[i] / l;
        p = Crossprod(A, pp12);
        l = sqrt(Dotprod(p,p));
        for(i=0;i<3;i++) p1->vaxis[i] = p[i] / l;
        p1=p2;
        }

}

/*******************************************************************

     Function   Verifies if there are any collisions between the
                tool and the surface at each position in the list .

          Syntax  *Threeaxisrough(NCPos_c *finallist ,Precis_t *homepos,
                                                                                            Precis_t dd,Precis
_t thick)

     Remarks   'finallist' is a pointer to the beginning of the list of positions
                'homepos' is the start point of tool
                                          'dd' is depth of cut
                                     'thick' is  the thickness for finishin
 Return value   list of positions.

 *******************************************************************/


NCPos_c *Threeaxisrough(NCPos_c *finallist ,Precis_t *homepos,Precis_t dd)

{
                Precis_t        x,z,zmin=999.,zmax=-999.9;
      int           f=0,j,m, k,i,n;
      NCPos_c        *finallist1,*pos1,*pos,*pos2;
                finallist1=finallist;
                finallist1=finallist1->next;
                pos1=finallist;
                pos1=pos1->next;
                pos= NC_CreateList();
                pos2=pos;

        do
                {
                        z=pos1->poffset[2];
                        if(z<zmin)zmin=z;
                        if(z>zmax)zmax=z;
                        f++;
                }while(pos1=pos1->next);

                n=itrunc(zmax - zmin)/dd;

        do
```

```
                {
                        finallist1=finallist1->next;
                }while(finallist1->next);

        for(j=1;j<=n;j++)
                {
                        m = j-(j/2)*2;
                if(m){
                                                        pos1=finallist1;
                        }else{pos1=finallist;}

                        i=0;
                        for(i=1;i<f;i++)
                                {
                              if(!m){ pos1=pos1->next;}
           else
            {
                        pos1=pos1->prev;
            }
                        if(pos1->poffset[2] + dd*(n-j) < zmax || x < zmax)
                                        {
               pos2 = NC_AddPosAtBot(pos);
                                        pos2->flag=PON;
                                        pos2->vaxis[0]=0.;
                                        pos2->vaxis[1]=0.;
                                        pos2->vaxis[2]=1.;
             pos2->poffset[0]=pos1->poffset[0];
             pos2->poffset[1]=pos1->poffset[1];
             pos2->poffset[2]=pos1->poffset[2]+dd*(n-j);
             x=pos2->poffset[2];
                                        }
                                }

               }
      pos2 = NC_AddPosAtBot(pos);
      pos2->flag=PON;
      for(j=0;j<3;j++)pos2->poffset[j]=homepos[j];
return pos;
}
```

