SCIENCE CHINA
Information Sciences

• **RESEARCH PAPERS** •

July 2011 Vol. 54 No. 7: 1407–1418
doi: 10.1007/s11432-011-4237-4

Curve fitting and optimal interpolation on CNC machines based on quadratic B-splines

ZHANG Mei[1], YAN Wei[2], YUAN ChunMing[1]*,
WANG DingKang[1] & GAO XiaoShan[1]

[1]*Key Laboratory of Mathematics Mechanization, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing* 100190*, China;*
[2]*Research Institute of Petroleum Exploration and Development, Beijing* 100083*, China*

Received December 30, 2009; accepted November 17, 2010

Abstract In this paper, curve fitting of 3-D points generated by G01 codes and interpolation based on quadratic B-splines are studied. Feature points of G01 codes are selected using an adaptive method. Next, quadratic B-splines are obtained as the fitting curve by interpolating these feature points. Computations required in implementing the velocity planning algorithm mentioned by Timer et al. are very complicated because of the appearance of high-order curves. An improved time-optimal method for the quadratic B-spline curves is presented to circumvent this issue. The algorithms are verified with simulations as well as on real CNC machines.

Keywords quadratic B-spline, G01 codes, feature point, velocity planning, interpolation

Citation Zhang M, Yan W, Yuan C M, et al. Curve fitting and optimal interpolation on CNC machines based on quadratic B-splines. Sci China Inf Sci, 2011, 54: 1407–1418, doi: 10.1007/s11432-011-4237-4

1 Introduction

G01 codes (micro line segments) have been used extensively in high-speed and high-accuracy CNC (computer numerical control) machining. The traveling path represented by G01 codes is computationally-intensive. It always requires large amounts of computer memory, and introduces many extreme changes in traveling directions. If we interpolate directly along the line segments generated by the G01 codes, machining becomes less efficient and workpiece surfaces may not be smooth enough. Lv et al. [1] used an arc transition interpolation method to smooth the cutting paths. However, the results still do not meet accuracy requirements, sometimes giving as much as twice the data volume compared to the original G01 codes. Zhang et al. [2] gave a multi-period turning method to interpolate the tool path and smooth the cutting paths at the same time. But their method also cannot compress the G01 codes. To solve this problem, we can use splines to approximate the traveling path, thereby providing a method of data compression. The number of line segments approximated by a single curve is called the compression ratio.

Spline interpolation is widely used to approximate the tool path generated by G01 codes. Yau et al. [3, 4] addressed a continuous-short-blocks criterion, that initially divides the G01 codes into groups,

*Corresponding author (email: cmyuan@mmrc.iss.ac.cn)

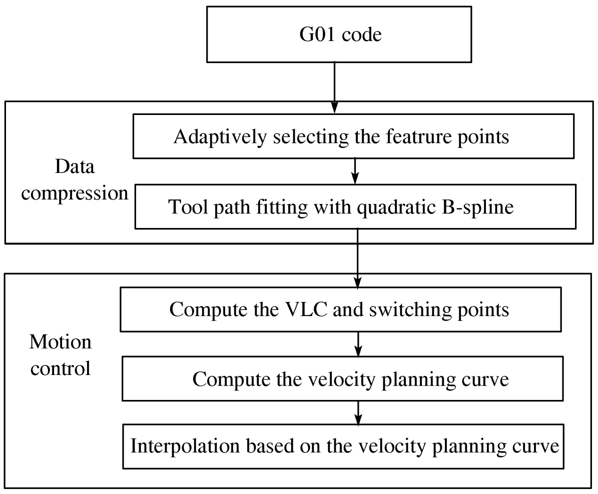

G01 code
Data compression
Adaptively selecting the featrure points
Tool path fitting with quadratic B-spline
Motion control
Compute the VLC and switching points
Compute the velocity planning curve
Interpolation based on the velocity planning curve

Figure 1 Flow chart for curve fitting and interpolation.

and then finds the spline that interpolates all points in a group. This procedure may introduce too many short curves when the points generated by G01 codes are very close to each other. A more practical means is to exploit only the feature points generated by the G01 codes, and then apply the spline interpolation to these points to approximate the tool path [5]. A new adaptive method for finding all feature points is proposed in this paper. Next, an interpolation algorithm using quadratic B-splines is implemented to construct the tool path from the feature points. Although refs. [6, 7] have given several methods of spline approximations, polynomial curves of degree three and higher or rational curves may generate many complicated computations when executing the velocity-planning algorithm. For this reason, quadratic B-splines are used for this construction so that the algorithm can be performed in real time.

In curved path machining, Ye et al. [8] used the constant-velocity transition method. Altintas et al. [9] addressed a new velocity function based on the acceleration and jerk-limit control. Yu et al. [10, 11] proposed a new S-shape acceleration and deceleration method based on filtering. Some other velocity-planning approaches have been proposed in [12–16], but all these methods used tangent acceleration and jerk-limit control. However, by not taking into account the acceleration capabilities of each motor axis, the maximal acceleration of each motor axis is not fully utilized. Therefore, a velocity planning method for multi-axis acceleration bounds is required [17, 18–21]. In [17], Timar et al. proposed a time-optimal interpolation method for general spline curves based on maximal acceleration capabilities of each motor axis. However, the method cannot be executed in real time for polynomial curves of degree higher than two.

To avoid these deficiencies, a new approximation and interpolation algorithm for G01 codes based on quadratic B-splines is proposed in this paper. Being both efficient and stable, this method can be widely used in CNC machining and can meet both the requirements of efficiency and precision in CNC systems.

2 Curve fitting and velocity planning for quadratic B-splines

Our approach consists two main parts: data compression and motion control. A processing flow chart is shown in Figure 1.

