Mathematics Mechanization Research Preprints
KLMM, Chinese Academy of Sciences
Vol. 30, 90-106, May 16, 2011

Space Cutter Radius Compensation based on Shape Reconstruction for Multi-axis Milling

Li Han [1, 2, *] Xiao-Shan Gao [2] Hongbo Li [2]

[1] *College of Computer & Information Technology, Liaoning Normal University, 116029, Dalian*
[2] *KLMM, Institute of Systems Science, Chinese Academy of Sciences, 100190, Beijing*
[*] *Corresponding author: Phn: +86-411-84227305, Fax: +86-411-82153377, E-mail: hl_dlls@dl.cn*

Abstract

The numerical control (NC) program for multi-axis milling depends on the parameters of the tool, in particular the radius of the tool. When there is any tool dimensional change due to various reasons, the user needs to re-generate the NC program, which is a time consuming procedure. In this article, a cutter radius compensation method for multi-axis milling is proposed. It takes a general NC program as the input, recovers the normal vectors of the milling surface from the NC program via surface reconstruction, and uses these vectors as compensation vectors to realize space cutter radius compensation. The algorithm has linear complexity. The compensation algorithm is shown to be very effective in reducing the number of undercut points through simulation with the software VERICUT and with real milling for real world NC programs.

Keywords: Space cutter radius compensation, shape reconstruction, multi-axis milling, NC milling.

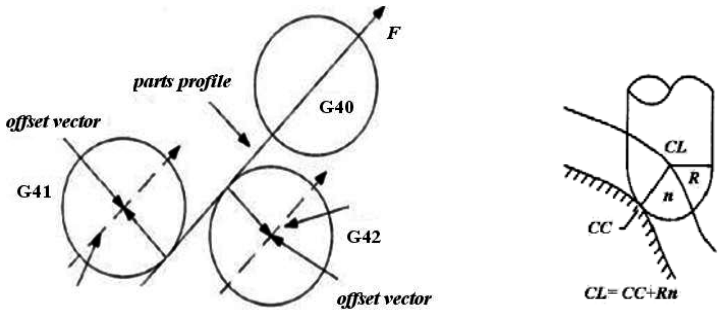
1. Introduction

Multi-axis milling is an important processing method for complex surface manufacturing, which has been widely used in industries of aerospace, automobile, steamboat, die-making, etc. The numerical control (NC) program for multi-axis milling is usually generated by mapping the tool path to the machine-axis movement, which depends on the parameters of the tool, in particular the radius of the tool. As a consequence, when the cutter geometry varies due to tool wear or tool change, we need to re-generate the NC program with the parameters of the new tool, which is a time consuming procedure. In many cases, the user even does not have the original CAD model of NC program, and thus impossible to re-generate the NC program for the new tool. Therefore, to find a cutter radius compensation (CRC) method which is capable of real-time compensating the cutter variation during multi-axis NC milling becomes crucial.

To achieve CRC, we need to compute a new position for the cutter center such that the cutter contact point with the milling surface is still the same for a new cutter with a different radius. The basic principle of CRC in 2-axis or 2.5-axis milling is to program the tool path with the given part profile, and the NC system calculates the cutter center trajectory with the new cutter radius in accordance with a given compensation direction on a 2D plane (left compensation: G41, right compensation: G42) as shown in Fig1-a. Nowadays, CRC function in 2-axis and 2.5-axis CNC milling

used in contouring milling has been well studied (Bao and Tansel 2000; Chen et al.1992; Bohez 2002).

However, the 3D or 5D CRC for multi-axis milling is still a difficult problem. As shown in Fig1-b, the tool path in multi-axis ball-end milling is generated in 3D space. The cutter center location (CL) is obtained by offsetting the cutter contact (CC) point with the amount of cutter radius along its surface normal vector (Bi and Li 2003; Moreton and Durnford 1999). Once the cutter radius *R* changes, we need to compensate the variation of the cutter radius by offsetting the cutter center point along that of surface normal vector. Therefore, to know the surface normal vector of each CC point is the key for CRC in multi-axis ball-end milling.



(a) (b)

Fig1. (a) 2D CRC. (b) The *CL* point is an offset of *CC* point with the amount of cutter radius *R* along that of surface normal vector *n* in 3-axis ball-end milling

Unfortunately, the general NC program provides only the CL data as a set of discrete points, the CC points and the surface normal vectors are not given in NC program. The CRC thus becomes difficult, because we do not know which direction to compensate.

In this paper, we propose an efficient CRC method for multi-axis milling (see Fig.2). Firstly, it takes the CL data in the NC program as input, based on a coordinate transformation matrix, we obtain the cutter trajectory in workpiece coordinate system (WCS). Secondly, the CL surface is reconstructed and the normal vector for each CL point on the CL surface is estimated. Thirdly, we show that the normal vectors of the CC surface at each CC point can be computed from that of the corresponding CL point. Finally, the normal vectors of the CC surface are used as compensation vectors to obtain extended NC program which can be used to achieve real-time space CRC.

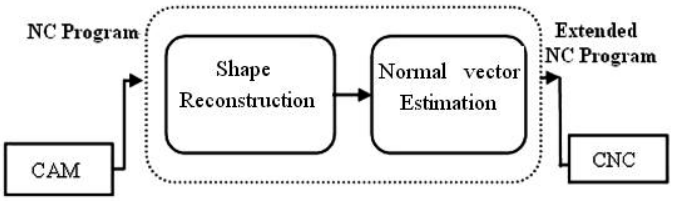


Fig2. Space cutter radius compensation system

In the method, by taking into the advantage of the distribution of CL data along cutter trajectories, the proposed algorithm of shape reconstruction and normal vector computation has a linear complexity in terms of the number of CL points in the NC program. Thus, real-time computation is possible.

We performed extensive experiments with real world NC programs through simulation with the software VERICUT and with real NC machine milling. The compensation algorithm is shown to be very effective in reducing the number of undercut points.

Our method also provides a way to restore the machined surface if the CAD model of the machines surface is not accessible. This has other applications, such as interference detection and manufacturing simulation.

The rest of the paper is structured as follows. We present a shape reconstruction algorithm based on 3-axis and 5-axis NC programs in Section 2. The implementation of space CRC is described in Section 3. Section 4 discusses the experimental results of shape reconstruction and the CRC processing in software Vericut and real NC machine milling. We conclude the paper in Section 5.

2. Shape reconstruction from NC program

The NC program generation for multi-axis milling is usually implemented in two stages. First, the tool paths which indicate the cutter movements in workpiece coordinate system (WCS) are planned in pre-processing, which is independent of the machine structure. Second, the NC program is obtained based on a specific machine structure and CNC system in post-processing, where the CL data in WCS are transformed into corresponding machine-axis movements in the machine coordinate system (MCS) (Bohez 2002; Wang et al. 2002; Jayaram et al. 1997).

In our method, we take the general NC program as input, where the linear interpolation (G01 code) is mainly concerned. Firstly the CL data extracted from G01 codes in the NC program is transformed from MCS to WCS in order to obtain the cutter center trajectories on manufactured surface.

Secondly, we propose a clustering algorithm to construct the adjacent relations between the cutter trajectories, and then triangular patches are constructed based on Delaunay Triangulation method (Alexa et al. 2003; Eck and Hoppe 1996; Levin 2003). Furthermore, the surface normal vectors of the CL surface are effectively estimated by adopting a moving least square (MLS) method (Flynn and Jian 1989; Chen and Schmitt 1992; Kamberov 2004).