```
/****************************************************************************
 nc.c:
 ~~~~
        Subroutines for creating, manipulating and outputing a linked list
        containing the information for a tool path.


   Written by: Abbas Vafaeesefat
               Serge Gravelle

       Ecole Polytechnique, Universite de Montreal, December 15 1993
 ****************************************************************************/


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

#include <libkg.h>
#include <libkgl.h>
#include <libkap.h>
#include <mymacros.h>

#include "nc.h"


NCPos_c *NC_SearchForUV(NCPos_c *startpos, Precis_t uu, Precis_t vv)
{
        NCPos_c *pos;

        pos=startpos;

        while(pos)
        {
                if (pos->puv[0]==uu && pos->puv[1]==vv) break;
                pos=pos->next;
        }

        return pos;
}



NCPos_c *NC_SearchForFLAG(NCPos_c *startpos, PosFlag flag)
{
        NCPos_c *pos;

        pos=startpos;

        while(pos)
        {
                if (pos->flag==flag) break;
                pos=pos->next;
        }

        return pos;
}



NCPos_c *NC_SearchForXYZ(NCPos_c *startpos, Precis_t xx, Precis_t yy, Precis_t zz)
{
        NCPos_c *pos;

        pos=startpos;

        while(pos)
        {
                if (pos->pxyz[0]==xx && pos->pxyz[1]==yy && pos->pxyz[2]==zz) break;
                pos=pos->next;
```

```
        }

        return pos;
}



void    NC_ShowList(NCPos_c *list)
{
        NCPos_c *pos;
        int        counter=0;

        pos=list;

        NC_ShowHeader();

        while(pos)
        {
                NC_ShowPos(pos);
                ++counter;
                pos=pos->next;
        }

        printf("\n%d positions in this list\n", counter);
}



void NC_ShowHeader(void)
{
        printf("\n");
        printf("\n      u      v|     px      py      pz|     nx      ny      nz| FLAG *=PON");
        printf("\n ------ ------|------- ------- -------|------- ------- -------|------");
        printf("\n              |     ax      ay      az|    tcx     tcy     tcz|");
        printf("\n =================================================================");
}



void  NC_ShowPos(NCPos_c *pos)
{
        char      onoff[2]={' ','*'};

        printf("\n   %6.3f %6.3f|%7.2f %7.2f %7.2f|%7.3f %7.3f %7.3f| %c",
                     pos->puv[0],    pos->puv[1],
                     pos->pxyz[0],  pos->pxyz[1],  pos->pxyz[2],
                     pos->vnrml[0], pos->vnrml[1], pos->vnrml[2],
                     onoff[pos->flag]  );
        printf("\n                 |%7.3f %7.3f %7.3f|%7.3f %7.3f %7.3f|",
                     pos->vaxis[0],    pos->vaxis[1],    pos->vaxis[2],
                     pos->poffset[0], pos->poffset[1], pos->poffset[2]);
}



NCPos_c *NC_CreateList(void)
{
        NCPos_c *list=NULL;

        return (NC_NewPos(&list,NULL,NULL));    /* returns pointer to new list */
}



NCPos_c *NC_AddPosAtTop(NCPos_c **list)
{
        return (NC_NewPos(&*list,NULL,*list));
}
```

```
NCPos_c *NC_AddPosAtBot(NCPos_c *list)
{
        NCPos_c *pos, *temp=NULL;

        pos=list;
        while(pos->next) pos=pos->next;

        return (NC_NewPos(&temp,pos,NULL));
}



NCPos_c *NC_InsertPos(NCPos_c *list, NCPos_c *pos)
{
        NCPos_c *temp=NULL;
        return (NC_NewPos(&temp,pos,pos->next));
}



NCPos_c *NC_NewPos(NCPos_c **list, NCPos_c *prev, NCPos_c *next)
{
        NCPos_c  *newpos;
        int      i;

        /* create new position in memory */
                if (!(newpos = (NCPos_c *)malloc(sizeof(NCPos_c)) ))
                        {printf("Not enough memory to allocate buffer\n"); exit(1);}

        /* and initialize data */
                        newpos->puv[0]=0.;
                        newpos->puv[1]=0.;

                        for (i=0; i<3; i++) {
                                newpos->pxyz[i]=0.;
                                newpos->vnrml[i]=0.;
                                newpos->vaxis[i]=0.;
                                newpos->poffset[i]=1.;
                        }

                        newpos->flag=PON;


        /* step one: link new with prev and next */
                newpos->next=next;
                newpos->prev=prev;

        /* step two: link prev and next with new */
                if(prev!=NULL) newpos->prev->next=newpos;
                if(next!=NULL) newpos->next->prev=newpos;

        /* change list pointer to newpos unless this is a new list */

                if(*list!=NULL & prev==NULL) *list=newpos;

        return newpos;
}


NCPos_c  *NC_DeletePos(NCPos_c **list, NCPos_c *pos)
{
        NCPos_c *nowpos;

        /* detach links to pos */
```

```
			if (pos==NULL) {
				printf("\nNCPos_note: Nothing to erase!");
				nowpos=NULL;
				}
			else if (pos->prev==NULL && pos->next==NULL) {   /* erase last item */
				printf("\nNCPos_note: List erased!");
				*list=NULL;
				nowpos=NULL;
				}
			else if (pos->prev==NULL) {   /* item at top of list */
				pos->next->prev=NULL;
				*list=pos->next;
				nowpos=pos->next;
				}
			else if (pos->next==NULL) {      /* item at bottom of list */
				pos->prev->next=NULL;
				nowpos=pos->prev;
				}
			else {
				pos->prev->next=pos->next;
				pos->next->prev=pos->prev;
				nowpos=pos->next;
				}

	/* erase info by freeing memory */
		free(pos);

	return nowpos;   /* NOTE: the nowpos acts like the delete key */
}



NCPos_c *NC_MakeListFromUV(Handle_t kgsurf, Precis_t *uvlist, int nb_pts)
{
	NCPos_c  *list, *pos;
	Precis_t        *uu, *vv,
				*pxyz,
				*vnrml,
				normaldir=1.,
				vaxis[3]={0.,0.,1.};
	int             i;

	pxyz  = (Precis_t *) malloc(3*sizeof(Precis_t));
	vnrml = (Precis_t *) malloc(3*sizeof(Precis_t));



	for (i=0; i<nb_pts; i++) {

		if (i)  pos = NC_AddPosAtBot(list);
		else {
				list = NC_CreateList();
				pos  = list;      }

		uu = (uvlist+2*i);
		vv = (uvlist+2*i+1);

		pos->puv[0] = *uu;
		pos->puv[1] = *vv;


			pxyz=KG_InterpolSurfOnUV(kgsurf, uu, 1, 0, 0);

			pos->pxyz[0] = *pxyz;
			pos->pxyz[1] = *(pxyz+1);
			pos->pxyz[2] = *(pxyz+2);

			vnrml=KG_InterpolSurfNormalOnUV(kgsurf, uu, 1);
```

```
/*       if (*(vnrml+2) < 0) {normaldir = -1. ;}
                   else {normaldir = 1.;}
                   inclure si probleme avec direction des normales*/

                   pos->vnrml[0] = normaldir * (*vnrml);
                   pos->vnrml[1] = normaldir * (*(vnrml+1));
                   pos->vnrml[2] = normaldir * (*(vnrml+2));


                   pos->vaxis[0] = vaxis[0];
                   pos->vaxis[1] = vaxis[1];
                   pos->vaxis[2] = vaxis[2];

                   pos->flag = POFF;

        }

        free(pxyz);