The G01 codes consisting of an ordered sequence of points in 3-D space is often generated by a CAD/CAM system and specifies linear motion of the machine. If interpolations are implemented on the line segments specified by two successive points in the G01 codes directly, they reduce the machining efficiency and cause mechanical vibrations. Therefore, to circumvent the problem, we use quadratic B-splines to construct initially an approximate tool path based on the points generated by the G01 codes, then implement the interpolation algorithm on the spline curves. For convenience, we call the points generated by G01 codes *G01 points*.

To accomplish data compression in the curve-fitting process, we use a single spline curve to approximate as many G01 points as possible with the given accuracy. There are three steps in the procedure:

1. Divide the points from the G01 codes into groups. Because there is always a large number of points, we cannot in general use a single B-spline to approximate all of these points simultaneously. Criteria to group the points that take into consideration both machine characteristics and geometric features of the tool path is presented in subsection 3.1.

2. Adaptively select all feature points from the G01 points with quadratic Bézier curve.

3. Finally, we use quadratic B-splines to interpolate all the feature points to obtain a smooth curved tool path.

After constructing the curved tool path, we begin the velocity planning and real-time interpolation, so that we can obtain the time-optimal interpolation algorithm. There are also three main steps to this process:

1. According to how the machine performs and the properties of the fitting tool path, the velocity limit curve (VLC) together with the switching points (definitions are ginven in later sections) are computed.

2. Given these velocity switching points, the control axis of each switching points and the VLC, we can compute the actual velocity curve.

3. Using this actual velocity curve and the interpolation tolerance, the interpolating points are computed one by one.

3 Fitting the G01 points with quadratic B-splines

In this section, we discuss in detail the method by which we group the G01 points, select the feature points and fit the tool path with quadratic B-splines.

3.1 Grouping the G01 codes

In most CNC machining, there is always a large amount of G01 codes for one part program, which can not be approximated by only one single B-spline curve. Hence, we have to partition these G codes into groups such that the points of each group can be fitted by a single B-spline curve.

To begin, we must check the distance between every pair of adjacent points against a threshold value which is set by the properties of the workpiece. If the distance is larger than this value, we keep the line segment between these two points, which are called *breaking points.*

In CNC machining, efficiency and accuracy are conditioned one by the other. To meet accuracy requirements, the velocity of the cutting tool has to decrease when it approaches corners. In particular, when the curvature of a corner is very high, i.e. indicating a sharp corner, we have to treat the corner point as a breaking point to maintain fitting errors below the tolerance of the system. This indicates that we must partition the G01 points based on the corresponding discrete curvature and then decide on the threshold value for curvature taking into account the mechanical properties of the machine. If a point's discrete curvature exceeds the threshold value, the corresponding point must be treated as a breaking point.

According to the method proposed in [22], the discrete curvature of the mid-point of three adjacent points can be computed. After obtaining the discrete curvatures of all G01-code points, the points with extreme curvatures are selected and classified as *initial feature points.*

Let v be the velocity of machining, κ the curvature, and r the radius of the curve, and a_N the centripetal acceleration. Because $r = \frac{v^2}{a_N} = \frac{1}{\kappa}$, the largest curvature of the tool path that ensures the tool moving smoothly is

$$\kappa_{\max} = \frac{a_{N\max}}{v^2}. \tag{1}$$

This $\kappa_{\max}$ could conceivably be the curvature threshold value. If there are *initial feature points* for which the associated curvature exceeds the threshold value, these would be classified as breaking points.

However, this threshold value does not always work well in all situations. When the machining speed is not constant, this procedure will not give a stable threshold value. Therefore, we use an estimated value that is given by the mechanical properties and the processing mode. When the discrete curvature of the *initial feature points* satisfies the following condition, these points would be classified as breaking points:

$$\kappa > \kappa_{\max} = \alpha \frac{a_{\max}}{F^2}, \tag{2}$$

where $a_{\max}$ is the combined acceleration of all axes, F is the given feedrate that is the maximal machining velocity specified by the system, and α is a rational coefficient that is obtained emirically. In this paper, we use $\alpha = 4$ or 9.

3.2 Adaptively selecting the feature points

Having performed the classification, every group of G01 points can now be approximated by just one B-spline, recalling that this increases the efficiency of the machining. In most cases, a group may still contain a large number of points. If going through every point [4, 5, 23], the fitting curve may comprise of substantially-many short polynomial curves and take on an unexpected shape [24]. To avoid such drawbacks, we proposed an adaptive method to select first only the feature points from the G01 points, and then use the quadratic B-spline to interpolate all of these feature points. There are two main steps constituting this adaptively selecting feature points algorithm:

Starting between two breaking points, compute the discrete curvature of every data points using the method described in subsection 3.1. Find the points exhibiting extreme curvature and classify them as *initial feature points*.

Next, use the quadratic Bézier curve to add the new feature points. Let the input of the initial feature points be $\{P_{i1}, \ldots, P_{ir}\}$, where P_{i1}, P_{ir} are two breaking points. The quadratic Bézier curve has the form:

$$B(u) = (1-u)^2 Q_0 + 2u(1-u)Q_1 + u^2 Q_2 \qquad (0 \leqslant u \leqslant 1),$$

where $u(\in [0,1])$ is the affine parameter and Q_0, Q_1, Q_2 are the control points that determine the curve.

Using the quadratic Bézier curve to interpolate every triplet of adjacent feature points $\{P_{ij-1}, P_{ij}, P_{ij+1}\}$, we have $Q_0 = P_{ij-1}, Q_2 = P_{ij+1}, u_0 = 0, u_2 = 1$. Using the cumulative chord length method to compute u_1, we get the linear equation for Q_1. We have then obtained the quadratic Bézier interpolation curve $B(u)$.