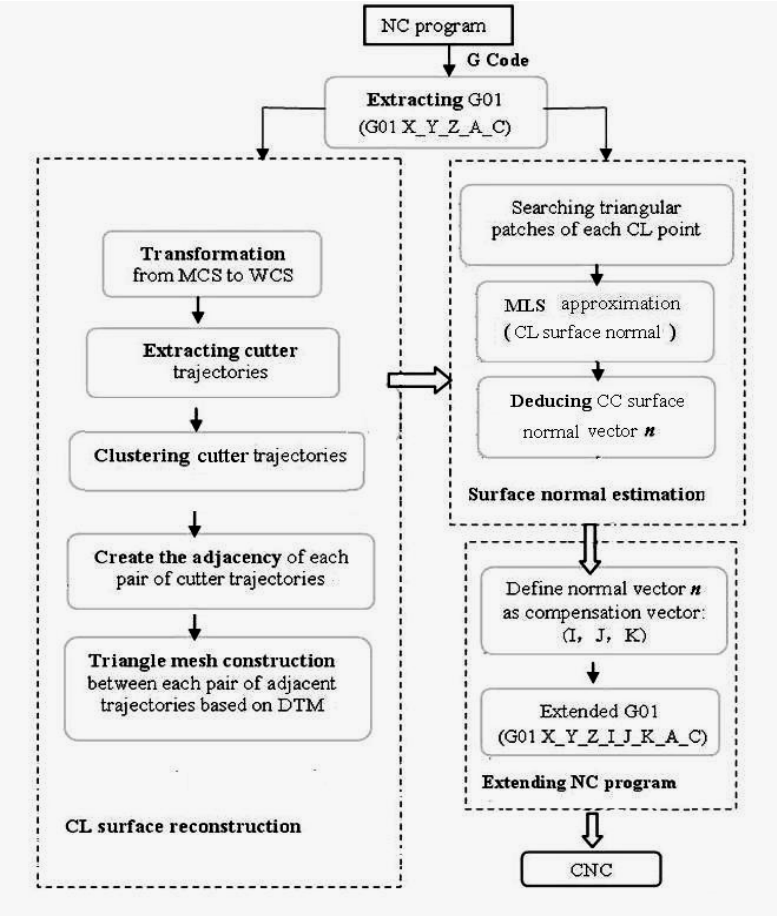


Fig3. The framework of our method

Finally, by analyzing the relationship between the CL surface and the CC surface, we deduce the surface normal vector of CC surface (the manufactured surface) at each CC point. The surface normals at the CC points can be used as the radius compensation vectors to generate extended G01 code which is used to implement real-time space CRC in CNC system. The overall procedure is given in Fig3.

2.1 Transformation matrix from MCS to WCS

Our goal is to recover the surface normal vector of each CL point on CL surface. However, the CL points extracted from G01 codes in NC program are in the MCS. We thus need a transformation matrix to transform the coordinates from MCS to WCS.

The G-code format in 3-axis milling is G01 X_Y_Z_, where X, Y and Z denote the cutter location in MCS, and there is only a translation transformation T_1 between WCS and MCS. However, in 5-axis milling, the G-code is composed of not only cutter location coordinates but also two-axis rotation angles, which are dependent on the machine structure. In our case we take the 5-axis machine with dual-turntable of A and C as an example. The G-code is thus described as: G01 X_Y_Z_A_C_, where C is the workbench rotation angle around the Z-axis and A is the rotation angle around the X-axis. Therefore, the transformation consists of the translation transformation T_1 and two rotation transformations T_2 and T_3:

$$T_1 = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad T_2 = \begin{pmatrix} \cos C & -\sin C & 0 & 0 \\ \sin C & \cos C & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad T_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos A & -\sin A & 0 \\ 0 & \sin A & \cos A & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1}$$

Let (X,Y,Z,A,C) be the CL data in MCS and $(x_{c0},y_{c0},z_{c0},a_x,a_y,a_z)$ the corresponding data in WCS, where a_x,a_y,a_z denotes the orientation of the tool relatively to workpiece. Then the transformation from MCS to WCS for 5-axis milling can be written as follows:

$$\begin{cases} X = x_{c0}\cos C + y_{c0}\sin C \\ Y = -x_{c0}\sin C\cos A + y_{c0}\cos C\cos A + z_{c0}\sin A + d\sin A \\ Z = x_{c0}\sin C\sin A + y_{c0}\cos C\sin A + z_{c0}\cos A + d\cos A \end{cases} \qquad \begin{cases} a_x = \sin A\cdot\sin C \\ a_y = -\cos C\cdot\sin A \\ a_z = \cos A \end{cases} \tag{2}$$

So, if the CL point (X,Y,Z) and the rotation angles A and C are given in the G-code, we can restore the CL point (x_{c0},y_{c0},z_{c0}) in WCS with reverse matrix $T_1^{-1},T_2^{-1},T_3^{-1}$:

$$(x_{c0} \quad y_{c0} \quad z_{c0} \quad 1) = (X \quad Y \quad Z \quad 1)T_1^{-1}T_2^{-1}T_3^{-1} \tag{3}$$

In the case of 3-axis milling, the *CL* data in WCS is achieved by Eq.4.

$$(x_{c0} \quad y_{c0} \quad z_{c0} \quad 1) = (X \quad Y \quad Z \quad 1)T_1^{-1} \tag{4}$$

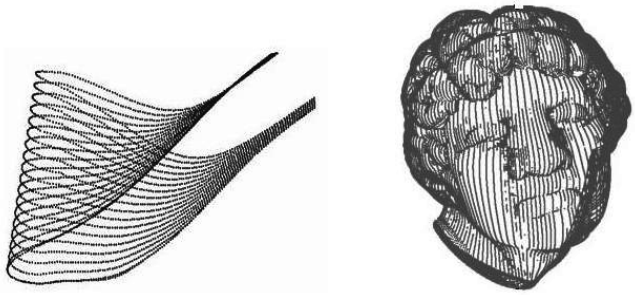


Fig4. The cutter center location (CL) points are obtained from 5-axis NC programs of the impeller and David sculpture

In fact, the CL point in multi-axis NC program is usually the cutter-tip location instead of cutter center location, and then the cutter center location CL_{center} can be calculated by the unit cutter axis vector $A(a_x, a_y, a_z)$ which is obtained from Eq.2:

$$CL_{center} = CL + A \cdot R \tag{5}$$

As shown in Fig4, we extract the CL data from G01 codes in the NC programs of impeller and David sculpture, and by Eq.3 and Eq.5 we can acquire the CL data in WCS. For convinience, the CL data refered in the following description is the cutter center location data.

2.2 Shape reconstruction algorithm

In this section, we will present an efficient algorithm to reconstruct the CL surface from the CL points in the 3-axis and 5-axis NC programs.

Firstly, the cutter trajectories are extracted by analyzing the distribution of the CL points. Secondly, the cutter trajectories are clustered and adjacent relations are created. Finally, triangular patches are constructed by using the Delaunay Triangulation method.

2.2.1 Cutter trajectories extraction

In our method, we consider two kinds of NC programs: 3-axis NC program generated by the iso-planar tool path planning and the general 5-axis NC programs. As we will see below that the 3-axis NC program generated with iso-planar method needs special treatments.

The iso-planar tool path planning is extensively adopted in three-axis milling due to its robustness and simplicity by intersecting a set of parallel vertical planes with the design surface (see Fig5).