        free(vnrml);

        return list;
}



NCPos_c *NC_MakeListFromXYZ(Handle_t kgsurf, Precis_t *xyzlist, int nb_pts)
{
        NCPos_c  *list, *pos;
        Precis_t        puv[2],
                                pxyz[3],
                                *vnrml,
                                normaldir=1.,
                                vaxis[3]={0.,0.,1.};
        int             i;


        vnrml = (Precis_t *) malloc(3*sizeof(Precis_t));



        for (i=0; i<nb_pts; i++) {


                if (i)  pos = NC_AddPosAtBot(list);
                else {
                                list = NC_CreateList();
                                pos  = list;      }

        pxyz[0] = *(xyzlist+i*3);
              pxyz[1] = *(xyzlist+i*3+1);
              pxyz[2] = *(xyzlist+i*3+2);

                        pos->pxyz[0] = pxyz[0];
                        pos->pxyz[1] = pxyz[1];
                        pos->pxyz[2] = pxyz[2];


                        KG_xyz2uv(kgsurf, pxyz, &puv[0], &puv[1]);


                        pos->puv[0] = *puv;
                        pos->puv[1] = *(puv+1);


                        vnrml=KG_InterpolSurfNormalOnUV(kgsurf, puv, 1);

 /*     if (*(vnrml+2) < 0) {normaldir = -1. ;}
```

```
                        else {normaldir = 1.;}
                        inclure si probleme avec direction des normales*/

                        pos->vnrml[0] = normaldir * (*vnrml);
                        pos->vnrml[1] = normaldir * (*(vnrml+1));
                        pos->vnrml[2] = normaldir * (*(vnrml+2));


                        pos->vaxis[0] = vaxis[0];
                        pos->vaxis[1] = vaxis[1];
                        pos->vaxis[2] = vaxis[2];

                        pos->flag = POFF;

        }

        free(puv);
        free(vnrml);

        return list;
}



int     NC_OutputSampSet(char *name, NCPos_c *list)
{
        Handle_t        samp;
        NCPos_c *pos;
        Precis_t        *points=NULL;
        FILE            *fp;
        Medium_t i=0;

        /* count number of items in the list */
        pos=list;
        do { i++; } while(pos=pos->next);

        /* store the data in a temporary variable */
        if (!MALLOC(points, Precis_t, i*3))
                printf("Memory Allocation Error (SampFromList)");

                i=0;
                pos=list;
                do {
                        *(points+(i*3))   = pos->poffset[0];
                        *(points+(i*3)+1) = pos->poffset[1];
                        *(points+(i*3)+2) = pos->poffset[2];
                i++;
                } while(pos=pos->next);


        samp = KG_NewSampSet(i,3,0,points,NULL); /* i should be medium_t,NULL=-1*/

        KG_StatusSampSet(samp);

      if ((fp = fopen( name,"wb")) != NULL) {
           KG_SetOutputFile( fp );
           KG_StoreSampSet( samp );
           fclose ( fp );
      }
      else
           {printf("\nFile error: WriteListAsSampSet\n"); exit (1);}

   KG_DeleteSampSet( samp );

        free(points);
}
```

```
Handle_t                  NC_NewSampSetFromList(NCPos_c *list, int TYPE)
{
        Handle_t          samp;
        NCPos_c *pos;
        Precis_t          *points=NULL;

        int               i=0;

        /* count number of items in the list */
        pos=list;
        do { i++; } while(pos=pos->next);

        /* store the data in a temporary variable */
        if (!MALLOC(points, Precis_t, i*3))
                printf("Memory Allocation Error (SampFromList)");

                i=0;
                pos=list;
                do {
        if (TYPE==1) {
                          *(points+(i*3))   = pos->poffset[0];
                          *(points+(i*3)+1) = pos->poffset[1];
                          *(points+(i*3)+2) = pos->poffset[2];
        }
        else {
                          *(points+(i*3))   = pos->pxyz[0];
                          *(points+(i*3)+1) = pos->pxyz[1];
                          *(points+(i*3)+2) = pos->pxyz[2];
        }
                i++;
                } while(pos=pos->next);

        samp = KG_NewSampSet(i,3,0,points,NULL); /* i should be medium_t,NULL=-1*/

        free(points);

return samp;
}



void            NC_WriteList(char *name, NCPos_c *list)
{
                NCPos_c *pos;
                FILE            *fp;

                if((fp=fopen(name,"wt"))!=NULL)
                {
                        pos=list;
                        do
                        {

                        fprintf(fp,"%ld\n",pos->flag);
                        fprintf(fp,"%lf %lf\n", pos->puv[0],pos->puv[1]);
                        fprintf(fp,"%lf %lf %lf\n",pos->pxyz[0],pos->pxyz[1], pos->pxyz[2]);
                        fprintf(fp,"%lf %lf %lf\n",pos->vnrml[0],pos->vnrml[1],pos->vnrml[2]);
                        fprintf(fp,"%lf %lf %lf\n",pos->vaxis[0],pos->vaxis[1],pos->vaxis[2]);
                        fprintf(fp,"%lf %lf %lf\n",pos->poffset[0],pos->poffset[1],pos->poffset[2]);

                        } while(pos=pos->next);
                }
                else printf("file error--sg");

                fclose(fp);
}
```

```
NCPos_c *NC_ReadList(char *name)
{
                NCPos_c *pos, *list;
                int i=0;
                FILE            *fp;

        if((fp=fopen(name,"rt"))!=NULL)
        {


                while(!feof(fp))
                {

                if (i)
                        pos = NC_AddPosAtBot(list);
                else {
                        list = NC_CreateList();
                        pos  = list; }

                fscanf(fp,"%d"   ,&(pos->flag));
                fscanf(fp,"%lf%lf",&(pos->puv[0]), &(pos->puv[1]));
                fscanf(fp,"%lf%lf%lf",&(pos->pxyz[0]),&(pos->pxyz[1]),&(pos->pxyz[2]));
                fscanf(fp,"%lf%lf%lf",&(pos->vnrml[0]),&(pos->vnrml[1]),&(pos->vnrml[2]));
                fscanf(fp,"%lf%lf%lf",&(pos->vaxis[0]),&(pos->vaxis[1]),&(pos->vaxis[2]));
                fscanf(fp,"%lf%lf%lf",&(pos->poffset[0]),&(pos->poffset[1]),&(pos->poffset[2]));
                i++;
                };
        }
        else printf("file error--sg");
        fclose(fp);

   pos = NC_DeletePos(&list,pos);

        return list;
}



void            NC_Output5axis(char *name, NCPos_c *list)
{

                NCPos_c *pos;
                FILE            *fp;

                if((fp=fopen(name,"wt"))!=NULL)
                {
                        pos=list;
                        do
                        {
                        fprintf(fp,"%+lf %+lf %+lf ",
                                        pos->poffset[0],pos->poffset[1],pos->poffset[2]);
                        fprintf(fp,"%+lf %+lf %+lf\n",
