Pencil Curve Tracing via Virtual Digitizing

Jung W Park
School of Mechanical Engineering, Yeungnam University, Taegu, Korea

Yun C Chung
Dept of CAD/CAE/CAM Research & Development, Chrysler Technology Center

Bo H Kim, and Byoung K Choi
Dept of Industrial Engineering, Korea Advanced Institute of Science & Technology

Key words: Pencil-curve, digitizing, CAD/CAM, Extended Z-map.

Abstract: Pencil-curve machining, which is a single-pass ball-endmilling along a concave edge on a die surface, is widely employed in die-surface machining. The cutter-path used for pencil-curve machining, which is the trajectory of the "ball-center point" of a ball-endmill sliding along a concave-edge region on the die surface, is called *pencil-curve*. Presented in the paper is a pencil-curve tracing algorithm in which "concave-type" sharp edges are computed from a "virtually digitized" model of the tool-envelope surface. The resulting "initial" pencil-curves are then refined by applying a series of fairing operations. Illustrative examples and methods for enhancing accuracy are also presented. The proposed pencil-curve tracing algorithm has been successfully implemented in a commercial CAM system specialized in die-machining and in the CAD/CAM system CATIA®

1. INTRODUCTION

In other SFF processes, parts need to be decomposed to meet layer thickness and surface finish requirements. In SDM process, parts (and support material) are decomposed primarily to eliminate accessibility problem in machining. Geometric conditions that require part
Pencil-curve machining, which has long been popular among car-body stamping-die makers, is a single-pass 3-axis ball-endmilling along a concave edge on a die surface. It is applied either 1) to relieve "cutting-loads" at the concave edge region during a subsequent regional milling or 2) to clean up "uncut volumes" at the concave edge region using smaller ball-endmills after the regional milling. The former type is often called *relief pencil-cutting* and the latter *clean-up pencil-cutting*. At Toyota Motor Corporation, for example, the "relief-type" pencil-curve machining has been extensively employed in machining stamping dies for inner panel parts[1]. Pencil-curve machining is considered as one of the key functions required for a CAM system specialized in die-machining. Examples of such CAM systems include Clicks[2] from Japan, Z-Master[3] from Korea, Work-NC[4] from France, and Tebis[5] from Germany.

Considering its importance in NC cutter-path generation for die-machining, it is surprising that no publications describing how to find pencil-curves are available in open literature (to the best of our knowledge). Presented in the paper is a straightforward *pencil-curve tracing* method which is quite efficient and robust. It has been implemented in a commercial CAM system[3] and, recently, has also been ported into the CATIA System (Dassault Systems, France) as well.

2. PROBLEM DEFINITION AND OVERALL PROCEDURE

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35392-0_40

G. J. Olling et al. (eds.), *Machining Impossible Shapes*

As indicated in Figure 1 (the dotted lines), the trajectory of the "ball-center point" of a ball-endmill sliding along a concave edge region on the part-surface becomes a **pencil-curve**. It is assumed that the **part-surface** is represented by a collection of trimmed parametric surfaces. In the following, we will use the term "surface" for a trimmed parametric surface $\mathbf{r}(u,v)$, such as a Bezier or NURB surface. In the case of a car-body stamping-die, a part-surface consists of a large number of surfaces (up to over 3,000 surfaces in a typical inner-panel die). The pencil-curve tracing problem may be formulated in a number of ways:

- SSI (surface/surface intersection) problem: all the offset-surface intersection curves[6] as well as self-intersection curves are computed and then they are linked to form pencil-curves.
- Circular blending problem: the so called "rolling-ball" or "ball-position sampling" method[7,8] is applied in order to numerically trace out the ball-center points.
- Sharp-edge detection problem: a discrete model of the *tool-envelope surface* is constructed and then "concave-type" sharp-edges are traced out from the discrete model.

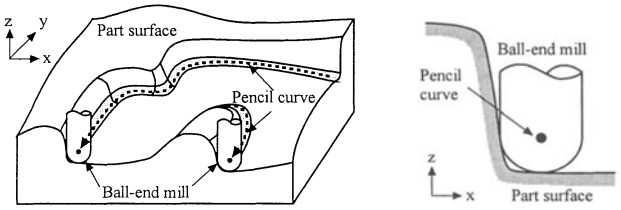


Figure 1. Pencil-curve Tracing Problem

The third approach is adopted in the paper mainly because the first two approaches would suffer from numerical singularities when the *fillet-radius* of the concave edge region approaches to the radius of the ball-endmill (Unfortunately, most of the pencil-curve machining operations are performed under this condition). In the third approach, the tool-envelope surface may be represented either as a "triangulated" facet model[9,10] or as a "z-map" model[11]. The z-map representation is used in this paper.

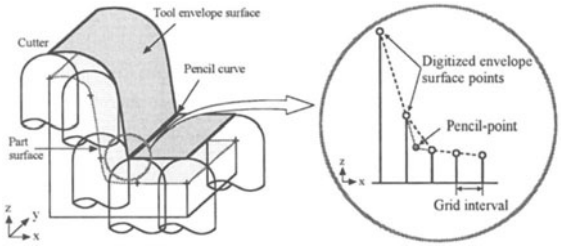


Figure 2 Concave-edge Tracing on the Tool-envelope Surface (TES)

As depicted in Figure 2, a **tool-envelope surface** (TES) is the *envelope* of "cutter swept-volumes" which are generated by sweeping an "inverse" ball-endmill over the part-surface with its ball-center positioned on the part-surface. Since it is not easy to find a mathematical representation of the TES, the trick is to "virtually digitize" it and then store the digitized data in a **z-map** which is nothing but a two-dimensional real array where the z-values at *grid-points* on the x,y-plane are stored (More details about the *virtual digitizing process* will be given in the next section). With the z-map model of the TES on hand, the remaining problem

is to detect concave-type sharp-edge points called **pencil-points** from each row and column of the z-map array (See graph inside the circle in Figure 2).