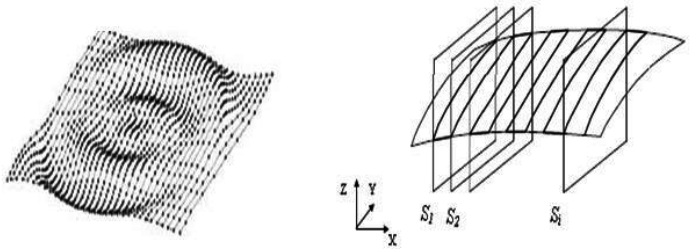
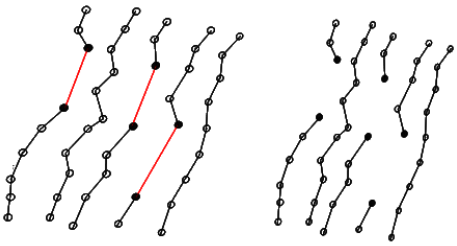


Fig5. The cutter location (CL) points in 3-axis milling

It is known that 3-axis milling is usually used for rapid surface milling, such as the wood cut NC machine as shown in Fig. 6. In such case, there often exist many straight cutter trajectories as shown in Fig6. When the trajectory is a horizontal line, the machine surface is local plane where the Gaussian curvature is zero and the normal vector is perpendicular to the CC plane. Therefore, we need only consider the CL points on non-horizontal lines.

Firstly, the adjacent CL points on each sectional-plane are connected by a set of polylines. We can determine if the cutter is moving on a horizontal line by considering the depth of the adjacent points. If two adjacent CL points have the same depth, and the Euclidean distance between them is larger than *1.5*St,* where *St* is the step length of milling, we then disconnect the horizontal line segments. In this way, the CL points distributed on each sectional-plane are reconstructed as a set of non-horizontal sectional lines. For the example shown in Fig 6, which is the 3-axis NC program for a flower vase milling, we extract 1848 cutter trajectories from 99103 points.



(a) The non-horizontal cutter trajectories extraction

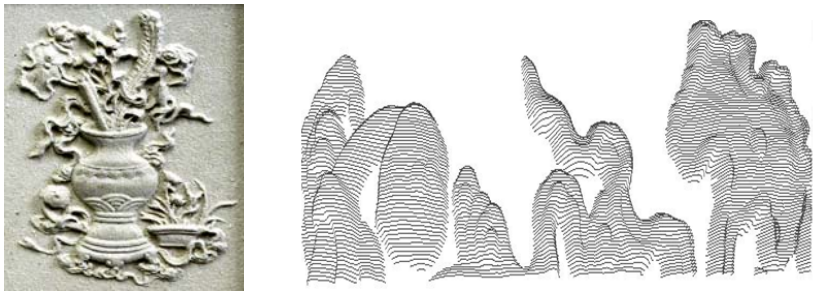
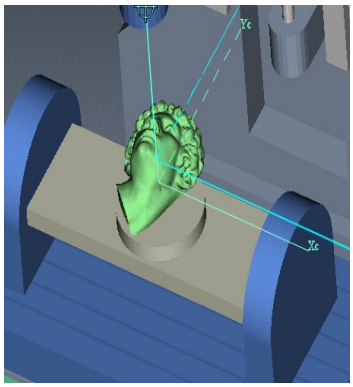
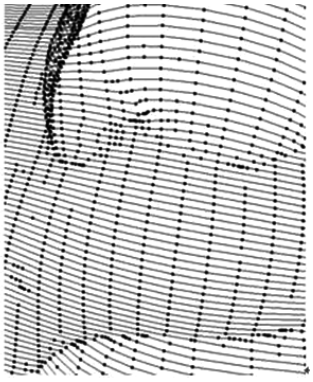


Fig6. (b) Cutter trajectories are extracted from the 3-axis NC program of flower vase milling

The general 5-axis milling is usually adopted for surface finishing, and hence the CL points are evenly distributed. As a consequence, we usually do not need to handle horizontal trajectories. In this case, we can determine the end of a cutter trajectory if there is one rotation angle reaches the maximum. Consider the 5-axis machine with dual-turntable of A and C as an example, where the C rotation-axis is the spindle which provides the unlimited rotation of the cutter around the Z-axis in space (from -360 to 360). The A rotation-axis provides the cutter rotary around the X-axis and it is normally limited from -90 degree to 90 degree (see Fig7-a). In the G-code of the David sculpture (see Fig7-b), the maximum angle of A-axis appears alternately, thus the adjacent cutter trajectories can be extracted by analyzing the given rotation angle of A in the G-code. As shown in Fig7-c, we extract 720 cutter trajectories from 60702 CL points in NC program of David sculpture milling.




```
G01 X.354 Y139.314 Z-132.246 A90. C0.0
X0.0 Y139.46 Z-132.463
X-.354 Y139.314 Z-132.68
X-.5 Y138.96 Z-132.771
X-.46 Z-132.966 C-.269
X-.325 Y139.284 Z-132.356 A89.81 C-.19
X0.0 Y139.418 Z-131.807 A89.731 C0.0
X.325 Y139.284 Z-131.644 A89.81 C.19
X.46 Y138.96 Z-131.958 A90. C.269
X.42 Z-131.763 C.537F2000.
X.389 Y139.119 Z-131.329 A89.794 C.496
X.297 Y139.254 Z-131.061 A89.62 C.38
X.16 Y139.343 Z-130.993 A89.504 C.206
X0.0 Y139.374 Z-131.153 A89.463 C0.0
X-.16 Y139.343 Z-131.521 A89.504 C-.206
X-.297 Y139.254 Z-132.031 A89.62 C-.38
X-.389 Y139.119 Z-132.686 A89.794 C-.496
X-.42 Y138.96 Z-133.16 A90. C-.537
X-.38 Z-133.352 C-.806
```



(a) 5-axis machine configuration (b) G-code of David sculpture (c) cutter trajectories extraction

Fig7. Cutter trajectories extraction from 5-axis NC program of David sculpture milling

To extract the cutter trajectories, we need to traverse the NC program once. As a consequence, the procedure costs *O (N),* where *N* is the number of CL points in the NC program.

We use the following data structure ***Line_Node*** to record a cutter trajectory (see Fig8). The ***Region_Node*** points to adjacent cutter trajectories. ***Line_List*** is used to save the line segments in the algorithm. ***ArcNode*** keeps the triangular patch for a CL point during the triangulation process.

Meanwhile, we use ***Span_L*** to represent the span of each cutter trajectory on the XY-plane in the case of 3-axis milling assuming the sectional planes perpendicular to the X-axis. The span of a line segment parallel to the Y-axis is represented by the Y-coordinates of the start and end points of the segments, which are stored in the *min* and *max* nodes.

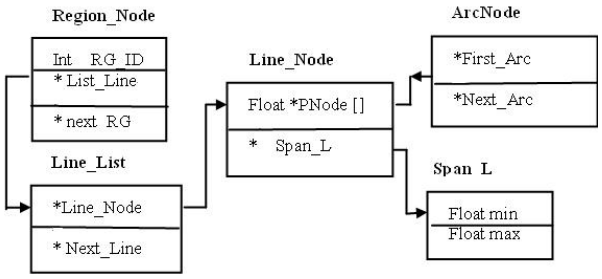


Fig8. The data structure of our reconstruction algorithm

2.2.2 Reconstruction algorithm

An algorithm will be presented to create the adjacent relations among those cutter trajectories. In the case of 5-axis NC programs, the adjacencies among the cutter trajectories can be created as we extract them from the G code in sequence. However, in the case of 3-axis NC programs, there exists a different number of cutter trajectories in each sectional plane. Therefore, we need a clustering algorithm to effectively create appropriate adjacent relations among them.