                                        pos->vaxis[0],pos->vaxis[1],pos->vaxis[2]);
                        } while(pos=pos->next);
                }
                else printf("file error--sg");

                fclose(fp);
}


void            NC_Output3axis(char *name, NCPos_c *list)
{

                NCPos_c *pos;
                FILE            *fp;

                if((fp=fopen(name,"wt"))!=NULL)
                {
                        pos=list;
                        do
                        {
                        fprintf(fp,"%+lf %+lf %+lf\n",
                                                pos->poffset[0],pos->poffset[1],pos->poffset[2]);
                        } while(pos=pos->next);
                }
                else printf("file error--sg");

                fclose(fp);
}



void            NC_Set5axisInfo(NCPos_c *list)
{
        NCPos_c *pos;

        pos = list;
   do
        {
                pos->vaxis[0] = pos->vnrml[0];
                pos->vaxis[1] = pos->vnrml[1];
                pos->vaxis[2] = pos->vnrml[2];

        } while(pos=pos->next);
}



void            NC_CalcOffset(NCPos_c *list, Precis_t RR, Precis_t rr, Precis_t thick)
{
        Precis_t a, m[3] ,a1;
        NCPos_c *pos;
   int          i;

        pos = list;
   do
        {
                a  =  Dotprod(pos->vnrml, pos->vaxis);

                m[0] = pos->vnrml[0] - a * pos->vaxis[0];
                m[1] = pos->vnrml[1] - a * pos->vaxis[1];
                m[2] = pos->vnrml[2] - a * pos->vaxis[2];
                a1 = Dotprod(m,m);
                for(i=0;i<3;i++) m[i]=m[i]/a1;


                pos->poffset[0] = pos->pxyz[0] + (pos->vnrml[0])*(rr+thick) + m[0]*(RR-rr);
                pos->poffset[1] = pos->pxyz[1] + (pos->vnrml[1])*(rr+thick) + m[1]*(RR-rr);
                pos->poffset[2] = pos->pxyz[2] + (pos->vnrml[2])*(rr+thick) + m[2]*(RR-rr);

        } while(pos=pos->next);
}


Precis_t    Dotprod( Precis_t *v1,  Precis_t *v2)
{

   return ( v1[0]*v2[0] + v1[1]*v2[1] + v1[2]*v2[2] );

}


Precis_t    *Crossprod( Precis_t *v1, Precis_t *v2)
{
   Precis_t  *v, length;
```

```
        MALLOC(v, Precis_t, 3);

   v[0] =  v1[1]*v2[2] - v1[2]*v2[1];
   v[1] = -v1[0]*v2[2] + v1[2]*v2[0];
   v[2] =  v1[0]*v2[1] - v1[1]*v2[0];

        length = sqrt(Dotprod(v,v));
        v[0] = v[0]/length;
        v[1] = v[1]/length;
        v[2] = v[2]/length;

return v;   /* NOTE: returns vector of unit length */
}



void            NC_AppendListToList( NCPos_c *list1, NCPos_c **list2, int REVERSE)
{
        NCPos_c  *pos, *pos2, *pos3;

        /*  GO TO BOTTOM OF LIST */
        pos = list1;
        while(pos->next)  pos=pos->next ;

        if (REVERSE) {
                pos2 = *list2;
                while(pos2->next)  pos2=pos2->next ;

                while (pos2->prev)
                {
                        pos->next = pos2;
                        pos3 = pos2->prev;
                        pos2->prev = pos;
                        pos = pos->next;
                        pos2 = pos3;
                }
                pos->next = pos2;
            pos2->prev = pos;
            pos2->next = NULL;
                *list2 = NULL;
        }

        else {
                pos->next = *list2;
                pos->next->prev = pos;
                *list2 = NULL;
        }

}



Precis_t  *NC_MakeIsoPath(Precis_t  us, Precis_t uf,  Precis_t vs,  Precis_t vf,
                                                                 Precis_t du,  Precis_t dv,  int
*npoints)
{
                Precis_t *puv, *puvref, u, v;
           int          n;

                if (!du || !dv ) return;

                n = ceil( (Abs(uf-us)/du+1) * (Abs(vf-vs)/dv+1)*2 );
                MALLOC(puv, Precis_t, n);
           puvref = puv;

           for (        v=vs; v<=vf; v+=dv ) {
           for (        u=us; u<=uf; u+=du ) {
                        *puv = u;  ++puv;
                        *puv = v;  ++puv;
                }}

                        puv = NULL;

                *npoints = n/2;

return puvref;
}



void            NC_FindLimits( Handle_t surf, Precis_t *normal, int ndivisions,
                                        Precis_t *ui, Precis_t *vi)
{
        Precis_t        *puv, *pxyz, uv, duv, alpha, beta, point[3];
        Precis_t                uf, vf;
        Precis_t                xmax = -99999.9,  xmin = 99999.9, x;
        Medium_t                i, imin, imax;
        int                     j, npoints;

        npoints = (ndivisions+1)*4;

        MALLOC(puv, Precis_t, npoints*2);

        duv = 1./ndivisions;

        uv=0.;
        for (j=0; j<=ndivisions; j++)
        {
                puv[0+j*8] = uv;
                puv[1+j*8] = 0.;
                puv[2+j*8] = 0.;
                puv[3+j*8] = uv;
     puv[4+j*8] = uv;
                puv[5+j*8] = 1.;
                puv[6+j*8] = 1.;
                puv[7+j*8] = uv;
        uv+=duv;
        }

        pxyz = KG_InterpolSurfOnUV(surf,puv,npoints,0,0);


        alpha = -1*asin(normal[2]);     /* assumes that normal is unit vector */
        beta  = -1*atan2(normal[1],normal[0]);

        for (i=0; i<npoints; i++)
        {
                /* TRANSFORM EACH POINT   R(beta,z)*R(alpha,x)*pxyz
                point[0] = *(pxyz+i*3)*cos(beta) -
                                        *(pxyz+i*3+1)*sin(beta)*cos(alpha) +
                                        *(pxyz+i*3+2)*sin(beta)*sin(alpha);
                point[1] = *(pxyz+i*3)*sin(beta) +
                                        *(pxyz+i*3+1)*cos(beta)*cos(alpha) -
                                        *(pxyz+i*3+2)*cos(beta)*sin(alpha);
                point[2] = *(pxyz+i*3+1)*sin(alpha) +
                                        *(pxyz+i*3+2)*cos(alpha);  */

                x = *(pxyz+i*3)*cos(beta) -
                          *(pxyz+i*3+1)*sin(beta)*cos(alpha) +
                          *(pxyz+i*3+2)*sin(beta)*sin(alpha);
                if (x<xmin) {xmin=x; imin=i;}
                if (x>xmax) {xmax=x; imax=i;}
        }

        *ui = puv[imin*2];
        *vi = puv[imin*2+1];
        uf = puv[imax*2];
        vf = puv[imax*2+1];

        free(puv);
        free(pxyz);

}


void  NC_DefinePlane(Precis_t *normal,Precis_t *point,Precis_t *a,Precis_t *b,
         Precis_t *c, Precis_t *d)
{
   Precis_t   length;

   length = sqrt(Dotprod(normal,normal));

   *a = normal[0]/length;
   *b = normal[1]/length;
   *c = normal[2]/length;

   *d = -(*a) * point[0] - (*b) * point[1] - (*c) * point[2];
}
```

```
/*****************************************************************
 ncsimul5.c
 ~~~~~~~~~~
   Simulate the machining process in 5axis mode.

