SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

March 2016, Vol. 59 032103:1–032103:9
doi: 10.1007/s11432-015-5322-x

A B-spline curve extension algorithm

Yang LU[1,2,3,4], Kanle SHI[1,3,4], Junhai YONG[1,3,4,5], Hejin GU[5]* & Haichuan SONG[1,2,3,4]

[1]*School of Software, Tsinghua University, Beijing* 100084*, China;*
[2]*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084*, China;*
[3]*Key Laboratory for Information System Security, Ministry of Education of China, Beijing* 100084*, China;*
[4]*Tsinghua National Laboratory for Information Science and Technology, Beijing* 100084*, China;*
[5]*Jiangxi Academy of Science, Nanchang* 330096*, China;*

Received October 17, 2014; accepted January 12, 2015; published online November 23, 2015

Abstract B-spline curve extension is an important operation in computer aided design systems. In this paper, we present a new extension algorithm for B-spline curves. The algorithm uses curve unclamping to generate a uniform B-spline curve segment from the original curve and gradually extends the segment to pass through every target point. Algorithms of uniform B-spline curves are used such that our algorithm has a low time cost and can easily handle arbitrary-order derivative constraints at the target points. Generalization for non-uniform rational B-spline curve extension is also discussed, and examples show the efficiency of our method.

Keywords curve extension, B-spline/NURBS, unclamping, clamping, uniform

Citation Lu Y, Shi K L, Yong J H, et al. A B-spline curve extension algorithm. Sci China Inf Sci, 2016, 59(3): 032103, doi: 10.1007/s11432-015-5322-x

1 Introduction

B-spline curve extension is a basic operation in computer aided design systems [1]. Given an original curve $\boldsymbol{P}(t)$ and several target points $\{\boldsymbol{R}_i\}$, curve extension methods aim to find a resulting curve $\boldsymbol{Q}(t)$, which contains all the curve points of $\boldsymbol{P}(t)$ and also passes through every target point $\boldsymbol{R}_i$.

Curve extension is useful in many practical applications, such as curve blending, curve interpolation, and curve approximation. For instance, an original curve can be extended to the end point of another curve so as to implement the operation of curve blending [2]. Since the resulting curve passes through every target point, curve extension can also be treated as a solution for curve interpolation [3]. In curve approximation, curve extension can be used to extend the initial curve to the selected key points step by step so as to gradually improve the fitting accuracy [4].

The widely used algorithm for B-spline curve extension was proposed by Hu et al. [1]. They used curve unclamping to extend the original curve. One new control point was added for each target point and evaluated through the inverse process of the de Boor algorithm. Assuming that the degree of the original B-spline curve is p, the extension curve would have C^{p-1} continuity with the original curve at the joint point. Based on the work of Hu et al. [1], the following two papers mainly focused on handling the

* Corresponding author (email: guhejin@vip.163.com)

continuity constraints at the target points. Chen et al. [4] used the method of undetermined coefficients to control the tangent direction of each target point. Liu et al. [2] added three new control points to the extension curve for each target point so as to fit G^2 constraints at the target points. Other extension methods were also studied. Shetty and White [5] and Zhou et al. [6] directly generated a G^2 curve segment between the original curve and the target point for curve extension. Fan et al. [7], Mo et al. [8], and Xu et al. [9] used cubic B-spline curves for extension and focused on optimal shape control. Xu [10] extended the original curve to only part of the target points and used the remaining points for curve approximation. Zhang et al. [11, 12] focused on the extension problem for disk/ball B-spline curves.

There is still room to improve the method of Hu et al. [1]. Their method has a relatively high time cost, and derivative constraints cannot be directly specified for each target point. In this paper, we propose a new extension method, which has a low time complexity and can handle arbitrary-order derivative constraints for each target point. In our method, operations of curve clamping and unclamping are used, and algorithms of uniform B-spline curves are applied. We first present our method for B-spline curves, and then generalize it to handle non-uniform rational B-spline (NURBS) curves.