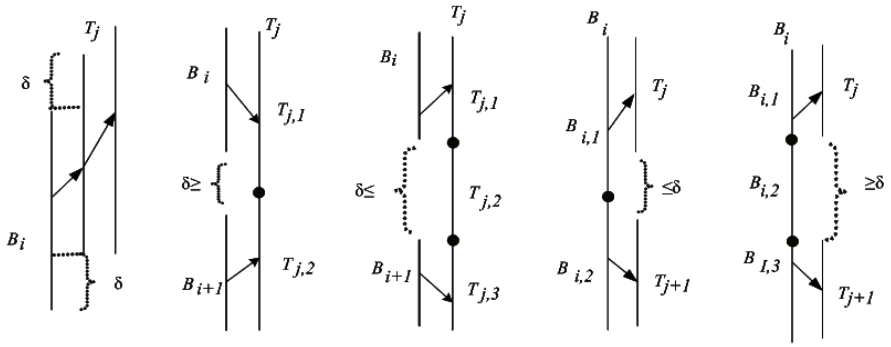
We define the cutter trajectory on a sectional plane Ω_t as the base line (B_i) and the cutter trajectory on an adjacent sectional plane Ω_{t+1} as the target line (T_j) (see Fig9).

For each base line B_i on the first sectional plane $\Omega_{t(t=0)}$, we start a **clustering procedure** to find an appropriate target line T_j on the adjacent sectional plane Ω_{t+1} and then cluster them into the same region RG_k.

In the clustering process, we classify three cases to gradually cluster the target lines into appropriate regions:

- **Simple case:** If the base line B_i shares the same span with the target line T_j, and there is no other line in this span, we then add T_j to the ***Line_List*** of base line B_i as an adjacent line (Fig9-a), and T_j is clustered into the region RG_k containing B_i.
- **Divide T-case:** If there are more than two base lines $B_{i+m}\ \ (m=0,1,...)$ which share the span with the target line T_j, then we divide T_j to create the adjacent relations with B_{i+m} respectively. Based on the distance *D* between the two adjacent base lines, two cases are considered. If the distance *D* is not larger than δ, (the threshold δ is defined by the average of *tool path interval* and *step length*), then we divide the target line T_j at the middle point of the two end points of the two target lines (see Fig9-b); otherwise, we divide the target line T_j responding to $B_i \rightarrow span->min$ and $B_{i+1} \rightarrow span->max$ as shown in Fig9-c. $T_{j,1}$ is clustered as adjacent line of B_i, and a new region node RG_{k+1} is created for $T_{j,2}$
- **Divide B-case:** If the span of the base line BL_i includes several target lines $T_{j+m}\ \ (m=0,1,2,...)$ (Fig9-d,e), then we adaptively divide B_i into sub-lines $B_{i,m}$ and create the adjacency relations similar to the preceding case.

In the clustering algorithm, we stepwisely check base lines and target lines on each two adjacent planes (Ω_t and Ω_{t+1}) to find an appropriate adjacency relationship for the line segments on them. Finally, all the cutter trajectories are clustered into regions with adjacency relationship.



(a) (b) (c) (d) (e)

Fig9. The clustering algorithm

Before giving the algorithm, we introduce two notations. Given sectional lines BL_i *and* SL_j , *If* ($fabs(BL_i \to span \to min - TL_j \to span \to min) \le \delta \,\&\&\, fabs(BL_i \to span \to max - TL_j \to span \to max) \le \delta$), then we say that the two sectional lines share the same span and use the symbol ' $\sim$ ' to denote the fact. If $(BL_i \to span \to min > TL_j \to span \to min) \,\&\&\, (BL_{i+1} \to span \to max > TL_j \to span \to min)$, then we say that the span of sectional line BL_i is included in that of TL_j and use the symbol ' $\subset$ ' to denote the fact. We now give the pseudo code of the algorithm.

Clustering Algorithm

Input: Section planes Ω_t $(t = 0,1,...,T-1)$ containing sectional lines.

Output: Regions that contain sectional lines which are adjacent to each other.

For each sectional plane Ω_t $(t = 0,1,...,T-1)$ ***from left to the right, do the following:***

Create a region RG_0 ***containing the first line segment in*** Ω_0.

Let B_i $(i = 0)$ ***and*** T_j $(j = 0)$ ***be the first line segments on*** Ω_t ***and*** Ω_{t+1} ***respectively.***

Consider the following cases:

1. ***If*** $(B_i \sim T_j)$ ***{*** ***// Do (simple_case)*** *(see Fig9-a)*

   Add line node T_j *to the region* RG_k *containing the base line* B_i.

   Create the adjacency relation for B_i *and* T_j

   $RG_k -> List\_line = B_i;\ B_i -> next\_line = T_j;$

   $B_i = B_{i+1};\quad T_j = T_{j+1}$ ***;*** *Goto step1;*

   }

2. ***Else if*** $(B_i \to span \subset T_j \to span \to max)$ ***{*** ***//Do (Divide T-case)***

   Let B_{i+1} *be the next line segment on* Ω_t.

   If $(B_{i+1} \to span \to \min) - (B_i \to span \to \max) < \delta)$ *(see Fig9-b)*

   {

   Divide T_j *into two lines* $T_{j,1}$ *and* $T_{j,2}$ *at point* $(B_i \to span \to \min - B_{i+1} \to span \to \max)/2$.

   Create the adjacency relations for B_i *and* $T_{j,1}$;

   $B_i -> next\_line = T_{j,1};\quad (BL_i \in RG_k;)$

   $B_i = B_{i+1};\quad T_j = T_{j,2};\quad$ *Goto step1;*

   }

   Else *(see Fig9-c)*

{

Divide T_j *into three line segments* $T_{j,1}$, $T_{j,2}$ *and* $T_{j,3}$ *at points* $B_i \rightarrow span \rightarrow min$; $B_{i+1} \rightarrow span \rightarrow max$;

Create the adjacency relations for B_i *and* $T_{j,1}$;

$B_i -> next\_line = T_{j,1}$; $(BL_i \in RG_k;)$

Create a new region whose starting line segment is $T_{j,2}$

$RG_{k+1} -> List\_line = T_{j,2}$;

$B_i = B_{i+1}$; $T_j = T_{j,3}$; *Goto step1;*

}

Endif

Endif

3. ***Else If*** $(T_j \rightarrow span \subset B_i \rightarrow span)$ ***{ //Do (Divide B-case)***

Let T_{j+1} *be the next line segment in* Ω_{t+1}.

If $(T_j \rightarrow span \rightarrow \min - T_{j+1} \rightarrow span \rightarrow \max) < \delta)$ *(see Fig9-d)*

{

Divide B_i *into two lines* $B_{i,1}$ *and* $B_{i,2}$ *at point* $(T_j \rightarrow span \rightarrow \min - T_{j+1} \rightarrow span \rightarrow \max)/2$;

Create the adjacency relations for $B_{i,1}$ *and* T_j ;

$B_{i,1} -> next\_line = T_j$; $(BL_{i,1} \in RG_k;)$

$B_i = B_{i,2}$; $T_j = T_{j+1}$; *Goto step1;*

}

Else *(see Fig9-e)*

{

Divide B_i *into three line segments* $B_{i,1}$, $B_{i,2}$, *and* $B_{i,3}$ *at points* $T_j \rightarrow span \rightarrow \min$, $T_{j+1} \rightarrow span \rightarrow \max$;

Create the adjacency relations for $B_{i,1}$ *and* T_j ;

$B_{i,1} -> next\_line = T_j$; $(BL_{i,1} \in RG_k;)$

$B_i = B_{i,3}$; $T_j = T_{j+1}$; *Goto step1;*

}

Endif

}

Endif

Endif

4. ***If (i>=L || j>=L) t=t+1; Goto step1;*** *// if the lines on current planes* Ω_t *and* Ω_{t+1} *are all checked, we start checking line segments from next two adjacent planes. L is the number of line segments on plane.*
5. ***If (t>=T) Goto step 6;*** *// T is the number of sectional planes.*
6. ***End***