An initial pencil-curve is then constructed as a sequence of the pencil-points. Finally, the initial pencil-curve has to be smoothed out while avoiding possible *cutter inteferences*[12] (to be used as a cutter-path). This step is called a *pencil-curve refinement*. In summary, pencil-curves are generated in three phases: 1) virtual digitizing of TES to obtain a z-map, 2) detection and tracing of concave-type sharp-edges from the z-map to obtain initial pencil-curves, and 3) refinement of the initial pencil-curves.

3. VIRTUAL DIGITIZING OF TOOL-ENVELOPE SURFACE

In order to apply a (virtual) digitizing process, the part-surface is assumed to be visible from above (which is the case with most of the stamping-dies). There are at least three ways of obtaining a z-map of the tool-envelope surface (TES):

- Obtain a z-map of the part-surface and then apply the so called *inverse offsetting method*[13].
- Facetize individual offset surfaces and then construct a z-map from the facet models.
- Apply a direct *cutting simulation*[14,15] by moving the inverse ball-endmill over the part-surface.

In practical applications, each of the three methods has its strength and weakness in terms of accuracy, computation time, robustness, etc. Obviously, there is a need for further research as to which one is better under what situations. The first method is simple to implement but least accurate, while the second method is very simple in concept but needs more programming efforts. The third method is a straightforward application of cutting simulation (but, probably most time-consuming).

Due to space limitation, only the first method will be described in this section. The z-map model of a TES is obtained in two steps: *virtual digitizing* of part-surface and z-map offsetting.

3.1 Virtual Digitizing of Part-surface: Z-map Sampling

In order to define the 2D domain of a z-map, a rectangular "majorizing box", in which the part-surface is contained, is defined on the x,y-plane. Let (x_0, y_0), W, and H denote the bottom-left corner point, width, and height of the 2D majorizing box, respectively, then the coordinates (x_i, y_j) of a grid point (i,j) are given by

$$x_i = x_0 + \Delta \cdot i \quad \text{and} \quad y_j = y_0 + \Delta \cdot j \quad \text{for } i \in [0,M] \text{ and } j \in [0,N], \tag{1}$$

where Δ is the *grid interval*, I and j are (integer) index variables, and the integers M (=W/Δ) and N(=H/Δ) denote the sizes of the z-map array.

Now, for each trimmed surface $\mathbf{r}(u,v) = (x(u,v), y(u,v), z(u,v))$, a closed 2D-curve C representing its trim-boundary is constructed on the x,y-plane. Then, for each grid point (x_i, y_j) located inside C, its z-value $Z[i,j]$ is computed by using the so called *2D-Jacobian inversion algorithm* (See, for example, page 305 of Reference 11) which is a Newton's iteration scheme for finding an intersection point between a vertical line and a parametric surface. If the numerical scheme fails at a grid-point, the parametric surface may be facetized and then the z-value at the grid-point may be obtained from the facet model.

3.2 Z-map Offsetting

The z-map of the TES is obtained by offsetting the z-map of the part-surface, $Z_p[i,j]$. A simple method of offsetting a z-map is the so called *inverse offset method*[12], in which the z-value of the "offset z-map" at a grid point (m,n) is expressed as follows[11]:

$$Z_o\ [m,n] = \max\ \{\ (\ Z_p[i,j] + B(i,j,m,n,R)\ ) :\ (i,j) \in I(m,n,R)\ \} \quad (2)$$

where, $I(m,n,R) = \{\ (i,j)\ |\ ((x_i - x_m)^2 + (y_j - y_n)^2) \le R^2\ \}$,

$B(i,j,m,n,R) = (R^2 - ((x_m - x_i)^2 + (y_n - y_j)^2))^{1/2}$,

x_i, x_m, y_j, y_n are given by (1),

R : offset distance (or ball radius).

In (2), the quantity "$Z_p[i,j]+B(i,j,m,n,R)$" denotes the z-value, at the grid-point (m,n), of the upper half of the ball whose center is positioned at $\mathbf{p}_{ij} = (x_i, y_j, Z_p[i,j])$.

The offset z-map expression (2) has a straightforward form, but it would take too much time when (2) is directly applied in obtaining an offset z-map. For example, the expression $B(i,j,m,n,R)$ is evaluated $(W/\Delta)(H/\Delta)(2R/\Delta)^2$ times, which is about 13 billion for an actual application case to be introduced later. Thus, in actual implementation, various schemes for reducing execution time are utilized. Some details of the inverse offset method may be found in Reference 11 (pp.360-361).

4. PENCIL-CURVE DETECTION AND TRACING

Described in this section is a procedure for detecting and tracing concave-type sharp edges (for which the term **concave-edges** will be used hereafter) from the z-map model of a tool-envelope surface (TES). It should be noted that a TES has no "convex-type" sharp-edges because it is an offset surface. Shown in Figure 3-a is a vertical cross-section of a z-map model in which a **concave-point** is denoted by a solid dot. A **vertical cross-section** (VCS) is constructed from the point sequence $\{\mathbf{p}_{ij} = (x_i, y_j, Z[i,j])\}$ by fixing "$i=i^*$ or $j=j^*$": x-directional VCS at the $j=j^*$ *grid-line* on the x,z-plane; y-directional VCS at the $i=i^*$ grid-line on the y,z-plane as shown in Figure 3-b. For each point on the VCS, a **concave-angle** α is defined as depicted in Figure 3-a. As a convention, *the concave-angle for a convex-point is set to zero.*

Among a sequence of consecutive concave-points, the one with the largest concave-angle is called **maximal concave-point**. Now we define the four concave-angles (α_j for j=1,2,3,4) around the maximal concave-point as follows (See Figure 3-a):

- α_1 = *Maximal concave-angle* : the concave-angle at the maximal concave-point.
- α_2 = *2nd-maximal concave-angle* : the larger one among the two angles adjacent to "α_1".
- α_3, α_4 = *Neighbor concave-angles* : the two concave-angles adjacent to "α_1 and α_2"

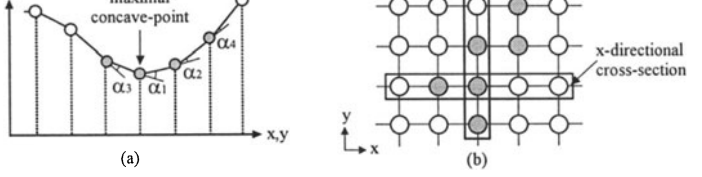


Figure 3 Construction of a Vertical Cross Section (VCS)

4.1 Basic Algorithm for Finding a Pencil-point

Before presenting the procedure for tracing out pencil-curves, the key algorithm for finding a pencil-point from a VCS will be described. We want to decide whether there exists

a pencil-point at (or near) the maximal concave-point based on the four concave-angles (α_j for j=1,2,3,4).