The rest of the paper is organized as follows. Section 2 introduces the concept of clamped/unclamped and uniform B-spline curves. Section 3 presents our extension algorithm for B-spline curves, and Section 4 generalizes the method for NURBS curve extension. Section 5 analyzes the time cost and presents several examples. Section 6 concludes the paper.

2 Preliminaries

2.1 Clamped and unclamped B-spline curves

A pth-degree B-spline curve is defined as

$$\boldsymbol{P}(t) = \sum_{i=0}^{n} N_{i,p}(t)\boldsymbol{P}_i, \quad t_p \leqslant t \leqslant t_{n+1}, \tag{1}$$

where $\{\boldsymbol{P}_i \mid i = 0, 1, \dots, n\}$ are the control points and $\{N_{i,p}(t)\}$ are the pth-degree B-spline basis functions defined on knot vector $T = \{t_0, t_1, \dots, t_{n+p+1}\}$ [13].

A B-spline curve is called clamped if its knot vector T satisfies $t_0 = t_1 = \cdots = t_p$ and $t_{n+1} = t_{n+2} = \cdots = t_{n+p+1}$. Otherwise, it is called unclamped. In most cases, we use only clamped B-spline curves, but unclamped B-spline curves are still useful.

Clamped and unclamped B-spline curves can be converted to each other through the operations of knot insertion and removal [13].

2.2 Uniform B-spline curves

We call an unclamped B-spline curve $\boldsymbol{P}(t)$ (as defined in (1))uniform if its knot vector is evenly distributed, which means $t_1 - t_0 = \cdots = t_{n+p+1} - t_{n+p} = \Delta$.

Evaluation of a uniform B-spline curve is simple, since its basis functions are all translates of one another. If $\boldsymbol{P}(t)$ is uniform, we can compute its curve point through the formula:

$$\boldsymbol{P}(t) = [u^p, u^{p-1}, \dots, 1]\ M_p\ [\boldsymbol{Q}_{j-p}, \dots, \boldsymbol{Q}_j]^{\mathrm{T}}, \tag{2}$$

where $t \in [t_j, t_{j+1}]$, $u = (t - t_j)/(t_{j+1} - t_j)$, and M_p is a $(p+1) \times (p+1)$ constant matrix (see [14]).

M_p is usually computed and stored in advance. In the following section, we present a method to calculate its value.

2.3 Evaluation of M_p for uniform B-spline curves

The following method can be used to compute the matrix M_p for arbitrary values of p.

Consider a uniform B-spline curve $\boldsymbol{Q}(t)$ defined on $t \in [0, 1]$ with degree p, control points $\{\boldsymbol{Q}_i \mid 0 \leqslant i \leqslant p\}$, and knot vector $\{-p, -p+1, \dots, p+1\}$. First, we clamp the curve to knot vector $\{0, 0, \dots, 0, 1, \dots, 1, 1\}$

to make it a Bézier curve [13]. Assume $\{\boldsymbol{P}_i\}$ are the control points after clamping. Then, we can compute the curve points through the formula:

$$\boldsymbol{Q}(t) = [B_{0,p}(t), B_{1,p}(t), \dots, B_{p,p}(t)][\boldsymbol{P}_0, \boldsymbol{P}_1, \dots, \boldsymbol{P}_p]^{\mathrm{T}},$$

where $\{B_{i,p}(t) = C_n^i t^i (1-t)^{n-i} \mid 0 \leqslant i \leqslant p\}$ are the Bernstein basis functions. Denote W_1 as a transform matrix from the power basis to the Bernstein basis, and W_2 as a transform matrix from control points $\{\boldsymbol{Q}_i\}$ to control points $\{\boldsymbol{P}_i\}$.

$$[B_{0,p}(t), B_{1,p}(t), \dots, B_{p,p}(t)] = [t^p, t^{p-1}, \dots, 1]\ W_1,$$

$$[\boldsymbol{P}_0, \boldsymbol{P}_1, \dots, \boldsymbol{P}_p]^{\mathrm{T}} = W_2\ [\boldsymbol{Q}_0, \boldsymbol{Q}_1, \dots, \boldsymbol{Q}_p]^{\mathrm{T}}.$$

Then, we have

$$\boldsymbol{Q}(t) = [t^p, t^{p-1}, \dots, 1]\ W_1 W_2\ [\boldsymbol{Q}_0, \boldsymbol{Q}_1, \dots, \boldsymbol{Q}_p]^{\mathrm{T}}.$$

It follows that $M_p = W_1 W_2$.