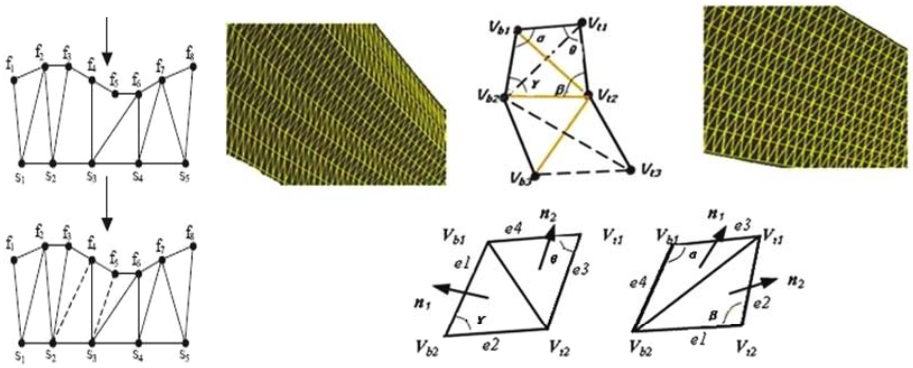
Since we only consider sectional lines in sequence on each two adjacent vertical planes, the procedure of the clustering algorithm costs *O (m)* where *m* is the number of extracted sectional lines.

2.2.3 Triangular mesh and normal vector computation

Liang and Lin (2002) presented a method to add line segments between two trajectories by computing a formula based on the number of points on each trajectory. However, this method is not able to avoid the long thin triangles (see Fig10-a).

Quality mesh generations should guarantee nice measurement of the shape. The Delaunay triangulation achieves fair triangles based on minimum maximum dihedral angle criterion and the rule

of minimum angle between normal vectors. We thus adopt the Delaunay triangulation to create line connections between each pair of adjacent cutter trajectories.



(a) Liang and Lin (2002) (b) Delaunay Triangulation

Fig10. Results based on Liang and Lin's method and Delaunay Triangulation method (DTM)

As shown in Fig 10-b, for two adjacent cutter trajectories, four vertices $\Phi(Vb_1, Vb_2, Vt_1, Vt_2)$ are used as two beginning vertices on each trajectory. There are two kinds of triangulation by adding diagonal line $\overrightarrow{Vb_1Vt_2}$ or $\overrightarrow{Vt_1Vb_2}$. The angles ($\alpha$, β, γ and θ) for two triangulations are computed respectively. If the angle $\alpha+\beta$ is larger than $\gamma+\theta$, the diagonal line $\overrightarrow{Vb_1Vt_2}$ is added; otherwise, a diagonal line $\overrightarrow{Vt_1Vb_2}$ is inserted.

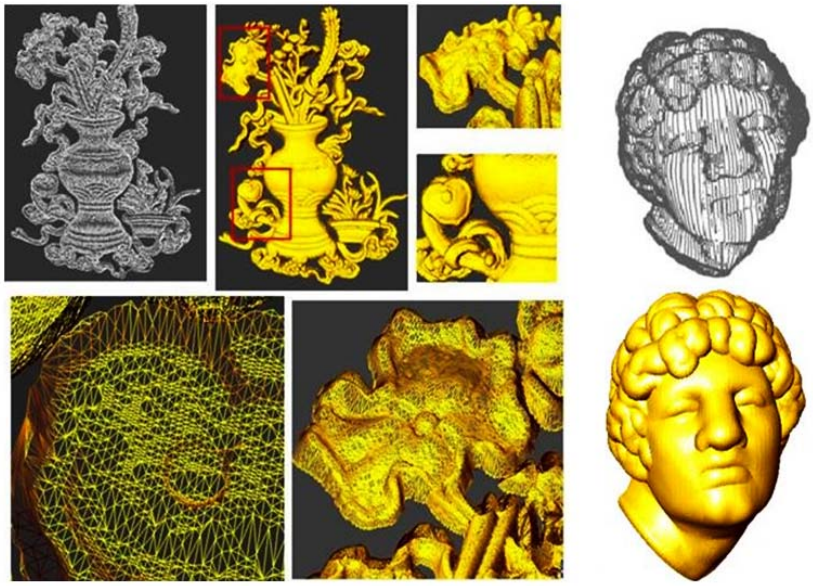


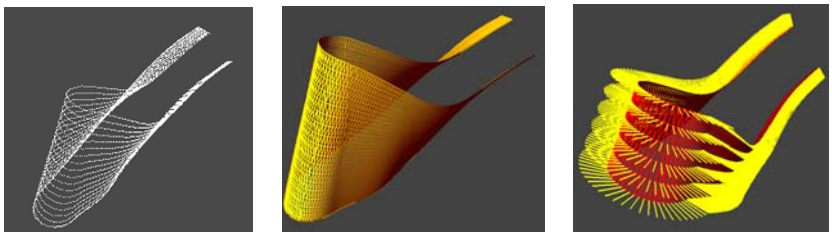
Fig11. Triangle mesh construction from *CL*-Data of flower vase and David sculpture

In order to optimize the triangle mesh, we further adopt the rule of minimum angle between normal vectors. Firstly, we calculate the normal vectors ($n_1 = \frac{e_1 \times e_2}{\|e_1 \times e_2\|}, n_2 = \frac{e_3 \times e_4}{\|e_3 \times e_4\|}$) of the two triangles in different triangulations (see Fig10-b), and the dihedral angles between the two triangles are obtained by dot product (i.e. $\theta = \pi - \arccos(n_1 \cdot n_2)$). In order to obtain a flat plane, we will choose the triangulation where the dihedral angle between the two triangles are larger. A combination (sum of the two kinds of angles) of the two rules is used to find the best result.

Finally, $\Phi(Vb_1,Vb_2,Vt_2,Vt_3)$ or $\Phi(Vb_2,Vb_3,Vt_1,Vt_2)$ is constructed. A set of triangles are finally tiled between each two adjacent cutter trajectories (see Fig11).

To construct each triangle, we need to compute six angles. During the triangulation process, we take advantage of the distribution of CL points along cutter trajectories. As a consequence, the procedure costs *O (N)*, where *N* is the number of the CL points in NC program.

Based on the achieved neighborhood of each CL point during the triangulation process, we adopt the theory of MLS (Flynn and Jian 1989; Chen and Schmitt 1992; Kamberov 2004) to obtain the surface normal vectors (see Fig12). The procedure can be described as follows**:**



(a) CL points of impeller (b) reconstructed triangular mesh (c) achieved surface norm vectors (yellow lines)

Fig12. The surface norm vector estimation of triangular mesh

Input: Triangular mesh of CL points $(P_i\}_{i=1}^{S}$.

Output: Surface normal vectors $(N_i\}_{i=1}^{S}$ at P_i.

(1) Generate the neighborhood of the reference CL point $\{(P_i\}_{i=1}^{S}\}$ based on the triangular mesh by searching $P_i->firstarc$ and $P_i->nextArc$. For a point *P*, we use the points *Q* such that *PQ* is the edge of a triangle mesh as its neighboring points. It is clear a point has at most eight neighboring points.

(2) Using the least square method to compute the surface normal for the reference point based on its neighboring points.

(3) Output the surface normal vector of each CL point.

Since we already established the adjacency information for the CL points in the triangulation procedure, to find the neighboring points for a given CL point costs a constant time. Therefore, the geometric parameter estimation in the MLS method avoids the time-consuming procedure of *KNN*（K nearest neighborhood）searching (Chen and Schmitt 1992; Kamberov 2004), and the total computation is simply the solving of the least square problem for each CL point. Also the number of points needed in the least square computation is at most eight, as mentioned in the algorithm. As a consequence, the procedure costs *O (N)*, where *N* is the number of the CL points in NC program.