   Written by: Abbas Vafaeesefat
               Serge Gravelle

   Ecole Polytechnique, Universite de Montreal, December 15 1993
 *****************************************************************/

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

#include <libkg.h>
#include <libkgl.h>
#include <libkap.h>
#include <mymacros.h>

#include "nc.h"

char   *InputName(char *text, char *name);
double  InputFloat(char *text, double dflt);



void main()
{

        Handle_t                display, display2,
                                                view,
                                                surf1, asamp, acurv, curv1, curv2,
                                                tool1, tool2;
        Precis_t                ptool1[9],
                                                ptool2[9],
                                                dx, dy, dz,
                                                testpt1[3]={0.,0.,0.},
                                                testpt2[3]={1.,2.,3.},
                                                R=.09375, r, HL,
                                                alpha, beta,
                                                PI = acos(-1.);
        Precis_t                Zdist=0,
                                                A1, A2, A3,
                                                p2[3],
                                                Si[3]={0., 0., 0.}, Sa[3]={0., 0., 0.};
        Small_t         nb_pu = 20,
                                                nb_pv = 20 ;
        Limits_t                lim ;
        Switch_t                RESHOW,
                                                SHOWOFFSETS = OFF,
                                                SHOWXYZ = OFF;

        GLPen_c      pen1 = { YELLOW_COLOR, SOLID_LINE_STYLE,
      THICK_LINE_WIDTH, CROIX_MARK_STYLE, NORMAL_MARK_SIZE,
      255, ISOLINES_RENDER } ;
        GLPen_c      pen2 = { LIGHTBLUE_COLOR, SOLID_LINE_STYLE,
      NORMAL_LINE_WIDTH, CROIX_MARK_STYLE, NORMAL_MARK_SIZE,
      255, ISOLINES_RENDER } ;
        GLPen_c      pen3 = { GREEN_COLOR, SOLID_LINE_STYLE,
      NORMAL_LINE_WIDTH, CROIX_MARK_STYLE, NORMAL_MARK_SIZE,
      255, ISOLINES_RENDER } ;
        GLPen_c      pen4 = { WHITE_COLOR, DASHED_LINE_STYLE,
      NORMAL_LINE_WIDTH, CROIX_MARK_STYLE, NORMAL_MARK_SIZE,
      255, ISOLINES_RENDER } ;
        GLPen_c      pen5 = { RED_COLOR, DASHED_LINE_STYLE,
      NORMAL_LINE_WIDTH, CROIX_MARK_STYLE, NORMAL_MARK_SIZE,
      255, ISOLINES_RENDER } ;

        Angle inc=900, azim=900, twist=0;
```

```
        float xtrans, ytrans, zoom;
        Coord dist=0.;

        APRefer_c       *ref_surf,
                                             *ref_tool1,
                                             *ref_tool2,
                                             *ref_curve;
        APGeom_c                *obj_surf=NULL,
                                             *obj_tool=NULL,
                                             *objs, *objt, *pprevobj, *prevobj, *obj_curve=NULL;

        static  num_plt=0;
        char            fich_plt[13]= "plot", *ext_plt = ".hpg";

        NCPos_c *path,
                                     *pos;
        int             i,k,t=9;
        float           f, tt, pp,pi=3.141592654;
        char                    *clef,name7[20] = "nc_5axis.dat";
        long                    device = 0;
        short                   data;
        FILE                    *fp,*ff;
        char                    surfname[] = "bouteil.bin",
                                             pathname[] = "nc_3axis.list",
                                             winname[] = "NC_Simul",
                                             output[20]= "output",
                                             *name1,
                                             *name2,
                                             *name3,
                                             *name4;

/* INPUT DATA FROM USER */

        R               = InputFloat("Tool Radius (R)",R);
        r               = InputFloat("Tool radius (r)",R);
        HL      = InputFloat("Tool length (HL)", R*10);
        Zdist = InputFloat("Workpiece origin height", Zdist);
        Sa[2] = Zdist;
        name1 = InputName("Surface filename",surfname);
        name2 = InputName("PathList filename",pathname);
        name3 = InputName("Window Text",winname);
        name4 = InputName("5-axis file",output);


        /* INITALIZE GRAPHICS WINDOW */

        GL_InitGraphics();
   minsize(900,700);
        GL_WinOpen( name3 );
        AP_SetWindowConfig();


        /* READ LIST OF POINTS DEFINING TOOL PATH, KRIG CURVE */

        path = NC_ReadList(name2);