Since we are dealing with "virtually digitized" data in which some "digitizing" errors may be contained, the "pencil-point detection problem" is more like a "hypothesis testing problem" in statistics. That is, the hypothesis that "there exists a pencil-point " is accepted if the following *sharpness condition* holds:

$$(\alpha_1 + \alpha_2) > \kappa_s, \tag{3}$$

where κ_s is the "sharpness criterion", and α_1, α_2 are maximal and 2nd-maximal concave-angles. A pencil-point is obtained only when the sharpness condition (3) is satisfied. The maximal concave-point becomes an "on-grid" pencil-point if the sharpness condition (3) is satisfied and the following *on-grid condition* holds (γ: on-grid criterion):

$$\alpha_1 > \kappa_g \cdot \alpha_2. \tag{4}$$

where $\kappa_g > 1$ is the "on-grid criterion". An on-grid pencil-point is shown in Figure 4-a. Otherwise, an "off-grid" pencil-point is newly defined in-between the maximal concave-point and the 2nd-maximal concave-point as shown in Figure 4-b. Observe that the concave-angle of the off-grid pencil-point is $\alpha_1+\alpha_2$. In the example case of Figure 4, κ_g is set to 4.

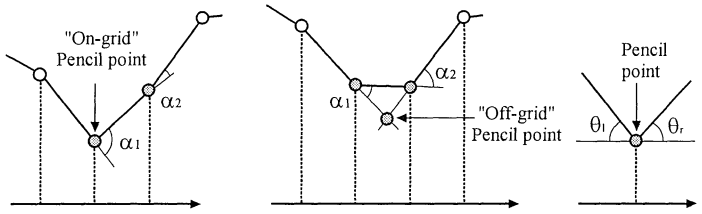


Figure 4. Determination of 3D Pencil-point and Wall-direction

The role of the "inclination-angles" θ_l and θ_r shown in Figure 4-c is explained below. Associated with a pencil-point are three attributes – "marching-direction", "wall-location", and "quality" – which may be expressed in Backus-Naur Form as:

<marching direction> ::= "+y" | "–y" | "+x" | "–x".
<wall-location> ::= "Right" | "Left" | "Undecided".
<quality> ::= "Gold" | "Silver" | "Bronze" | "Clay".

The **marching direction** (MD) of an x-directional pencil-point is either "+y" or "–y", and the MD of a y-directional pencil-point is "+x" or "–x". Let θ_l and θ_r , respectively, denote inclination angles of the left and right line segments meeting at the pencil-point as shown in Figure 4-c. Then, when viewed toward its marching-direction, the **wall-location** of a pencil-point may be determined by using following heuristic rule:

$$\begin{aligned} &\text{"}Left\text{"} \quad if\ \theta_l > \kappa_w \theta_r \\ &\text{"}Right\text{"} \quad if\ \theta_r > \kappa_w \theta_l \\ &\text{"}Undecide\text{"}\ \ Otherwise \end{aligned} \tag{5}$$

where $\kappa_w > 1$ is the "wall-location criterion". In pencil-curve machining, the direction of a pencil-curve should be set such that the wall is located to the right side of the pencil-curve (in order to provide a "down-milling" mode).

The **quality** of a pencil-point is graded as "Gold", "Silver", "Bronze", or "Clay". Again, we use the following heuristic rule in determining **pencil-point quality**:

$$
\begin{aligned}
&"Gold" && \text{if } \alpha_2 = 0 \\
&\text{else } "Silver" && \text{if } (\alpha_3 + \alpha_4)/(\alpha_1 + \alpha_2) \le \varepsilon_s \\
&\text{else } "Bronze" && \text{if } (\alpha_3 + \alpha_4)/(\alpha_1 + \alpha_2) \le \varepsilon_b \\
&\text{else } "Clay"
\end{aligned}
\tag{6}
$$

where ε_s and ε_b are the "silver-point criterion" and "bronze-point criterion", respectively.

The values for the criteria used in the above expressions (3) to (6) may be provided by the "user" to suit his/her purposes. In the pencil-cut tool-path generation for car-body stamping-dies, the following:

- Sharpness criterion in (3): $\kappa_s = 20°$.
- On-grid criterion in (4): $\kappa_g = 4$.
- Wall-location criterion in (5): $\kappa_w = 2$.
- Silver-point criterion in (6): $\varepsilon_s = 0.01$.
- Bronze-point criterion in (6): $\varepsilon_b = 0.07$.

However, it should be noted that each of the concave-angles α_i appearing in the above expressions is a "nominal" angle defined on a vertical cross-section which is obtained by intersecting the z-map surface with a vertical plane. Thus, in order to find a "true" concave-angle, the z-map surface has to be intersected by a tilted plane which is perpendicular to the pencil-curve at the current concave-point $\mathbf{p}_{ij}$.