3. Space cutter radius compensation

We propose an efficient approach to the space CRC by extending the G01 code with compensation vectors. It consists of three steps: firstly, the normal vector of each CC point is deduced from that of the corresponding CL point; secondly, the normal vector of each CC point is used as the compensation vector to generate extended G01 code; thirdly, the compensation function is implemented in the CNC system to achieve space CRC.

3.1 The deduction of surface normal vector of CC point

Since ball-end cutter and flat-end cutter are the special circumstances of filleted-end cutter, we

then take the filleted-end cutter as a generic shape to analyze the relationship between the surface normal vector of CC point and that of CL point.

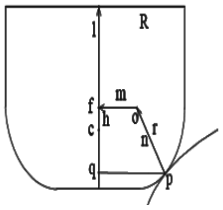


Fig13. The filleted-end cutter in 3-axis milling

The example in Fig13 shows the filleted-end cutter in 5-axis milling. Its end is composed of a flat bottom and the surface of revolution generated by a quarter-circle. The inner radius and outer radius are r and R respectively. The CL point is denoted by vector **c**, and is on the cutter axis. The CC point is denoted by vector **p**. The direction of the cutter axis is denoted by unit vector **l**. The normal direction of the surface under milling at point **p** is denoted by unit vector **n.** The foot drawn from **p** to the cutter axis is denoted by vector **q**. **o** is the point obtained from **p** by translating along direction **n** the distance r, and **f** is the foot drawn from **o** to the cutter axis. The signed distance from **c** to **f** along the cutter axis is a constant, and is denoted by *h*.

The direction from **o** to **f** corresponds to unit vector

$$m = \frac{n-(n\cdot l)l}{\left|n-(n\cdot l)l\right|} = \frac{n-(n\cdot l)l}{\sqrt{1-(n\cdot l)^2}} \tag{6}$$

Then

$$c = f - hl = p + rn + (R-r)m - hl \tag{7}$$

Now let (**c,l**) vary by assuming that **c** = *c(u, v)* is on a given surface parameterized by *u, v*. The total differentiation of a vector-valued function **x** is

$$dx = \frac{\partial x}{\partial u}du + \frac{\partial x}{\partial v}dv \tag{8}$$

From $n^2 = l^2 = 0$, we have

$$n\cdot dn = l\cdot dl = 0 \tag{9}$$

By the definition of normal vectors, we have

$$n\cdot dp = 0 \tag{10}$$

By differentiating, we get

$$dm = \frac{dn-(dn\cdot l)l-(n\cdot dl)l-(n\cdot l)dl}{\sqrt{1-(n\cdot l)^2}} + \frac{(n-(n\cdot l)l)(n\cdot l)(dn\cdot l+n\cdot dl)}{(\sqrt{1-(n\cdot l)^2})^3} \tag{11}$$

Making inner product with **n**, we get

$$n\cdot dm = -\frac{(n\cdot dl)(n\cdot l)}{\sqrt{1-(n\cdot l)^2}} \tag{12}$$

By differentiating Eq.7, we have

$$dc = dp + rdn + (R-r)dm - hdl \tag{13}$$

Making inner product with n, we get

$$n\cdot dc = (R-r)n\cdot dm - hn\cdot dl = -(R-r)\frac{(n\cdot dl)(n\cdot l)}{\sqrt{1-(n\cdot l)^2}} - h(n\cdot dl) \tag{14}$$

If $R = r$ and $h = 0$, *i.e.*, the cutter has ballnose end, then $n\cdot dc = 0$, and **n** is the normal vector of surface **c** = c(*u,v*), that is, the surface normal vector of CC point consistent with that of CL point. Alternatively, if $dl = 0$, that is, cutter axis is a constant, and it implements 3-axis milling, we reach the

same conclusion. We thus proved the following theorem:

Theorem 1. In the cases of 3-axis NC milling or ballnose end in 5-axis milling, the unit normal vector of the CL surface is the same as that of CC surface at the corresponding point.

In the general case, by squaring Eq.14, we get

$$(n\cdot(dc+hdl))^2(1-(n.l)^2)=(R-r)^2(n\cdot dl)^2(n\cdot l)^2 \tag{15}$$

It is composed of two equations of degree 4 in **n**. They together with $n^2=1$ suffice to solve **n**. Therefore, the normal vector of the CC surface can always be computed from that of the CL surface.

3.2 The implementation of space CRC

The space CRC function in multi-axis ball-end CNC milling system is to compute the offset trajectory of the cutter center in accordance with the variation of the cutter radius. Its essence is to offset the *CC* point according to its surface normal vector (compensation vector) with the amount of new cutter radius.

In Eq.7, **c** and **p** are a pair of corresponding CL and CC points, and **n** is the normal of the CC surface at point **p**. Note that the normal vector **n** has been computed in Section 3.1. All other parameters in Eq.7 are known. We can realize space CRC function as follows. Once the radius of cutter varies from R and r to R_n and r_n respectively, the new CL point c_n is generated by Eq.16.

$$c_n=c+\Delta rn+(\Delta R-\Delta r)m \tag{16}$$

where $\Delta r=r_n-r,\Delta R=R_n-R$.

Suppose that the original G01 code is G01 X_Y_Z_A_C_where X, Y and Z are the coordinates of the CC point A, C are the two rotation angles. We change the G code to an extended one like G01 X_Y_Z_A_C_I_J_K_, which also contains the normal vector (I, J, K) of the corresponding CC point as shown in Fig 14.

```
N0755 G01 X0.000  Y54.820  Z97.351  A32.005  C90.000  I-0.000  J0.996  K0.090  F 250.000
N0760 X1.426  Y54.419  Z97.566  A31.777  C93.353  J0.996  K0.095  F 2.575
N0765 X2.866  Y53.126  Z98.248  A31.044  C97.009  J0.994  K0.109  F 2.557
N0770 X3.564  Y52.103  Z98.773  A30.471  C98.998  J0.993  K0.121  F 2.544
N0775 X4.231  Y50.806  Z99.420  A29.753  C101.128 J0.991  K0.136  F 2.527
N0780 X4.853  Y49.225  Z100.184 A28.888  C103.429 J0.988  K0.155  F 2.508
N0785 X5.414  Y47.348  Z101.053 A27.878  C105.940 J0.984  K0.177  F 2.486
```

Fig14. The extended NC program with compensation vectors *(I, J, K)*

The space CRC function can be implemented by instruction G45 which is unspecified and belongs to the same function group with G41, G42 in CNC systems. G45 can be used with G00, G01, G90, G91, F, S, and T together, and the command format is:

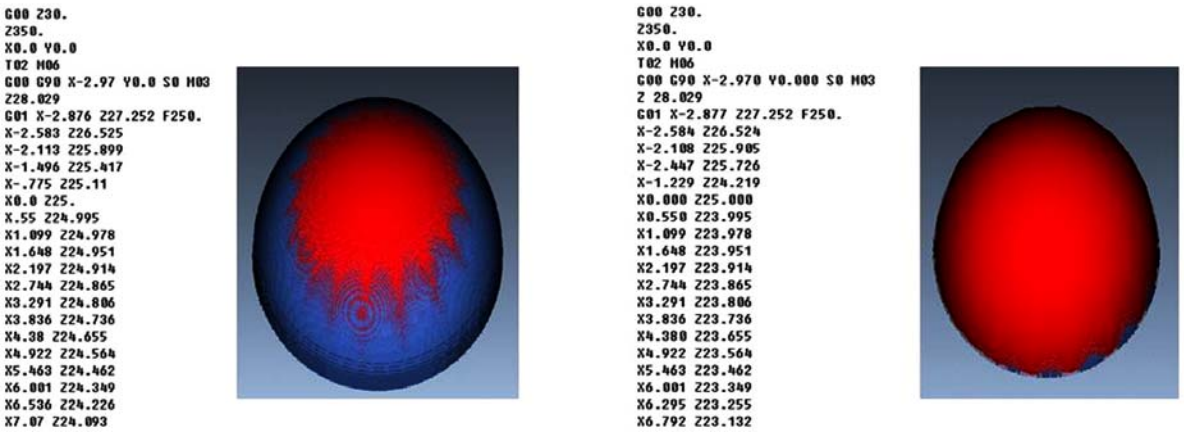
G01 G45 X… Y… Z…A...C... I…J … K… F … T…. (17)