After obtaining the Bézier curve, we want to find the original positions of P_{ij-1} and P_{ij+1} in the G01 point sequence. We calculate the distances of all the data points between P_{ij-1} and P_{ij+1}, except for P_{ij}, to the Bézier curve. If all distances meet the accuracy requirement of the system, then no new feature point needs to be added. Otherwise, the point furthest away must be added to the feature-point sequence.

Repeat the procedure until $P_{ir} = Q_2$ and all distances computed from the corresponding points to this curve meet accuracy requirements given for the system. We then obtain a new sequence of feature points $\{P_{i1}, \ldots, P_{ir'}\}$.

As illustrated in Figure 2, Figure 2(a) is a subsequence of G01 points represented by $\circ$. Two end-points presented by $+$ are breaking points which satisfy condition (2). In Figure 2(b), the points designated by $\square$ are extreme-curvature points that become initial feature points; the curve in Figure 2(c) is the quadratic Bézier curve interpolating the three adjacent initial feature points (only one segment is shown for which a new feature point needs to be added), the point indicated by $\diamond$ is one for which the distance to the curve is larger than the error tolerance for the system. So this point will be added to the sequence of feature points. The tool path illustrated by the quadratic B-spline interpolating all the feature points is depicted in Figure 2(d). As we can see in the figure, the adaptive method based on selecting all the feature points has performed well in capturing the characteristics of the tool path.

Algorithm 1: Adaptively selection of the feature points

Input: A sequence of feature points between two breaking points $\{P_{i1}, \ldots, P_{ir}\}$, where P_{i1}, P_{ir} are the breaking points.

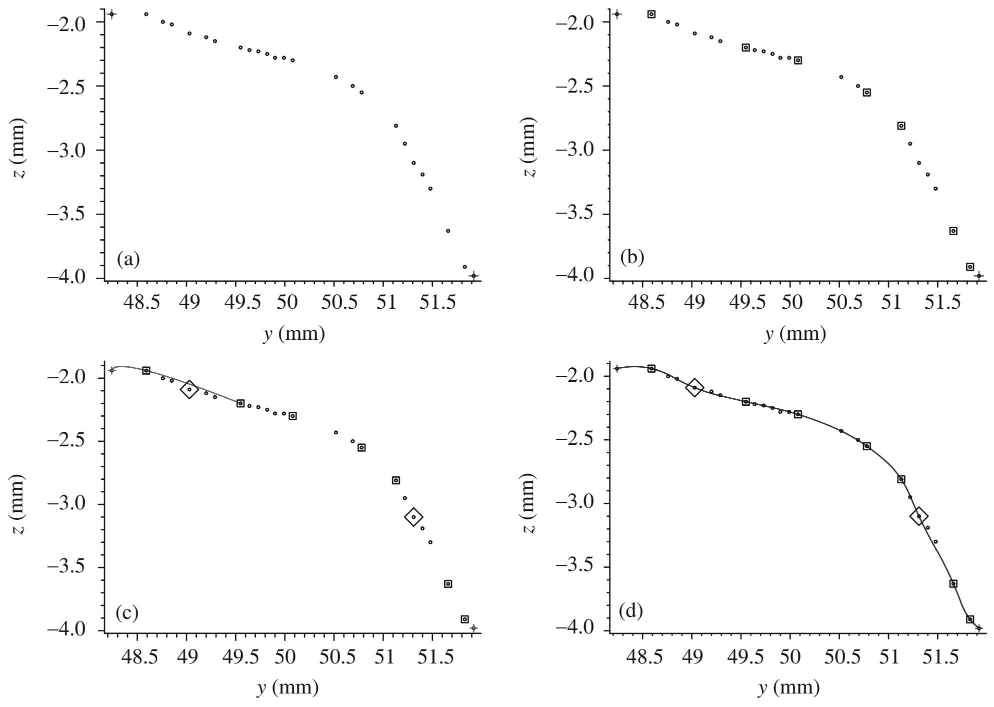


Figure 2 Adaptive quadratic B-spline fitting. (a) A section of G01 codes between two breaking points; (b) curvature extreme points; (c) adaptively adding new feature points; (d) quadratic B-spline fitting.

Output: A new sequence of feature points $NewP_i = \{P_{i1}, \ldots, P_{ir'}\}$.

Let $\{Q_1, Q_2, Q_3\}$ be the set of adjacent feature points which will be interpolated by a quadratic Bézier curve. Let $Q_1 = P_{i1}, Q_2 = P_{i2}, Q_3 = P_{i3}$.

Let $NewP_i$ be the new sequence of feature points, and the initial value is $NewP_i = \{P_{i1}, \ldots, P_{ir}\}$.

1) Use the Bézier curve to interpolate $\{Q_1, Q_2, Q_3\}$ so as to generate the curve $B(u)$.

2) Find the original positions of $\{Q_1, Q_2, Q_3\}$ in the G01 point sequence, and then compute the distances from all points between Q_1 and Q_3 to $B(u)$ except for Q_2.

3) If the maximal distance is larger than the error tolerance, then the corresponding point P_k is selected. If its position in the G01-code sequence is before Q_2, then let $Q_3 = Q_2,\ Q_2 = P_k$; otherwise, $Q_3 = P_k$; return to 1

4) If Q_2, Q_3 are not in $NewP_i$, then add them into the feature points sequence $NewP_i$ one by one.

5) If $Q_3 \neq P_{ir}$, let the next adjacent point of Q_3 in $NewP_i$ be P_{next}, then let $Q_1 = Q_2,\ Q_2 = Q_3,\ Q_3 = P_{\text{next}}$, return to 1.