A simple method of constructing the tilted intersecting plane is depicted in Figure 5. Shown in Figure 5-a is the case where the nominal concave-angle α at $\mathbf{p}_{ij}$ is defined on an x-directional VCS (The construction for a y-directional VCS is the same). As depicted in Figure 5-a, a tilted plane is defined by rotating the vertical plane around the "axis of rotation" (the horizontal line passing through $\mathbf{p}_{ij}$) by β degrees. These vertical and tilted planes appear as lines on the yz-plane as shown in Figure 5-b. Then, a reasonable choice for the tilted intersecting plane is the "bisector line" at $\mathbf{p}_{ij}$ on the y-directional VCS, and the draft angle of the bisector-line becomes the **tilt-angle** β. Once the tile-angle β is known, the "true" concave-angle α_t for a given nominal concave-angle α can be approximated as follows:

$$\alpha_t = 2 \tan^{-1}(\tan(\alpha/2) \cos\beta) \tag{7}$$

which may be verified from a trigonometric construction involving α,β. From now on, it is assumed that all the concave-angles in (3) to (6) are true concave-angles obtained from (7).

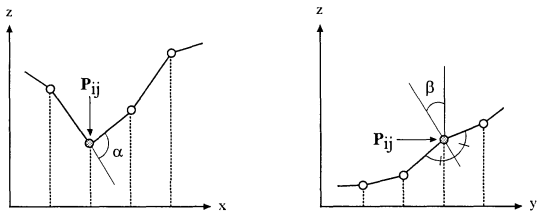


Figure 5. Concave-angle and Tilt-angle

4.2 Dectection of initial pencil-point

In order to speed up the pencil-curve tracing process, the concave regions where the pencil-points might be located are marked in the *mark-map* array M[*i,j*]. The mark-map constructing procedure shown in Figure 6 may be summarized as follows:

Construct_Mark-map(Z[*i,j*], $\rho_{fillet} \rightarrow$ M[*i,j*]) ;

1. Input: Z-map model Z[*i,j*] and filleting radius ρ_{fillet} ;
2. Generate a *filleted z-map,* F[*i,j*] from Z[*i,j*] via an upward-offsetting followed by a downward-offsetting using the same *offset-distance* ρ_{fillet} ;
3. Generate a *difference z-map,* D[*i,j*] by subtracting Z[*i,j*] from F[*i,j*] for all *i,j* ;
4. For all *i, j* do { if D[*i,j*]>ε and "Z[*i,j*] is concave in x- or y-directional VCS" then mark the grid-point (i.e. set M[*i,j*]=1), else set M[*i,j*]=0 } ;

A reasonable choice for the filleting radius would be "three grid-intervals" ($\rho_{fillet} = 3\gamma$, where γ is the grid-interval).

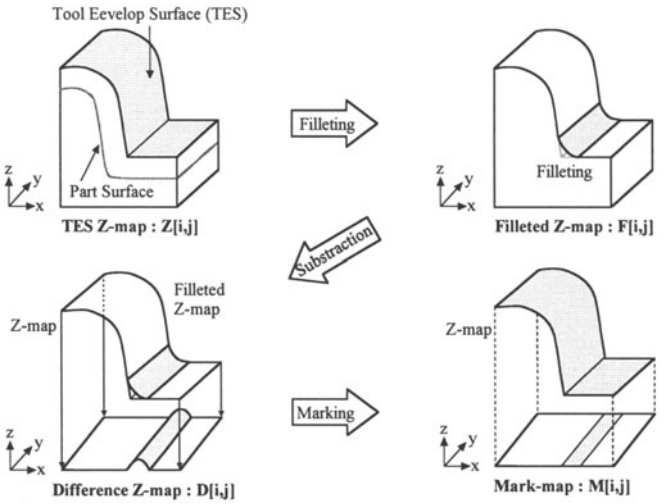


Figure 6. Steps for Generating a Mark-map

Further, before describing the main procedure for detecting an initial pencil-point, a function returning a pencil-point (together with wall-location and quality) for given MCP (maximal concave-point) and MD (marching direction) is presented first. From the results presented in the previous section, the function "Get_pencil-point" may be expressed as follows:

Get_pencil-point(MCP, MD→ Pencil-point, Wall-location, Quality**) ;**

1. Input: MCP (maximal concave-point) ; MD (marching direction) ;
2. Compute the "true"concave-angles α_j for *j*=1,2,3,4 at the MCP;
3. If the sharpness condition (3) is not true, then Pencil-point ⇐ "Null" and stop ;
4. If the on-grid condition (4) is true, then Pencil-point ⇐ MCP,
   else the Pencil-point is determined from the construction given in Figure 4-b.
5. Determine the Wall-location with respect to the MD by using the rule (5).
6. Determine the Quality by applying the rule (6).

With the above algorithm on hand, we are ready to the main algorithm for detecting an initial pencil-point. Since all the pencil-points are confined in the "marked regions" consisting of marked grid-points (M[*i,j*] ≡ 1), the pencil-point detection process starts from a marked grid-point: Starting from a marked grid-point, the x-directional and y-directional vertical cross-sections (VCSs) are scanned to find a maximal concave-point (**MCP**). And then, the VCS on which the MCP is found is examined closely to detect an initial pencil-point (**IPP**). The step-by-step procedure for detecting an IPP and setting its initial marching-direction (**IMD**) may be expressed as follows:

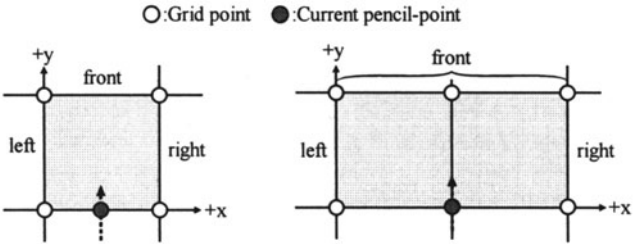
Detect_initial-pencil-point(M[*i,j*] → IPP, IMD, Wall-location, Quality, M[*i,j*]**) ;**

1. Input: mark-map M[*i,j*] ;
2. Find a marked grid-point from M[*i,j*]; If not found, then IPP ⇐ "Null" and stop ;
3. For the marked grid-point, find an MCP from the x-directional VCS and then
   1) set its IMD_x to "+y" or "–y" and
   2) call Get_pencil-point(MCP,IMD_x ⇒ IPP_x, Wall-$location_x$, $Quality_x$) ;
4. For the marked grid-point, find an MCP from the y-directional VCS and then
   1) set its IMD_y to "+x" or "–x" and
   2) call Get_pencil-point(MCP,IMD_y ⇒ IPP_y, Wall-$location_y$, $Quality_y$) ;
5. If (IIP_x ≡ IIP_y ≡ "Null") or ($Quality_x$ ≡ $Quality_y$ ≡ "Clay") then reset the marked grid-point to zero and go to Step 2.
6. If $Quality_x$ is better than $Quality_y$ then z ⇐ x, else z ⇐ y.
7. Return (IPP_z, IMD_z, Wall-$location_z$, $Quality_z$).