Assuming the original tool T01 has radius R and r. After a period of processing or a new tool T02 with different radius. The CNC system can automatically modify the CL coordinate by calling the compensation function to change tool as T02, without changing the existing proc edures. For instance, a tool compensation code is as follows: “G01 G45 X12.5 Y71.69 Z-3.83 A32.678 C101.654 l0.987 J-0.074 K-0.139 F100. T02”, where the new tool T02 is called to co mpute CL coordinate based on Eq.16.

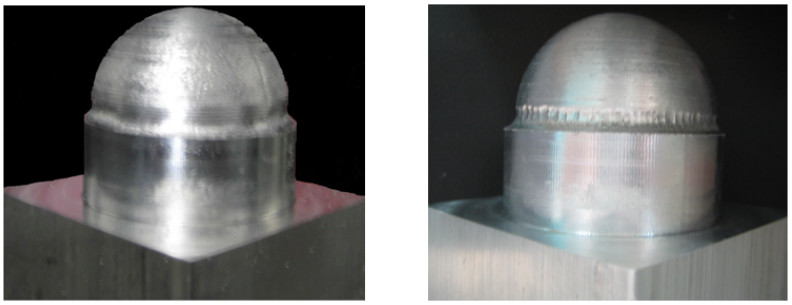
We now give the overall procedure of space CRC. Firstly, we extract the CL data from the original NC program, transform the coordinate from the MCS and the WCS, and reconstruct the CL surface.

Secondly, we compute the normal vector of each CC point and generate the extended G code. When the cutter radius varies, we can compute the new CL point by Eq.16. The overall complexity of the algorithm is *O(N)*, where *N* is the number of CL points in the NC program.

We tested the space CRC function in the simulation software **Vericut**. The NC program for 3-axis hemisphere milling is generated by using a standard 3mm ball-end cutter; and then we simulated the milling procedure by setting a 2.8 mm ball-end cutter. As shown in Fig15-a, there exist many undercut points (blue region) as expected. We rerun the program with the extended G code with the space CRC function. The simulation with the new NC program is shown in Fig15-b, where much less undercut points exist. In Fig15-c, we can see the results from real milling; the tool path trajectory on hemisphere surface is rather uneven before CRC, and the machining quality is much improved after CRC. More experimental results will be given in Section 4.



(a) The simulation of original NC program (b) the simulation of updated NC program



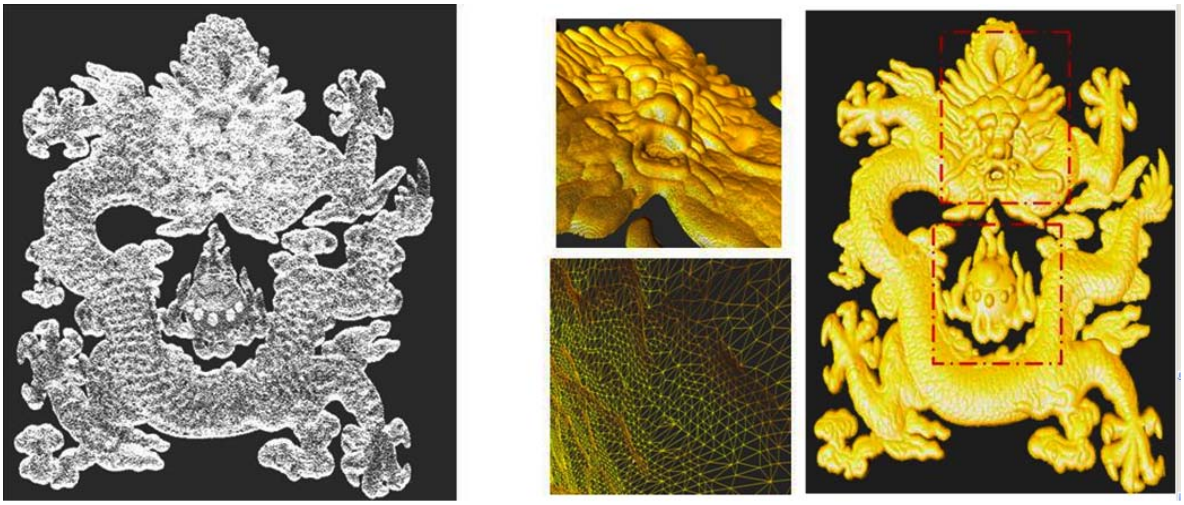
(c) Before CRC and after CRC in real milling

Fig15. The simulation of Space CRC for 3-axis hemisphere milling in Vericut and real milling

4. Experimental results

We have implemented our algorithms in C++ with Open inventor as the Graphical User Interface on a desktop PC with a 3 GHz P4-CPU and 2 GB RAM. Our program is tested with four practical NC programs: the Dragon (Fig. 16, Fig. 17), the Flower Vase (Fig. 6, Fig. 11), the Impeller (Fig. 12), and the David sculpture (Fig.4, Fig. 18).

Fig.16 presents the experimental result of surface reconstruction from the 3-axis NC program of a Dragon ball-end milling. 4206 cutter trajectories are extracted from 280137 CL points in the NC program and 551112 triangle patches are created based on our reconstruction method. In **Table 1,** we give the running times for our NC programs. These running times are compatible with the complexity analysis of our program.



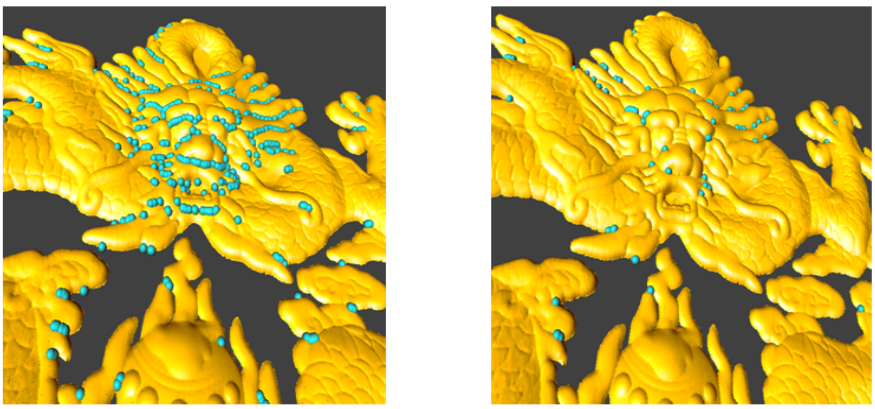
(a) CL points are extracted from NC program of dragon (b) reconstructed triangle mesh model

Fig16. The dragon model is reconstructed from NC program

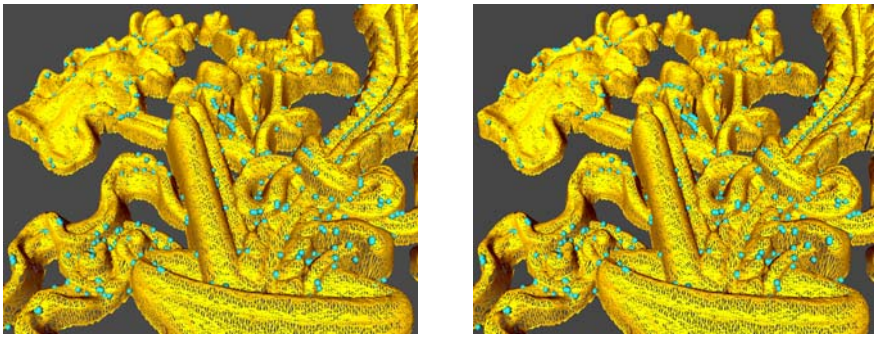
Table 1: The performance of our method.