3.3 Tool path fitting with the quadratic B-splines

The quadratic B-spline can be expressed as

$$C(u) = \sum_{i=0}^{n} Q_i N_2^i(u), \quad 0 \leqslant u \leqslant 1, \tag{3}$$

where Q_i $(i = 0, 1, \ldots, n)$ are the $n+1$ control points, $N_2^i(u)$ $(i = 0, 1, \ldots, n)$ are the basis functions defined on the knot vector $\boldsymbol{T} = \{t_0, t_1, \ldots, t_{n+3}\}$, which are the subintervals for the spline curve and u is the affine parameter.

According to the spline interpolation, a system of linear equations in the unknown control points can be constructed using the corresponding feature points. Because of the properties of quadratic B-splines, the coefficient matrix is triangular that can be solved by using the chasing method.

Thus by employing the quadratic B-spline to implement the curve-fitting procedure on the G01 point sequence, a calculation involving triangular coefficient matrix has a higher computing efficiency than

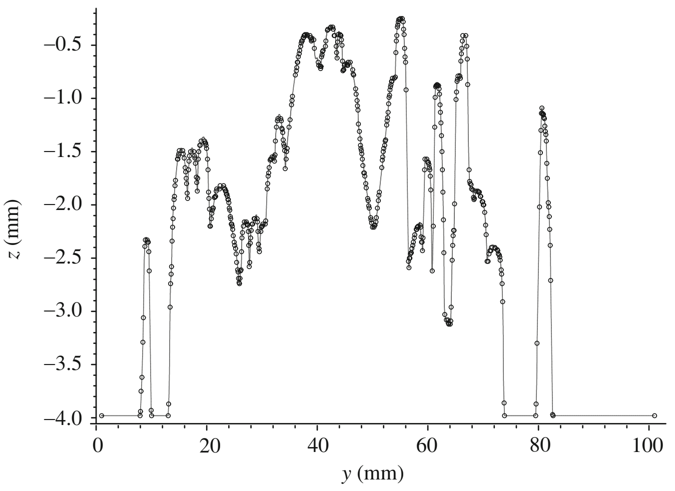


Figure 3 Fitting result.

a general system solved by Gaussian elimination [5, 25]. Moreover, we can significantly simplify the calculation involved in the velocity planning procedure.

3.4 Algorithm

Finally, we present the adaptive quadratic B-spline fitting algorithm for the G01 point sequence.

Algorithm 2: Adaptive quadratic B-spline fitting algorithm

Input: G01 point sequence $\{P_0, \ldots, P_n\}$.

Output: The set of quadratic B-splines $\{C_1(u), \ldots, C_m(u)\}$ as the fitting curve.

1. Compute the curvature threshold value $\kappa_{\max}$ as in (2).

2. Compute the discrete curvature κ_i $(i = 0, \ldots, n)$ with respect to every point in the G01 point sequence. Take the extreme-curvature points as the initial feature points, and classify as breaking points for which the curvature is larger than $\kappa_{\max}$.

3. For the feature point sequence $\{P_{i1}, \ldots, P_{ir}\}$, that lie between two breaking points, use the adaptive selecting method given in subsection 3.2 to generate the new feature point sequence $\{P_{i1}, \ldots, P_{ir'}\}$.

4. Use the quadratic B-spline to interpolate all the selected feature points to construct the curved tool path $C_i(u)$ as described in subsection 3.3.

Figure 3 is the curve-fitting result of a tool path along the "vase" workpiece in Figure 9. It contains 486 data points, and has been approximated by 34 B-splines including 173 quadratic polynomials. To show details more clearly, we have expanded the y-coordinate. So the compression ratios are 14.3 or 2.8 according to either for B-spines or quadratic polynomials respectively.

4 Time-optimal machining algorithm based on quadratic B-splines

In the following, we consider 3D spline curves. Before the interpolation step, we need to know the velocities for all parameters. The procedure that solves this problem is called velocity planning for which we need to analyze local properties of the curve. Let "$'$ " denote the derivative with respect to u, and "$\dot{}$ " the derivative with respect to time t.

We assume that the parametric curve $C(u)$ is expressed in the form given in eq. (3). Let the *parameter speed* of the curve be $\sigma(u) = \frac{ds}{du} = |C'(u)|$. Because $v(u) = \frac{ds(u)}{dt}$, $\frac{d}{dt} = \frac{ds}{dt}\frac{du}{ds}\frac{d}{du} = \frac{v}{\sigma}\frac{d}{du}$. Our goal is to compute the velocity curve $v(u) = \frac{ds}{dt}$ that minimizes the traversal time:

$$\min_{v(u)} T = \int_0^T dt = \int_0^1 \frac{\sigma}{v} du, \tag{4}$$

subject to

$$|a_x(u)| \leqslant A_x, \ \ |a_y(u)| \leqslant A_y, \ \ |a_z(u)| \leqslant A_z, \ \ \forall u \in [0, 1], \tag{5}$$

where (a_x, a_y, a_z) denotes the acceleration vector in Cartesian coordinates, and (A_x, A_y, A_z) denotes the maximal acceleration allowed.

4.1 Improved velocity planning method for quadratic B-splines

To solve the optimization problem (4), Timar et al. [17] proposed an algorithm based on a "bang-bang" control strategy, that is, there is always one axis that attains the acceleration bound. However, this method involves complicated algebraic equations for high order curves. In this section, we propose an essential improvement on the algorithm mentioned in [1] for the quadratic B-spline that substantially reduces the computational requirements.