It should be noted in Steps 3 and 4 that there always exists a maximal concave-point (either in Step 3 or in Step 4 or both) and that the selection of an initial marching-direction is made arbitrarily (the other direction will be tried anyway). The above algorithm will provide a valid IPP (initial pencil-point) unless there are no marked grid-points in the mark-map M[*i,j*], namely, there are no pencil-curves in the z-map.

4.3 Pencil-curve Tracing

Starting from the initial pencil-point, a complete pencil-curve can be traced by repeatedly marching toward the "next" pencil-point until nowhere to go. As shown in Figure 7, a **marching-cell** is defined in front of the current pencil-point: a "single cell" for an off-grid pencil-point (Figure 7-a) and a "double cell" for an on-grid pencil-point (Figure 7-b). A pencil-curve entering into a marching-cell through the current pencil-point leaves the cell through the next pencil-point which should be located on one of the three **cell-sides**: the Left-side, the Front-side, or the Right-side.



(a) Single cell (Off-grid pencil-point) (b) Double cell (On-grid pencil-point)

Figure 7 Marching-cell for Pencil-curve Tracing (Current marching-direction = "+y")

The **exiting directions** for the left-side, front-side, and right-side, respectively are called Left-direction (LD), Front-direction (FD), and Right-direction (RD). Note in Figure 7 where the current marching-cell is "+y", the three exiting-directions are "–x" (LD), "+y" (FD) and "+x" (RD). In general, for a given CMD (current marching-direction), the values of the exiting directions are given by:

If (CMD ≡ "+x") then {LD = "+y"; FD = "+x"; RD = "–y"}
If (CMD ≡ "–x") then {LD = "–y"; FD = "–x"; RD = "+y"} (8)
If (CMD ≡ "+y") then {LD = "–x"; FD = "+y"; RD = "+x"}
If (CMD ≡ "–y") then {LD = "+x"; FD = "–y"; RD = "–x"}

Once the next pencil-point is located in one of three cell-sides, it becomes the current pencil-point and the exiting-direction becomes the current marching-direction, and so on. During the

marching process, the current pencil-point is stored in a linked-list data structure called a **pencil-curve-list** and all the marked grid-points on the current vertical cross-section (VCS) are reset.

Based on the discussions so far, we now present a complete algorithm for making a single-step marching to the next pencil-point. That is, for given values of the current pencil-point and current marching-direction, the next pencil-point and next marching-direction are returned. The following abbreviations are used in the algorithm:

- CPP and NPP = current & next pencil-points;
- CMD and NMD = current & next marching-directions;
- MCP = maximal concave-point;
- LD, FD and RD = Left-, Front- and Right-directions;
- VCS = vertical cross-section.

Marching_next-pencil-point(CPP, CMD→ NPP, NMD**)** ;

1. Input: CPP (current pencil-point); CMD (current marching-direction) ;
2. For given CPP and CMD, construct a marching cell and assign its exiting directions (i.e. the values of LD, FD, and RD) according to (8).
3. If an MCP is found on the Left-side, call **Get_pencil-point(**MCP,LD→NPP_l, **..)**.
4. If an MCP is found on the Front-side, call **Get_pencil-point(**MCP,FD→NPP_f, **..)**.
5. If an MCP is found on the Right-side, call **Get_pencil-point(**MCP,RD→NPP_r, **..)**.
6. If no NPP is found from the above three steps, return (NPP = "Null") and stop.
7. If more than one valid NPP are obtained, then the one having the same Wall-location as that of the CPP is selected and return its NPP and NMD.
8. Store the CPP in the pencil-curve-list and reset all the marked grid-points located on the current VCS.

The above marching operation is repeatedly invoked until 1) no next pencil-point is found (i.e. NPP ≡ "Null") or 2) the initial pencil-point is encountered (i.e. NPP ≡ IPP). In the first case, the marching operation is restarted at the IPP with its marching direction reversed. Finally, the overall procedure for tracing all the pencil-curves in the z-map may be summarized as follows:

Procedure_pencil-curve_detection_and_tracing() ;

1. Construct_Mark-map(Z[*i,j*], ρ → M[*i,j*]) ; // ρ=3γ: fillet radius) //
2. Detect_initial-pencil-point(M[*i,j*] → IPP, IMD, Wall-location, Quality, M[*i,j*]) ;
3. If (IPP ≡ "Null") then stop.
4. Pass = "First-pass"; // Set flag for restarting the marching operation //
5. CPP = IPP ; CMD = IMD; // Set current pencil-point and marching direction //
6. Repeat {Marching_next-pencil-point(CPP,CMD→NPP,NMD);CPP=NPP;CMD=NMD}
   until (NPP ≡ "Null") or (NPP ≡ IPP) ;
7. If (NPP ≡ IPP) then {Go to Step 2}; // A *closed pencil-curve* has been obtained //
8. If (NPP ≡ "Null") and (Pass ≡ "First-pass")
   then { Pass = "Second-pass"; CPP = IPP; CMD = –IMD; Go to Step 6}
   else {Go to Step 2} ; // An open pencil-curve //

The result of the above procedure is a set of initial pencil-curves. Since the initial pencil-curves usually contain "errors", the resulting pencil-curves are "refined" further in order to be used as pencil-cut tool-paths.