Models (CL points: n)	Extracted cutter trajectories (m)	Numbers of triangles	Surface Reconstruction(s)	Differential properties estimation (s)	the rate of reducing undercut points
Dragon (280137)	4206	551112	60.59s	41.61s	97.31%
Flower Vase (99103)	1868	193559	5.43s	22.32s	92.78%
Impeller (4935)	20	9652	0.46s	1.26s	98.63%
David (60702)	720	118369	1.19s	4.27s	95.41%

In Fig17-d, an NC program of 3-axis dragon milling is originally generated with an F-ball of 3mm; when the cutter radius is set to 2.8mm, it generates 4836 undercut points (blue balls).



(a) There are 4836 undercut points before CRC (b) There are 130 undercut points after CRC

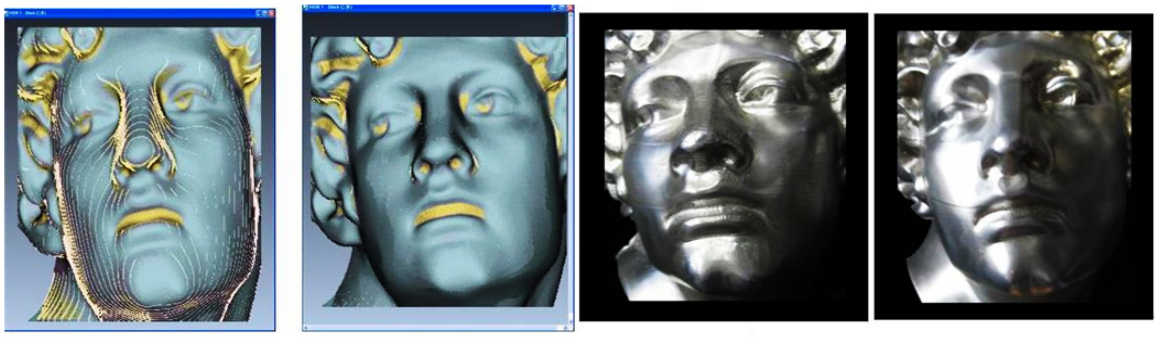


（c） the comparison of the undercut points in 3-axis flower vase milling

Fig17. The simulation of 3-axis dragon milling with F-ball cutter.

Based on our CRC method, the undercut points are significantly decreased to 130 as shown in Fig17-b. Fig17-c has shown the undercut detection results before CRC and after the CRC in 3-axis flower vase milling.

In Fig18, the CRC in 5-axis ball-end milling has been tested in **Vericut** and real milling. The standard NC program for the David sculpture milling is originally generated with a 3mm ball-end cutter, and then we use a 2.8mm ball-end cutter during the simulation. As shown in Fig18-a (left), there exist 11560 undercut points, which mostly appear in the nose and cheek parts. When we update the NC program with the obtained compensation vectors and new radius, there exist only 558 undercut points remained as shown in Fig19-a (right). It effectively reduced 95% of the undercut points. In Fig19-b, we can see the results from real milling. The lip and face parts are rough before CRC and much smoother after CRC.



(a) Before CRC and after CRC in Vericut (b) Before CRC and after CRC in real milling

Fig18. The experimental results of 5-axis David sculpture milling in **Vericut** and real milling

5. Conclusion

Since the cutter radius varies often due to tool wear or tool change, efficient space CRC method is required to meet the increasing demand in practice on better product quality and efficient milling. The CRC function in 2-axis and 2.5-axis CNC milling used in contouring milling has been well implemented by offsetting the cutter center trajectory in accordance with a given direction on a 2D plane. However, multi-axis CRC function is still a difficult issue because the conventional NC program is not capable of providing the space compensation vectors.

In this paper, we present an efficient space CRC method which takes general NC programs as input and recovers the normal vector of each CL point by reconstructing the surface of the CL points efficiently. Furthermore, the normal vectors of CC points can be deduced from that of the CL points,

and are used as compensation vectors. The original G01 codes are transformed into extended G01 codes which allow real-time CRC in the CNC system. We have implemented our CRC method and simulated the milling for several real NC programs in the Vericut software. The experimental results have shown that our algorithm is robustness and effective. The space CRC method avoids the NC program regeneration and tool replacement when there are tool wear and tool dimensional change, and thus effectively improves the quality and efficiency for NC milling.

Acknowledgements: The research presented in this paper is supported by a National Key Basic Research Project, a grant from NSFC, and a project of Liaoning BaiQianWan Talents program. We also want to thank Shenyang Institution of Computing for providing the NC programs.

References

1 Alexa M., Behr J., Cohen-Or D. and Fleishman S. (2003) Computing and rendering point set surfaces. *IEEE Transactions on Computer Graphics and Visualization*, 1, 3–15.
2 Bao W. Y., Tansel I. N. (2000) Modeling micro-end-milling operations. Part11: tool run-out, *International Journal of Machine Tools and Manufacture*. 2175-2192.
3 Chen J. S., Yuan J., Ni J., Wu S. M. (1992) Thermal error modeling for volumetric error compensation. *Sensors and Signal Processing for Manufacturing. ASME, PED*, 55, 113-125.
4 Bohez E. L. J. (2002) Compensating for systematic errors in 5-axis NC milling. *Computer-Aided Desig*n. 34, 391-403.
5 Bi J. X. and Li Q. (2003) The space radius compensation research of tools on five–axis linkage. *Machine Tool & Hydraulics.* 105-106.
6 Moreton D. and Durnford R. (1999) Three-dimensional tool compensation for a three-axis turning centre. *International Journal of Advanced Manufacture Technology*, 15, 649-654.
7 Wang S. M., Liu Y. L. , Kang Y. (2002) An efficient error compensation system for CNC multi-axis machines. *International Journal of Machine Tools & Manufacture*, 42, 1235-1245.
8 Jayaram S., KaPoor S. G., DeVor R. E. (1997) A model based approach for detection of process faults in the face milling process. *Transactions of NAMRC/SME*, 6, 117-122.
9 Eck M. and Hoppe H. (1996) Automatic reconstruction of B-spline surfaces of arbitrary topological type. *Computer Graphics, Proceedings* SIGGRAPH ’96, 325 - 334.
10 Levin D. (2003) Mesh-independent surface interpolation. *Geometric modeling for scientific visualization*, 37–49.
11 Liang S. R. and Lin A. C. (2002) Probe-radius compensation for 3D data points in reverse engineering, *Computers in. Industry*, 48, 241-251.
12 Flynn P. andJian A. (1989) On reliable curvature estimation. *Proc. of the IEEE conference on computer vision and pattern recognition*. 110–6.
13 Chen X. and Schmitt F. (1992) Intrinsic surface properties from surface triangulation. *Proc. of the European conference on computer vision*. 739–43.
14 Kamberov G. (2004) Topology and geometry of unorganized point clouds. *In: 2nd International symposium on 3d data processing, visualization, and transmission*. 743–50.