If we use the "bang-bang" control strategy, we can assume, without loss of generality, that x is the control axis and therefore $a_x = \pm A_x$. Then $\frac{q'}{2\sigma^2}x' + \frac{q}{\sigma^3}(\sigma x'' - \sigma' x') = \pm A_x$. Solving this differential equation, we have

$$q = \left(\frac{\sigma}{x'}\right)^2 (n \pm 2A_x x), \tag{6}$$

where n is a constant that can be computed from a specified point: $n = (\frac{x'(u_*)}{\sigma(u_*)})^2 q(u_*) \mp 2A_x x(u_*)$. The above curve $q(u)$ is called the *integration curve* or *integral velocity curve.*

Therefore, employing the "bang-bang" control strategy, we need only to find the starting and ending points, the feedrate at the starting points and the corresponding control axis of the integration curve. To obtain the starting points of the velocity curve, velocity limit curve (VLC) is introduced in [1], that is, the curve consisting of the maximal speed limits with respect to every parameter u.

According to the method mentioned in [1], the solution of the VLC has the following form:

$$v^2_{\mathrm{lim},P,R} = \frac{\sigma^2(\alpha_R A_R P' - \alpha_P A_P R')}{P'R'' - P''R'}, \tag{7}$$

where $v_{\mathrm{lim},P,R}$ denotes the VLC determined by the P and R axes, $\alpha_P = \pm 1$, $\alpha_R = \pm 1$, and $P, R \in \{x, y, z\}$ are two different axes.

To simplify the expression, we use P, R, M, N, K or H to represent one of the x, y and z control axes. For example, if $\alpha_P = 1$, the corresponding P axis means $+P$ axis; if $\alpha_P = -1$, the corresponding P axis represents the $-P$ axis; and A_P represents the maximal accelerating value of P axis. Now, let the quadratic spline curve for the P axis have the form: $P(u) = a_P\, u^2 + b_P\, u + c_P$, where a_P, b_P and c_P are the coefficients of second, first and constant terms, respectively.

Suppose the control axis is $+P$, then we want to find the first intersection of eqs. (6) and (7). This means we have to solve a higher-order algebraic equation. Generally, the equation has to be solved numerically. Taking into consideration of robustness and efficiency of the calculation, meeting real-time machining requirements is hard to achieve. Thus, for the sake of practicality, we need to avoid such computation of intersections in the velocity planning curve and the VLC calculations.

Because P and R could be any two of the x, y or z axes, the final VLC is the smallest one of all the VLCs computed from any two axes. Therefore,

$$v^2_{\mathrm{lim}} = \min_{P,R \in \{x,y,z\}} (v^2_{\mathrm{lim},P,R}). \tag{8}$$

To do this, we need to explain how to compute the three types of switching points [1] for a quadratic B-spline.

(i) Tangency points: Because the highest order of the curved path is quadratic, we do not need to compute the tangent points in our method of computation.

(ii) Discrete points and slope discontinuities of the VLC: For a parameter u, if the left and right limitation of v_{lim} is different, then we can choose the point with smaller v_{lim} value to be the discrete point. First, we need to derive the parameter values corresponding to the connection points (breaking points). These connection points could be the discrete points and slope discontinuities.

When P and R are the control axes, the points where the control axes are changed are possible slope discontinuities. We can solve the following linear equation to obtain the parameters: $\alpha_1 A_R P' - \alpha_2 A_P R' = \alpha_1' A_R P' - \alpha_2' A_P R'$, where α_i, α_i' equals ± 1, for $i = 1, 2$.

Another type of slope discontinuity of a VLC can be computed by the intersection points of the VLCs that are determined by different control axes (P, R) and $(M, N)(P, R, M, N \in \{x, y, z\})$. The parameter of the intersection point has the following explicit expression:

$$u = \frac{(\alpha_N A_N b_M - \alpha_M A_M b_N)(2a_R b_P - 2a_P b_R) - (\alpha_R A_R b_P - \alpha_P A_P b_R)(2a_N b_M - 2a_M b_N)}{(2\alpha_R A_R a_P - 2\alpha_P A_P a_R)(2a_N b_M - 2a_M b_N) - (2\alpha_N A_N a_M - 2\alpha_M A_M a_N)(2a_R b_P - 2a_P b_R)}. \tag{9}$$

All of the above parameters could be discrete points or slope discontinuities.

We can obtain the final VLC according to eq. (8), then check the above parameters to determine the final discrete points and slope discontinuities.

So, when we employ quadratic B-splines to solve all these three types of switching points, we can omit the computation of the tangent points and have explicit formulae for both discrete points and slope discontinuities.

When we compute the integration curve, we have to find the transition points of the control axes where the control axis has changed. If $+P$ is again the control axis, according to eq. (6), we can obtain the velocity planning curve $v^2 = (\frac{\sigma}{P'})^2(n_P + 2A_P P)$. Solving the acceleration equation of R axis gives: $\frac{q'}{2\sigma^2}R' + \frac{q}{\sigma^3}(\sigma R'' - \sigma' R') = \pm A_R$. As a consequence, the parameter of the transition point is a root of a cubic equation, whose unique real root is

$$u = \frac{\sqrt[3]{(a_R b_P - a_P b_R)(4a_P A_P c_x + 2a_P n_P - A_P b_P^2)(a_R A_P - A_R a_P)^2} + b_P a_R A_P - b_P A_R a_P}{2(-a_R A_P + A_R a_P)a_P}.$$

Thus, because of the quadratic B-spline, the parameter of the transition point has an explicit expression. Moreover, we are able to propose an improved velocity planning algorithm.

Algorithm 3: Optimal velocity planning algorithm for quadratic B-spline

Input: Piecewise quadratic B-spline $C(u) = (x(u), y(u), z(u))(0 \leqslant u \leqslant 1)$ with C^1 continuity at the connecting points.