5. PENCIL-CURVE REFINEMENT

The pencil-curves generated by the "pencil-curve detection & tracing" procedure may not be suitable for machining purposes because of the "sharp oscillations" in the pencil-curves. Thus, the initial pencil-curves are refined in order to obtain *smooth* and *gouge-free* pencil-

curves without compromising accuracy (The accuracy could be improved in some cases). The refinement operation is carried out in three steps: preprocessing, pencil-points fairing, and post-processing.

5.1 Pre-processing of the Initial Pencil-curves

The purpose of pre-processing is to obtain "useful" pencil-curves from the initial pencil-curves. The pre-processing step consists of 1) deletion of "low quality" pencil-curves, 2) curve-direction setting, and 3) curve concatenation. A pencil-point whose quality is "Clay" is called **clay-point** and the percentage of clay-points in a pencil-curve is called **clay-ratio**. A pencil-curve whose clay-ration is too high or length is too short may have little use in actual machining. Utilizing the "quality" and "wall-location" of each pencil-point, the following operations are carried out during the pre-processing phase:

1. Delete the curve-segments consisting of more than n_c consecutive clay-points.
2. Delete the pencil-curve whose clay-ratio exceeds r_c.
3. Delete the pencil-curve whose length is less than l_p.
4. Set the direction of each pencil-curve, if necessary, such that its wall is located on the right-hand side of the curve (Wall-location ≡"Right").
5. Concatenate two pencil-curves if their distance (i.e. the distance between their end-points) is less than a certain value l_c (and if their *end tangents* are collinear).

The "sensitivity" of the pre-processing can be adjusted by changing the values of the "pre-processing tolerances". The following values may be used as a "default":

- Clay-segment tolerance: $n_c = 10$.
- Clay-ratio tolerance: $r_c = 50\%$.
- Pencil-curve length tolerance: $l_p = 10\gamma$ (γ is the grid-interval).
- Concatenation tolerance: $l_c = 5\gamma$.

5.2 Pencil-points Fairing and Post-processing

Since the pre-processed pencil-curve may have sharp oscillations (fluctuations), the pencil-points fairing operation is performed by using a systematic fairing scheme based on **difference operators**. For a 3D point sequence $\{\mathbf{r}_j\}$, the n-th difference is defined as

$$D_j^n = D_{j+1}^{n-1} - D_j^{n-1} \; with D_j^0 = \mathbf{r}_j . \tag{9}$$

By setting (9) to zero at n=2, the "ideal" position ($\mathbf{r}_j'$) of the input point $\mathbf{r}_j$ is given by

$$\mathbf{r}_j' = (\mathbf{r}_{j+1} + \mathbf{r}_{j-1})/2, \tag{10-a}$$

Similarly, by setting (9) to zero at n=4, the "ideal" position ($\mathbf{r}_j'$) is expressed as

$$\mathbf{r}_j' = (\mathbf{r}_{j+1} + \mathbf{r}_{j-1})/2 + \{(\mathbf{r}_{j-1} - \mathbf{r}_{j-2}) + (\mathbf{r}_{j+1} - \mathbf{r}_{j+2})\}/6. \tag{10-b}$$

The physical meaning of the 2nd-difference fairing (10-a) and 4th-difference fairing (10-b) is shown in Figure 8: The 2nd-difference fairing would straighten the curve, while the 4th-difference fairing would lead to a "linear-curvature" curve if the point spacing is uniform. A quantity similar to the sum-of-squares of (10-b) is often used as a global smoothness measure of a digitized curve[18]. For a local straightening we use the 2nd-difference fairing equation (10-a), and the 4th-difference fairing (10-b) is employed for the global smoothing.

The 3D coordinates $\mathbf{r}_j = (x_j, y_j, z_j)$ of pencil-points can be decomposed into "domain" coordinates $\mathbf{p}_j = (x_j, y_j)$ and "height" $\mathbf{q}_j = (s_j, z_j)$, where s_j is the cumulative length of the pencil-curve given by

$$s_j = \Sigma |\, \mathbf{p}_i - \mathbf{p}_{i-1} | \;\; \text{for} \; j=1,2,.. \; \text{with} \; s_0=0. \tag{11}$$

Our strategy is to apply the point-data fairing operations separately to the domain-coordinates $\{\mathbf{p}_j\}$ and the height-coordinates $\{\mathbf{q}_j\}$. In both cases, the fairing operation is performed in two steps:

1. Local straightening of the "sharp oscillation".
2. Global smoothing of the straightened 2D point sequence.

The smoothed pencil-curves may further be refined to make them more suitable for pencil-cut machining. Additional refinement operations include: 1) spline-curve fitting, 2) remeshing and 3) gouge-checking. The point data defining a pencil-curve may first be fitted into piecewise cubic spline curves or rational B-splines[19,20].

Once a pencil-curve is represented as a composite parametric curve, a "remeshed" pencil-curve may be obtained by sampling evenly spaced 3D points from the parametric curve (As a rule of thumb, the distance between sampled points may be set to $\gamma/2$, one half of the grid-interval). Further, the remeshed pencil-points may be "reduced" by applying a piecewise linear approximation within a given tolerance (of 0.001mm as a default value).

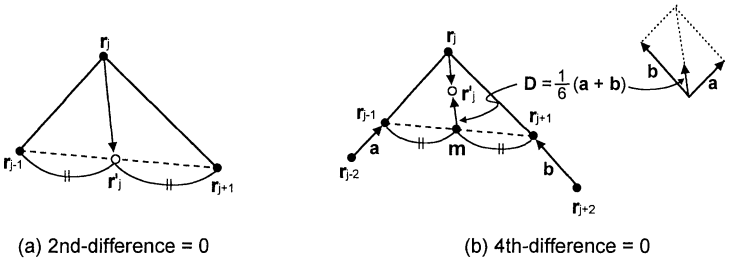


Figure 8 Physical meaning of the difference fairing

The final step in pencil-curve refinement is to make a gouge checking for each point in the pencil-curve: locate the cutter-center point at the pencil-point and then make an interference check to see the bottom of the ball-endmill gouges the part-surface as depicted in Figure 9. If a gouging is detected, the cutter may be lift slightly as depicted in the figure.