        GL_SetCurveResolution(60);
        asamp = NC_NewSampSetFromList(path, 1);
        KG_SetProfileA( ALONG_NORM, LINEAR_COVAR, LINEAR_DRIFT, 0.0);
        curv1 = KG_KrigCurve(asamp);
        display = GL_NewCurveFromKGCurve(curv1, NULL, 0);
        ref_curve = AP_ObjectReference(CURVE_TYPE, curv1, display);
        obj_curve = AP_NewGeomObject( obj_curve, ref_curve, ON, RED_COLOR);

        asamp = NC_NewSampSetFromList(path, 0);
        KG_SetProfileA( ALONG_NORM, LINEAR_COVAR, LINEAR_DRIFT, 0.0);
        curv2 = KG_KrigCurve(asamp);
        display = GL_NewCurveFromKGCurve(curv2, NULL, 0);
        ref_curve = AP_ObjectReference(CURVE_TYPE, curv2, display);
        obj_curve = AP_NewGeomObject( obj_curve, ref_curve, ON, RED_COLOR);



                          ff = fopen(name4, "wt");
                  if (ff == NULL) {printf("\nFILE NOT FOUND"); exit(1); }


        /* READ SURFACE FROM FILE AND MAKE OBJECT*/

                fp = fopen(name1, "rb");
                        if (fp == NULL) {printf("\nFILE NOT FOUND"); exit(1); }
                KG_SetInputFile(fp);
                surf1 = KG_LoadSurf();
                fclose(fp);


                        GL_SetSurfResolution(70,70);
                        display2 = GL_NewIsoSurfFromKGSurf( surf1,NULL,nb_pu,NULL,nb_pv);
                        ref_surf = AP_ObjectReference(SURFACE_TYPE, surf1, display2);



        /*      KRIG TOOL PARTS (1 & 2) AND MAKE THEM OBJECT */

                        ptool1[0] = 0.; ptool1[1] = 0.; ptool1[2] = HL;
                        ptool1[3] = 0.; ptool1[4] = -R; ptool1[5] = HL;
                        ptool1[6] = 0.; ptool1[7] = -R; ptool1[8] = 0.;

                        KG_SetProfileA( ALONG_NORM,  LINEAR_COVAR, LINEAR_DRIFT, 0.0);
                        asamp = KG_NewSampSet(3, 3, 0, ptool1, NULL);
                        acurv = KG_KrigCurve(asamp);
                        tool1 = KG_KrigSurfFromCurveRevolution( acurv, 'z', 360.0 );

                                display = GL_NewPolySurfFromKGSurf( tool1, 36, 36);
                                ref_tool1 = AP_ObjectReference(SURFACE_TYPE, tool1, display);

                        ptool2[0] = 0.; ptool2[1] = -R  ; ptool2[2] = 0.;
                        ptool2[3] = 0. ; ptool2[4] = -R+r-r/sqrt(2.); ptool2[5] = -r/sqrt(2.);
                        ptool2[6] = 0.; ptool2[7] = -R+r; ptool2[8] = -r;

                        KG_SetProfileA( POSITION_NORM, LINEAR_COVAR, LINEAR_DRIFT, 0.0);
                        asamp = KG_NewSampSet(3, 3, 0, ptool2, NULL);
                        acurv = KG_KrigCurve(asamp);
                        tool2 = KG_KrigSurfFromCurveRevolution( acurv, 'z', -360.0 );

                        /*tool2 = KG_KrigSphere(1.2);*/

                                display = GL_NewPolySurfFromKGSurf( tool2, 36, 36);
                                ref_tool2 = AP_ObjectReference(SURFACE_TYPE, tool2, display);

                        KG_DeleteSampSet( asamp );
                        KG_DeleteCurve( acurv );



/* FIRST, PLACE TRANSFORMED COPIES OF THE REF_OBJS INTO OBJ_LISTS */

                                i = 0;
                                /* go to last position of the path00*/
                                pos = path;  do {pos=pos->next;} while(pos->next);
 NC_Output5axis(name7, pos);
                                do
                                {

/* COMPUTE ANGLES */
                                        if (!pos->vaxis[0] && !pos->vaxis[1]) {tt=0.; pp=0.;}
```

```
                                    else {
                tt = pos->vaxis[0]/pos->vaxis[1];
                tt = atan(tt);
                if(pos->vaxis[1] < 0)tt=tt+pi;
                else{
                if(pos->vaxis[0]< 0 && pos->vaxis[1]>0)tt = 2* pi+tt;}
                pp =sqrt(Sqr(pos->vaxis[0])
                        +Sqr(pos->vaxis[1]))/pos->vaxis[2];
                                         pp = atan(pp);
                                    if(!pos->vaxis[2])pp=pi;
                                    if (!pos->vaxis[0] && pos->vaxis[1]<0) {tt=pi;}
                                    if (!pos->vaxis[1] && pos->vaxis[0]>0) tt=pi/2;
                                   if (!pos->vaxis[1] && pos->vaxis[0]<0) tt=-pi/2;
                                   if (pp>1.4)pp=1.4;}

/* COMPUTE NEW POSITION */
                                   A1 = cos(tt)*Sa[0] - sin(tt)*Sa[1] - Si[0];
                                   A2 = sin(tt)*Sa[0] + cos(tt)*Sa[1] - Si[1];
                                   A3 = Sa[2] - Si[2];


                                   p2[0] = cos(tt)*pos->poffset[0]
                                                - sin(tt)*pos->poffset[1] + A1 ;
                                   p2[1] = cos(pp)*sin(tt)*pos->poffset[0]
                                                + cos(pp)*cos(tt)*pos->poffset[1]
                                                - sin(pp)*pos->poffset[2]
                                                + cos(pp)*A2 - sin(pp)*A3 ;
                                   p2[2] = sin(pp)*sin(tt)*pos->poffset[0]
                                                + sin(pp)*cos(tt)*pos->poffset[1]
                                                + cos(pp)*pos->poffset[2]
                                                + sin(pp)*A2 + cos(pp)*A3 ;


t++;
fprintf(ff," N %3d  G01 X %7.5f Y %7.5f Z %7.5f A %7.3f C %7.3f \n", t, p2[0]+Si[0]-Sa[0],
p2[1]+Si[1]-Sa[1],p2[2]+Si[2]-Sa[2], pp*180/pi,tt*180/pi);