Output: Piecewise velocity-planning curve $v_{sd}(u)(0 \leqslant u \leqslant 1)$.

1. Compute the VLC, the discrete points and the slope discontinuities of the VLC.

2. Starting from $u = 0$ and initial velocity $v(0) = v_0$, compute the control axis H of this point in the $+u$ direction. Use H to compute the current velocity curve. As u increases, compute the acceleration of any non-control axis until it reaches the given bound, that is $\frac{vv'}{\sigma^2}H' + \frac{v^2}{\sigma^3}(\sigma H'' - \sigma' H') = \alpha_K A_K$, where K is any axis except for H. Compute the smallest value of parameter u of the axis transition. Check whether there exists a switching point such that at least one acceleration bound of the point is not satisfied. If this is the case, u is set to the smallest parametric value of these switching points. Recompute the control axis and the new velocity curve according to this parametric value u, the control axis and current velocity curve; repeat the above procedure until $u = 1$. Finally, we can determine the entire velocity curve v_F.

Starting from $u = 1$ and initial velocity $v(1) = v_e$, compute the control axis H in the $-u$ direction. According to H, compute the current velocity curve. As u decreases, if the acceleration of any non-control axis reaches the given bound, compute the axis transition parameter u, recompute the control axis and the new velocity curve until $u = 0$. Similar to the computation of v_F, we finally determine the velocity-planning curve v_B.

3. Compute the intersection point of the forward and backward velocity planning curve v_F and v_B: $(u_{in}, v(u_{in}))$, where $v(u_{in}) = v_F(u_{in}) = v_B(u_{in})$. The velocity planning curve is then

$$v_{sd}(u) = \begin{cases} v_F, 0 \leqslant u < u_{in}, \\ v_B, u_{in} \leqslant u \leqslant 1. \end{cases}$$

4. Check whether all switching points of the VLC are under the current velocity planning curve v_{sd}; if so, then we have obtained the final velocity planning curve v_{sd}, and return v_{sd}; otherwise, continue the following steps.

5. Select the switching point $(u_{sp}, v_{\lim}(u_{sp}))$ corresponding to the smallest value of the parameter u for the current velocity planning curve v_{sd}. Then compute the control axis in the $+u$ and $-u$ directions. According to this control axis, we can compute two new integration curves v_{sp-} and v_{sp+}, from point $(u_{sp}, v_{\lim}(u_{sp}))$ as u decreases and increases, respectively. Then compute $(u_{sp-}, v_{sp-}(u_{sp-}))$ and $(u_{sp+}, v_{sp+}(u_{sp+}))$ which are the intersection points of v_{sp-}, v_{sp+} and the current velocity curve v_{sd}. Update the whole velocity planning curve v_{sd} as follows:

$$v_{sd}(u) = \begin{cases} v_{sd}, 0 \leqslant u < u_{sp-}, \\ v_{sp-}, u_{sp-} \leqslant u < u_{in}, \\ v_{sp+}, u_{in} \leqslant u < u_{sp+}, \\ v_{sd}, u_{sp+} \leqslant u \leqslant 1, \end{cases}$$

return to step 4.

Remark 4.1. Comparing the above algorithm with that in [1], we have successfully avoided computing the intersections of the velocity curve and the VLC. However, we have to compute a new velocity planning curve when the current integration curve exceeds the VLC. In such case, we use a very-high-valued velocity curve in our algorithm instead of the exceeding part (this part can be checked by comparing the current velocity and a given upper bound, and if the current velocity exceeds the upper bound, we define the velocity of the exceeding parts as this value). This is because the final velocity-planning curve is below the VLC and the exceeding part will be replaced in further computations. This method leads to the determination of the complete velocity-planning curve and simplifies the computation. The final actual-velocity curve is composed of the minimal parts of the above velocity curves.

4.2 The interpolation method based on quadratic B-splines

For the given parameter u_i, compute the velocity according to the velocity curve v_{sd}, that is, let $v(u_i) = \min(v_{sd}(u_i), v_{lim}(u_i))$, and then compute the step size $\Delta L = v(u_i) \cdot T$ (T is the sampling period of the machine). Use ΔL and the first order approximation to obtain the initial value of the interpolation point. Next, the Newton-Raphson iteration is employed to approximate the interpolation parameter u_{i+1} [26].

4.3 Computational simulations

The fitting curve chosen for illustration is one of the tool paths in the “vase” workpiece in Figure 8. As seen in Figure 4, this curve is composed of 6 piecewise quadratic B-spline, and has C^1 continuity at the connecting point. The G01 point sequence is constructed by the cutting plane method, and as a consequence, there is only y- or z-axis control of the machining. The acceleration bounds of the control axes are $A_y = 3000$ mm/s^2 and $A_z = 1000$ mm/s^2. After the computation, we obtain the VLC and the velocity planning curve as displayed in Figure 5. The actual computed accelerations along the y- and z-axes are shown in Figures 6 and 7, respectively. From this, it can be seen that the actual machining is controlled basically under the optimal “bang-bang” control strategy.

5 Experimental results

To test the algorithms presented above, three workpieces for actual machining are chosen: i.e., “vase”, “Beijing Olympic Game”, and “head sculpture”, which are depicted in Figure 8. In Figure 9, we give the picture of the NC machine used in our experiments and two real “vase” manufactured in the NC machine with our method. The “Beijing Olympic Games” workpiece is a simple model compared with the others, but the “vase” is very complicated, which is typical of bi-axis machining. The “head sculpture” is highly representative of three axis machining. All simulations were performed on a PC with 2.13 GHz Intel Core 2 processor using the VC++6.0 environment.