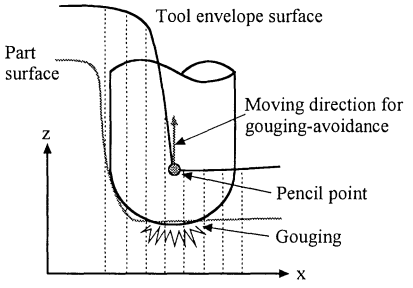


Figure 9. Gouge Checking

6. ACCURACY ENHANCEMENT

The proposed pencil-curve tracing scheme has been implemented in a commercial CAM system[3] which is currently being used in stamping-die machining by a number of industrial users. Obviously, the quality of the pencil-curves is dependent on the value of the z-map grid interval γ: the pencil-curve accuracy is improved as γ is decreased. However, as γ is decreased, both the computation time and memory requirement would increase rapidly. For a given die-surface size, the memory used up by a z-map model is proportional to $(1/\gamma)^2$ and the

execution time for a z-map offsetting operation (an important part of pencil-curve tracing) is roughly proportional to $(1/\gamma)^4$. In particular, it was observed that the maximum allowable size of a single z-map for an EWS having 100 MB of main memory was about 40 MB (Two z-map models are needed for an offsetting operation). For these reasons, a practical range of γ for car-body stamping-die applications is 0.4mm to 0.8mm, depending on the size of the die.

However, with the "allowable" value of grid interval ($0.4<\gamma<0.8$), the quality of the pencil-curve may be inadequate for a "clean-up pencil-cutting" application. The situation may be handled in two ways: 1) to use the pencil-curve as a "seed curve" for other numerical methods or 2) to employ adaptive sampling schemes. We use the second approach, while the first approach is used in CATIA®.

We have developed the pencil-curve generating module in the CATIA function of "Mold and Die Manufacturing" (MFG-DIES) in which the sub-process (Local Rework) uses the pencil-curves as "seed curves" for pencil machining or transversal guided cutting (clean-up cutting). Most procedures in the paper were adopted, while fairing operations and post-processing of pencil-points were omitted by intention. Pencil-curve tracing algorithms were developed as an extra execution program by ANSI-C, and a few interfacing functions by FORTRAN. The pencil-curves are generated and imported into CATIA as CRV (curve) elements each containing a sequence of 3D pencil-points. Finally, pencil-curves are used as "seed curves" for pencil-machining, or those for detecting unmachined areas that are to be milled by smaller tools.

There are at least three different adaptive sampling schemes: 1) an adaptive subdivision scheme such as the use of quad-trees, 2) addition of smaller z-maps at the marked regions, and 3) additional sampling at the "grid-edges" in the marked regions. We have considered all the three methods, but the third method has been adopted because it is easier to implement (than the first method) and faster to execute (than the second method).

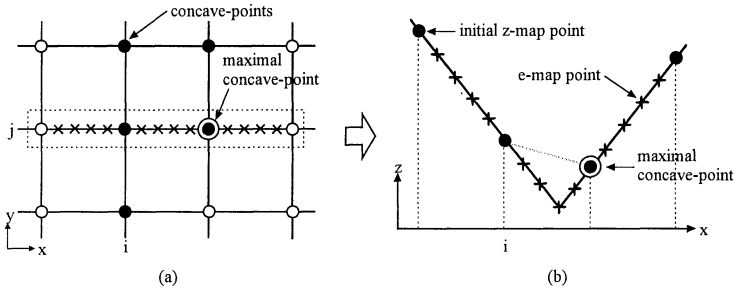


Figure 10. E-map Point Sampling for Accuracy Enhancement

The basic idea of the "additional sampling method" is depicted in Figure 10 where an x-directional vertical cross-section (VCS) is shown. A line segment joining two grid-points is called **grid-edge**. A fixed number of additional sampling positions are defined on each of the grid-edges connected to the concave-points on the VCS as shown in Figure 10-a, and then the z-values of the tool envelope surface (TES) at the additional sampling positions are digitized. The z-values sampled at the grid-edges are called **e-map points**. Now the procedure for "pencil-curve detection and tracing" (with a minor modification) is applied to the VCS shown in Figure 10-b to obtain initial pencil-curves. Then, the initial pencil-curves are refined as before. In actual implementation, 9 e-map points are sampled from each grid-edge on the VCS, which found to be more than enough for most of the die-machining applications. As a result, the accuracy (or precision) of the pencil-curve is increased 10 times with a small (about 50 percents for inner-panel stamping-die machining) increase in memory requirement.

Without the use of an e-map, the memory requirement could have been increased 100 times to achieve the same accuracy.

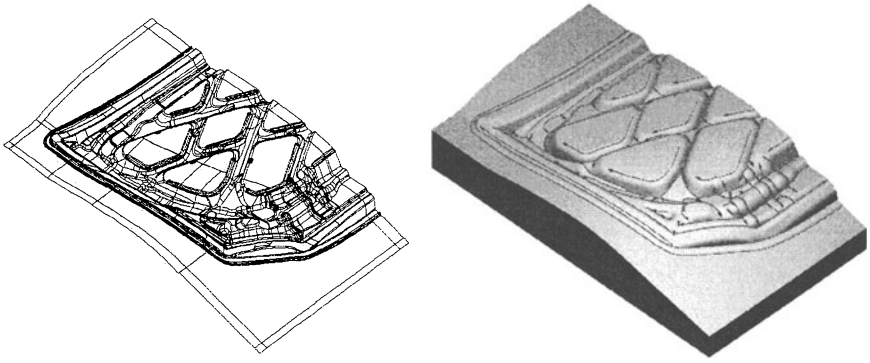
7. ILLUSTRATIVE EXAMPLES

Shown in Figure 11-a is a die surface (for an inner panel of a passenger car) consisting of 2,226 surfaces. The die surface dimensions are: W=836.5mm and H=1461.6mm. With the grid-interval of 0.7mm, the z-map of the die surface takes up 9.98 mega-bytes (MB) of main memory.