/* ADD TRANSFORMED SURFACE TO LIST */
                              obj_surf = AP_NewGeomObject( obj_surf, ref_surf, ON, GREEN_COLOR);

                                        GL_PushMatrix();
                                        GL_LoadMatrix( obj_surf->modeling );
                                             GL_Translate(-p2[0], -p2[1], 0.);
                                             GL_Rot ( pp*180./PI, 'x');
                                             GL_Rot ( tt*180./PI, 'z');
                                             GL_Translate(Sa[0], Sa[1], Sa[2]);
                                        GL_GetMatrix( obj_surf->modeling );
                                        GL_PopMatrix();


                                   /* ADD TRANSFORMED TOOLS TO LIST */
                                   obj_tool=AP_NewGeomObject(obj_tool,   ref_tool1,   ON,
YELLOW_COLOR);


                                        GL_PushMatrix();
                                        GL_LoadMatrix( obj_tool->modeling );
                                             GL_Translate( 0., 0., p2[2] );
                                        GL_GetMatrix( obj_tool->modeling );
                                        GL_PopMatrix();


                                   obj_tool=AP_NewGeomObject(obj_tool,   ref_tool2,   ON,
YELLOW_COLOR);


                                        GL_PushMatrix();
                                        GL_LoadMatrix( obj_tool->modeling );
```

```
                                                  GL_Translate( 0., 0., p2[2] );
                                            GL_GetMatrix( obj_tool->modeling );
                                            GL_PopMatrix();

                              } while (pos=pos->prev);

                         obj_surf = AP_NewGeomObject( obj_surf, ref_surf, ON, GREEN_COLOR);
                      obj_tool = AP_NewGeomObject(obj_tool, ref_tool1, ON, YELLOW_COLOR);
                         obj_tool = AP_NewGeomObject(obj_tool, ref_tool2, ON, YELLOW_COLOR);


                         /* INITIALIZE A VIEW */

                              view = GL_NewView (-1.0, 1.0, -1.0, 1.0);
                              GL_SetViewPosition (view, 1. ,900,900,0);
                              AP_CalcGeomObjectsLimits( obj_surf, &lim);
                              lim.zmax=lim.zmax+HL+R;
                              GL_SetViewLimits( view, &lim);
/*                            GL_SetViewLookAt( view, (lim.xmin+lim.xmax)/2,(lim.ymin+lim.ymax)/2,

(lim.zmin+lim.zmax)/2 ); */
                              GL_SetViewLookAt( view, 0., 0., 0.);
                              GL_ZoomViewFitLimits(view);




/* SIMULATE BY DISPLAYING ALL OBJECTS */


#ifdef __GL__
   qdevice( RAWKEYBD ) ;
   qdevice( REDRAW ) ;
#endif


while (device != ALTQKEY && device != ESCKEY)
{
        switch (device = AP_GetInput( &data ))
        {
        RESHOW = OFF;
        case REDRAW:
                GL_ReshapeView( view );
                RESHOW = ON;
                break;

        case ALTAKEY:
                GL_SetViewPan(view, 0., 0.);
                GL_ZoomViewFitLimits( view );
                RESHOW = ON;
                break;

        case LEFTKEY:
                azim -= 150;  if (azim<=0) azim=3600;
      GL_SetViewPosition (view, dist, azim, inc, twist);
                RESHOW = ON;
                break;

        case RIGHTKEY:
             azim += 150;  if (azim>=3600) azim=0;
      GL_SetViewPosition (view, dist, azim, inc, twist);
                RESHOW = ON;
                break;

        case UPKEY:
                inc += 150;  if (inc>=3600) inc=0;
      GL_SetViewPosition (view, dist, inc, inc, twist);
                RESHOW = ON;
```

```
            break;

        case DOWNKEY:
            inc -= 150;  if (inc<=0) inc=3600;
      GL_SetViewPosition (view, dist, inc, inc, twist);
            RESHOW = ON;
            break;


        case ALTHKEY:
            num_plt++ ;
            sprintf( fich_plt+4, "%02d\0", num_plt ) ;
            strcat( fich_plt, ext_plt ) ;
            if(!(fp = fopen( fich_plt, "wt" )))
                            { printf( "graph_file error\n"); exit(1);}
            GL_InitPlotterHP( fp ) ;

                    GL_PushMatrix();
                    GL_SetViewTransform( view );
                    AP_PlotGeomObject(obj_surf);
                    AP_PlotGeomObject(obj_tool);
                    AP_PlotGeomObject(obj_tool->next);
                    if (SHOWOFFSETS == ON) AP_PlotGeomObject(obj_curve);
                    if (SHOWXYZ == ON) AP_PlotGeomObject(obj_curve->next);
                    GL_PopMatrix();

            GL_EndPlotterHP( ) ;
            fclose( fp ) ;
            break ;


        case OKEY:
            if (SHOWOFFSETS == ON) {SHOWOFFSETS = OFF;}
            else {SHOWOFFSETS = ON;}
            RESHOW = ON;
            break;

        case PKEY:
            if (SHOWXYZ == ON) {SHOWXYZ = OFF; }
            else {SHOWXYZ = ON;}
            RESHOW = ON;
            break;
}

        if (RESHOW == ON) {
            GL_SetViewProjection( view );
            GL_PushMatrix() ;
            GL_ClearView( view, BLACK_COLOR ) ;
            GL_SetViewTransform( view ) ;

                            GL_SetPenStyle( &pen1 );
                            AP_DrawGeomObject(obj_surf);

                            GL_SetPenStyle( &pen2 );
                            AP_DrawGeomObject(obj_tool);
                            AP_DrawGeomObject(obj_tool->next);


                            GL_SetPenStyle( &pen3 );
                                    GL_DrawViewLimits( view ) ;

                            GL_SetPenStyle( &pen4 );
                            if (SHOWOFFSETS == ON) AP_DrawGeomObject(obj_curve);