Employing the algorithm addressed in section 2, we computed the spline compression ratio (the number of line segments approximated by one spline curve) for each workpiece; these were 27.3, 156.3, and

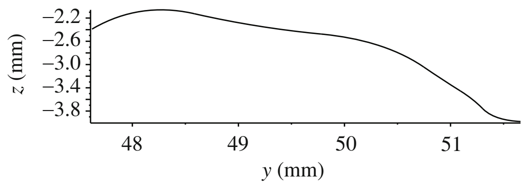


Figure 4 A fitting curve.

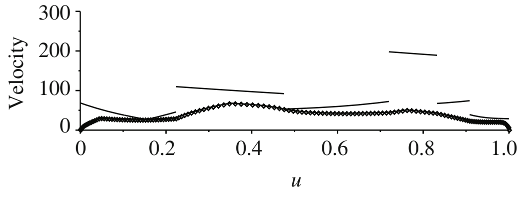


Figure 5 VLC, velocity planning curve, and velocities at the interpolation points. The unit for the velocities are mm/s.

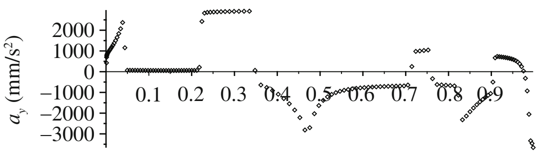


Figure 6 Computed acceleration chart of y-axis. The unit for the acceleration is mm/s^2.

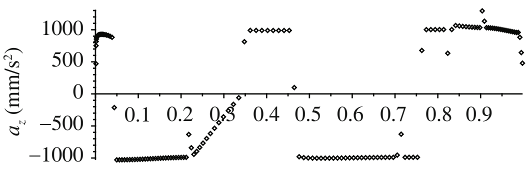
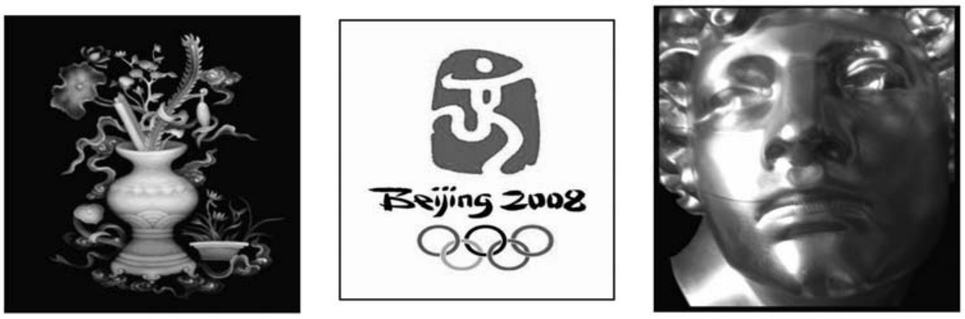
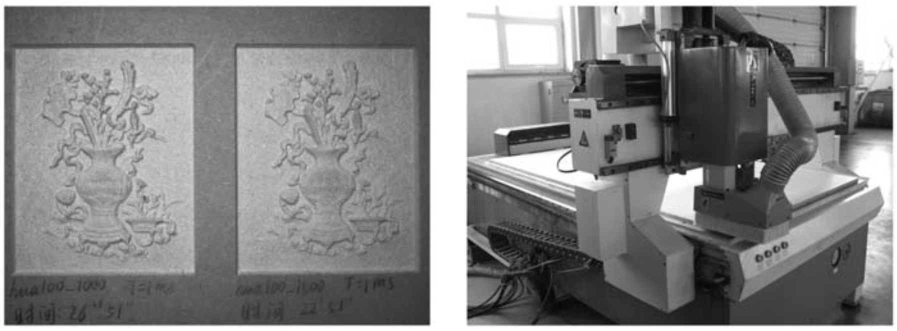


Figure 7 Computed acceleration chart of z-axis. The unit for the acceleration is mm/s^2.



"Vase" workpiece "Beijing Olympic Games" workpiece "Head-sculpture" workpiece

Figure 8 Workpieces.



"Vase"-NC machining picture Machine picture

Figure 9 Machine picture.

74 for "vase", "Beijing Olympic Games" and "head sculpture" workpiece, respectively. The polynomial compression ratio (the number of line segments approximated by one polynomial) of these three were 3.6, 15.5, and 8.1 respectively. We next used Algorithm 3 proposed in section 4 to perform velocity planning and interpolation. Comparing the results with the const velocity transition method [8], our computing

efficiency improved by 65%–130%. Compared the results with the traditional arc transition interpolation method, our computing efficiency improved by 30%–110%.

In regard to the computation times, the time spent on constructing the tool path can be neglected compared with that associated with velocity planning and interpolation. The computation time in performing the improved velocity planning method and interpolation method are almost the same, but are much less than the machining time. This means that these computations can be conducted in real time.

From the simulations, we can see that the algorithms we have proposed have worked well for both simple ("Beijing Olympic Games") and complicated ("vase" and "head sculpture") examples.

The machining examples show that our algorithm is practical and robust.

6 Closing remarks

The optimal velocity-planning and interpolation algorithms proposed in this paper are based on the quadratic B-splines. We used an adaptive method to select all feature points from the G01 point sequence, and then used quadratic B-splines to interpolate between all these feature points. From the point of view of simulations, this data-compression method is very efficient, accurate, and stable, and also can be widely used in CNC machining.

For quadratic B-splines, we proposed an efficient algorithm to compute actual-velocity curves within the "bang-bang" control strategy. In the computing procedure, we have made full use of the properties of quadratic B-splines. Our improved method mainly has resolved the following problems:

1. The computation of intersections of velocity curves and VLC is circumvented, thereby eliminating a large amount of computation that would have been necessary in solving the high-order algebraic equations.