The job was to generate "relief-type" pencil-cut tool-paths for a 50ϕ ball-endmill (R=25mm). As shown in Figure 11-b, a total of 96 pencil-curves are obtained from the CL z-map. Running on an IBM RS/6000-365 EWS, actual computation times were collected as:

1) Z-map offsetting (to obtain the CL z-map): 330 sec.
2) Mark-map generation: 186 sec.
3) Initial pencil-curve generation: 76 sec.
4) Pencil-curve refinement: 2497 sec.

Thus, starting from the z-map of the die surface, the total time for generating the 96 pencil-curves is about 52 minutes (among which about 80% is spent in refining the pencil-curves).



(a) CAD-data (b) Pencil-curves on the CL Z-map

Figure 11. Car-body Inner Panel: (a) CAD Model and (b) Pencil-curves

8. CONCLUSIONS AND DISCUSSIONS

A systematic procedure for finding pencil-curves from the CAD model of a die surface has been described in the paper. The distinctive feature of the proposed method is that the pencil-curve tracing problem is formulated as a *sharp-edge detection problem* in which smooth pencil-curves are obtained in three steps: virtual digitizing of the tool-envelope surface; detection and tracing of initial pencil-curves; and refinement of pencil-curves. The validity and robustness of the proposed method has been tested in commercial CAD/CAM systems. The heuristic rules applied in the research have been developed and tested mainly for machining car-body stamping-dies, in which most pencil-curves with various tool diameters (e.g., 8~30mm) could successfully be obtained. There may be, however, some difficulty in

tracing the pencil-curves with small tool diameter on a large area of part-surface due to the z-map resolution (i.e., grid-interval vs. memory requirement).
In addition, the *concave-edge tracing method* and the *point data fairing method* introduced in the paper may also be applicable to the processing of "physically" digitized data as well. However, further research efforts are needed to improve the quality of the pencil-curve and computational efficiency.

The method may be applied to 5-axis machining in some restricted environment. For example, consider a transition fillet between turbine blade surface & hub surface. The ball-nosed tool center points (pencil-curves) for the surface can be traced as long as the pencil-machining area can be viewed along a certain fixed tool-axis, in which z-map models for part-surface and ball-offset surface can be constructed, followed by tracing the pencil-curves on the area. Additional problems, however, should be tackled with care such as selecting cutter-axis at each pencil-point, avoiding cutter-interference, etc.

REFERENCES

1. Ikemoto, K *et al.* 'Development & practical application of advanced flexible production system for a year-around continuous operation' *Proc. of IFIP CAPE Conference,* (1991), pp.97-104
2. *Clicks User Manual,* ARGO Tech., Japan (1990)
3. *Z-Master Reference Manual,* Cubic Tek, Korea (1992)
4. *Work-NC User Guide,* Sescoi, France (1993)
5. *TEBIS- The Complete Solution for Tool, Die, Mold, and Pattern Manufacturing,* Tebis Technishe Informationssysteme, Germany (1994)
6. Wang, Y 'Intersection of offsets of parametric surfaces' *Computer Aided Geometric Design,* Vol 13 (1996), pp.453-465
7. Klass, R and Kuhn, B 'Fillet and surface intersections defined by rolling balls' *Computer Aided Geometric Design,* Vol 9 (1992), pp.185-193
8. Choi, B K, Park, J W and Chung, Y C 'Variable radius blending by ball position sampling' *Proc. of the 1st Pacific Conf. on Computer Graphics & Appl.,* World Scientific Pub Co, (1993), pp.221-234
9. Sheng, X and Hirsch, B E 'Triangulation of trimmed surfaces in parametric space' *Computer-Aided Design,* Vol 24 No 8 (1992), pp.437-444
10. Choi, B K *et al.* 'Triangulation of scattered data in 3D space' *Computer-Aided Design,* Vol 20 No 5 (1988), pp.239-247
11. Choi, B K *Surface Modeling for CAD/CAM,* Elsevier (1991)
12. Choi, B K and Jun, C S 'Ball-end cutter interference avoidance in NC machining of sculptured surfaces' *Computer-Aided Design,* Vol 21 No 6 (1989), pp.371-378
13. Takeuchi, Y *et al.* 'Development of a personal CAD/CAM system for mold manufacturing' *Annals of CIRP,* Vol 38 No 1 (1989), pp.429-432
14. Jerard, R B, Drysdale, R L and Hauck, K 'Geometric simulation of numerically controlled machining' *Proc. of ASME Int'l Conf. on Computers in Engineering,* ASME, New York (1988), pp.129-136
15. Chung, Y C and Choi, B K 'Non-parametric modeling of cutter swept surfaces for cutting simulation' *Trans of the Society of CAD/CAM Engineers,* Vol 1 No 1 (1996), Seoul, Korea, pp.45-55 (in Korean)
16. Faux, I D and Pratt, M J *Computational Geometry for Design and Manufacture,* Ellis Horwood (1980)
17. Renz, W 'Interactive smoothing of digitized point data' *Computer-Aided Design,* Vol 14 No 5 (1982), pp.267-269
18. Eck, M and Jaspert, R 'Automatic fairing of point sets' Sapidis, N S (Ed.) *Designing Fair Curves and Surfaces,* SIAM (1994), pp.45-60
19. Chou, J J and Piegl, L A 'Data reduction using cubic rational B-splines' *IEEE CG & A,* May 1992, pp.60-68
20. Fang, L and Gossard, D C 'Multidimensional curve fitting to unorganized data points by nonlinear minimization' *Computer-Aided Design,* Vol 27 No 1 (1995), pp.48-58