                            GL_SetPenStyle( &pen5 );
                            if (SHOWXYZ == ON) AP_DrawGeomObject(obj_curve->next);


            GL_PopMatrix();
```

```
                swapbuffers();
   }
}

#ifdef __GL__
   unqdevice( RAWKEYBD ) ;
   unqdevice( REDRAW ) ;
#endif


swapbuffers();

                        objt=obj_tool;
                        objs=obj_surf;
                        do {
                                GL_SetViewProjection( view );
                                GL_PushMatrix() ;
                                GL_ClearView( view, BLACK_COLOR ) ;
                                GL_SetViewTransform( view ) ;

                                        objs=objs->next;
                                        objt=objt->next->next;

                                        GL_ClearView (view, BLACK_COLOR);

                                GL_SetPenStyle( &pen3 );
                                        GL_DrawViewLimits( view ) ;

                                GL_SetPenStyle( &pen1 );
                                        AP_DrawGeomObject(objs);

                                GL_SetPenStyle( &pen2 );
                                        AP_DrawGeomObject(objt);
                                        AP_DrawGeomObject(objt->next);

                                GL_PopMatrix();
                           swapbuffers();

                        while (device != NKEY && device != ALTHKEY)
                                                        {device = AP_GetInput( &data );
                                                        if (device == SKEY) swapbuffers;}

                        if (device == ALTHKEY) {
                                num_plt++ ;
                                sprintf( fich_plt+4, "%02d\0", num_plt ) ;
                                strcat( fich_plt, ext_plt ) ;
                                if(!(fp = fopen( fich_plt, "wt" )))
                                                { printf( "graph_file error\n"); exit(1);}
                                GL_InitPlotterHP( fp ) ;

                                        GL_PushMatrix();
                                        GL_SetViewTransform( view );
                                        AP_PlotGeomObject(objs);
                                        AP_PlotGeomObject(objt);
                                        AP_PlotGeomObject(objt->next);
                                        GL_PopMatrix();

                                GL_EndPlotterHP( ) ;
                                fclose( fp ) ;
                                break ;
                        }
                        device = NULL;

                        } while (objs->next);

getchar();

                        AP_VisitGeomObjects(obj_surf, AP_DeleteGeomObject);
                        AP_VisitGeomObjects(obj_tool, AP_DeleteGeomObject);
                GL_DeleteView( view );
                GL_EndGraphics();
}


char  *InputName(char *text, char *name)
{
        char    *input;
        MALLOC (input, char, 25);

        printf("\n%s [%s] > ",text,name);
        gets(input);
        if (*input==NULL) return name;
        else return input;
}

double  InputFloat(char *text, double dflt)
{
        char    input[25];

        printf("\n%s [%f] > ",text,dflt);
        gets(input);

        if (*input==NULL) {return dflt;}
        else return atof(input);
}
```

```
/*   Header file for nc.c */

typedef enum
{
        POFF,      /* 0, NULL, FALSE */
        PON        /* 1, TRUE */
} PosFlag;


typedef struct ncpos_c NCPos_c;
struct ncpos_c
{
        Precis_t   puv[2];
        Precis_t   pxyz[3];
        Precis_t   vnrml[3];
        Precis_t   vaxis[3];
   Precis_t   poffset[3];
        NCPos_c   *prev;
        NCPos_c   *next;
        PosFlag    flag;
};


NCPos_c  *NC_NewPos(NCPos_c **list, NCPos_c *prev, NCPos_c *next);
NCPos_c  *NC_CreateList(void);
NCPos_c  *NC_AddPosAtTop(NCPos_c **list);
NCPos_c  *NC_AddPosAtBot(NCPos_c *list);
NCPos_c  *NC_InsertPos(NCPos_c *list, NCPos_c *pos);
NCPos_c  *NC_DeletePos(NCPos_c **list, NCPos_c *pos);

NCPos_c *NC_MakeListFromXYZ(Handle_t kgsurf, Precis_t *xyzlist, int nb_pts);
NCPos_c *NC_MakeListFromUV(Handle_t kgsurf, Precis_t *uvlist, int nb_pts);

void      NC_ShowHeader(void);
void      NC_ShowPos(NCPos_c *pos);
void      NC_ShowList(NCPos_c *list);

NCPos_c *NC_SearchForUV(NCPos_c *startpos, Precis_t uu, Precis_t vv);
NCPos_c *NC_SearchForFLAG(NCPos_c *startpos, PosFlag flag);
NCPos_c *NC_SearchForXYZ(NCPos_c *startpos, Precis_t xx, Precis_t yy, Precis_t zz);


void             NC_Set5axisInfo(NCPos_c *list);
void      NC_CalcOffset(NCPos_c *list, Precis_t R, Precis_t r,Precis_t thick);
void      NC_AppendListToList( NCPos_c *list1, NCPos_c **list2, int ORDER);


void             NC_WriteList(char *name, NCPos_c *list);
NCPos_c *NC_ReadList(char *name);

void      NC_Output3axis(char *name, NCPos_c *list);
void             NC_Output5axis(char *name, NCPos_c *list);
int              NC_OutputSampSet(char *name, NCPos_c *list);

Handle_t         NC_NewSampSetFromList(NCPos_c *list, int TYPE);

Precis_t  Dotprod(Precis_t *v1,  Precis_t *v2);
Precis_t *Crossprod(Precis_t *v1, Precis_t *v2);

Precis_t *NC_MakeIsoPath(Precis_t us, Precis_t uf, Precis_t vs,
                                                 Precis_t vf, Precis_t du, Precis_t
dv, int *npoints);
void             NC_FindLimits(Handle_t surf, Precis_t *normal, int ndivisions,
                                                Precis_t *ui, Precis_t *vi);
void             NC_DefinePlane(Precis_t *normal,Precis_t *point,
                                                 Precis_t *a,Precis_t *b,Precis_t
*c, Precis_t *d);
```