2. Employing quadratic B-splines has reduced a number of computing steps and avoided the occurrence of the high-order equation. Essentially, we can obtain all solutions explicitly. Thus circumventing complicated computations arising from employing numerical methods.

However, our method is based on considering a complete curve over the workpiece; the velocity at its two end-points must be specified. If we want to increase the machining velocity, we have to introduce global methods to plan the velocities at the end-points. Such global methods will involve a lot of backtracking computation which may lower the efficiency. With this in mind, we will be focusing on local methods in future research, particularly in regard to velocity planning at the end-points, so as to maintain global optimal control and to achieve real time interpolation within the accuracy required.

Acknowledgements

This work was supported by the Knowledge Innovation Project of Chinese Academy of Sciences (Grant No. KGCX2-YW-119-1), and the National Natural Science Foundation of China (Grant No. 608210021/F02). The authors would like to thank Shenyang Institute of Computing Technology, Chinese Academy of Sciences for their kind help to implement our algorithm in their machine and to manufacture "Vase" workpiece.

References

1 Lv Q, Zhang H, Yang K M, et al. Study on the method of increasing turning velocity during CNC continuous machining (in Chinese). Manuf Tech Mach Tool, 2008, 3: 79–83

2 Zhang L X, Sun R Y, Gao X S, et al. An optimal solution for high-speed interpolation of consecutive micro-line segments and adaptive real-time lookahead scheme in CNC machining (in Chinese). MM-Preprints, 2010, 29: 206–227

3 Yau H T, Wang J B. Fast Bezier interpolator with real-time lookahead function for high-accuracy machining. Int J MachTools Manuf, 2007, 47: 1518–1529

4 Wang J B, Yau H T. Real-time NURBS interpolator: application to short linear segments. Int J Adv Manuf Tech, 2009, 41: 1169–1185

5 Zhao X, Yin Y, Yang B. Dominant point detecting based non-uniform B-spline approximation for grain contour. Sci China Ser E-Tech Sci, 2007, 50: 90–96

6 Farin G, Hoschek J. Handbook of Computer Aided Geometric Design. Elsevier Science, 2002

7 Wang R H. Numerical Rational Approximation (in Chinese). Shanghai: Shanghai Science and Technology Press, 1980
8 Ye P Q, Zhao S L. Study on control algorithm for micro-line continuous interpolation (in Chinese). China Mech Eng, 2004, 15: 1354–1356
9 Erkorkmaz K, Altintas Y. High speed CNC system design. Part I: Jerk limited trajectory generation and quintic spline interpolation. Int J Mach Tools Manuf, 2001, 41: 1323–1345
10 Yu D, Hu S H, Gai R L, et al. Research on acceleration and deceleration for CNC machine tools based on filtering (in Chinese). China Mech Eng, 2008, 19: 804–807
11 Gai R L, Lin H, Zheng L M, et al. Design and implementation of velocity planning algorithm for high speed machining (in Chinese). J Chin Comput Syst, 2009, 30: 1067–1071
12 Shi C, Zhao T, Ye P Q, et al. Study on S-shape curve acceleration and deceleration control on NC system (in Chinese). China Mech Eng, 2007, 18: 1421–1425
13 Du D S, Yan C L, Li C X. An adaptive NURBS interpolator with real-time look-ahead function (in Chinese). J Shanghai Jiaotong Univ, 2006, 40: 843–847
14 Lin M T, Tsai M S, Yau H T. Development of a dynamics-based NURBS interpolator with real-time look-ahead algorithm. Int J Mach Tools Manuf, 2007, 47: 2246–2262
15 Liu X S, Jia Q X, Yuan X H. A study of NURBS interpolation algorithm and ACC-DEC control (in Chinese). Modul Mach Tool Automat Manuf Tech, 2007, 2: 60–64
16 Yuan C M, Gao X S. Time-optimal interpolation of CNC machines along parametric path with chord error and tangential acceleration bounds. MM-Preprints, 2010, 29: 165–188
17 Sebastian D T, Farouki R T, Smith T S, et al. Algorithms for time-optimal control of CNC machines along curved tool paths. Robot Com-Int Manuf, 2005, 21: 37–53
18 Zhang K, Gao X S, Li H B, et al. A Greedy Algorithm for feed-rate planning of CNC machines along curved tool paths with confined jerk for each axis. MM-Preprints, 2010, 29: 189–205
19 Bobrow J E, Dubowsky S, Gibson J S. Time-optimal control of robotic manipulators along specified paths. Int J Robot Automat, 1985, 4: 3–17
20 Shiller Z, Lu H H. Robust computation of path constrained time optimal motions. In: Proceedings, IEEE International Conference on Robotics and Automation, Cincinnati, OH, 1990. 144–149
21 Timar S D, Farouki R T. Time-optimal traversal of curved paths by Cartesian CNC machines under both constant and speed-dependent axis acceleration bounds. Robot Com-Int Manuf, 2007, 23: 563–579
22 Coeurjolly D, Svensson S. Estimation of curvature along curves with application to fibres in 3D images of paper. Lect Notes Comput Sci, 2003, 2749: 247–254
23 Lin H, Wang G, Dong C. Constructing iterative non-uniform B-spline curve and surface to fit data points. Sci China Ser F-Inf Sci, 2004, 47: 315–331
24 Piegl L, Tiller W. The NURBS Books. Berlin: Springer, 1997
25 Kang S J, Zhou L S, Zhang D L. Fitting cutter location points with B-spline curves accurately (in Chinese). Mech Eng Automat, 2007, 26: 95–97
26 Shi R M. Numerical Computation (in Chinese). Beijing: Advanced Education Press, 2004