W_1 can be computed through the following formula:

$$W_1^{i,j} = \begin{cases} C_p^j \times C_{p-j}^{p-i-j} \times (-1)^{p-i-j}, & \text{if } i+j \leqslant p, \\ 0, & \text{otherwise,} \end{cases}$$

where $W_1^{i,j}$ $(0 \leqslant i,\ j \leqslant p)$ is the element of W_1 at the ith row and jth column.

W_2 can be computed through curve clamping. Assign control points of the uniform curve $\{\overline{\boldsymbol{Q}}_i \mid 0 \leqslant i \leqslant p\}$ as the ordinary basis vectors of the $(p+1)$-dimension space

$$(\overline{\boldsymbol{Q}}_i = \Big(\underbrace{0, \dots, 0}_{i-1}, 1, \underbrace{0, \dots, 0}_{p-i+1})^{\mathrm{T}}\Big),$$

and call the curve clamping algorithm to get the control points of the clamped curve $\{\overline{\boldsymbol{P}}_i \mid 0 \leqslant i \leqslant p\}$. Then, we have

$$W_2 = [\overline{\boldsymbol{P}}_0, \overline{\boldsymbol{P}}_1, \dots, \overline{\boldsymbol{P}}_p]^{\mathrm{T}}.$$

After W_1 and W_2 are determined, M_p can be computed through $M_p = W_1 W_2$.

3 Extension algorithm for B-spline curves

The extension problem is described as follows. The input is a clamped B-spline curve $\boldsymbol{P}(t)$ with degree p, control points $\{\boldsymbol{P}_i \mid 0 \leqslant i \leqslant n\}$, and knot vector

$$\{\underbrace{t_0, \dots, t_p}_{p+1}, t_{p+1} \dots, t_n, \underbrace{t_{n+1}, \dots, t_{n+p+1}}_{p+1}\}, \tag{3}$$

where $t_0 = \cdots = t_p$ and $t_{n+1} = \cdots = t_{n+p+1}$. We need to extend it to the target points $\{\boldsymbol{R}_i \mid 1 \leqslant i \leqslant s\}$, and fit high-order derivative constraints $\{\boldsymbol{R}_i^{(0)}, \boldsymbol{R}_i^{(1)}, \dots, \boldsymbol{R}_i^{(m_i)}\}$ at each target point $(0 \leqslant m_i < p)$.

The original domain of $\boldsymbol{P}(t)$ is defined over the interval $[t_p, t_{n+1}]$. If a new control point $\boldsymbol{P}_{n+1}$ and a new knot t_{n+p+2} are added to the curve, its domain will expand to $[t_p, t_{n+2}]$. Notice that the point $\boldsymbol{P}_{n+1}$ only affects $\boldsymbol{P}(t)$ in the interval $[t_{n+1}, t_{n+2}]$, and the shape of $\boldsymbol{P}(t)$ within its origin domain remains unchanged. So, $\boldsymbol{P}(t)$ can be extended to a target point $\boldsymbol{R}$ by just setting $\boldsymbol{P}(t_{n+2}) = \boldsymbol{R}$ and solving the value of $\boldsymbol{P}_{n+1}$. The method of Hu et al. [1] used the inverse process of the de Boor algorithm to evaluate new control points, which has a relatively high time cost and is hard to handle derivative constraints at the target points. To overcome this problem, our algorithm uses another way to evaluate new control points.

Our basic idea is using uniform B-spline curves to generate the extension curve. The algorithm consists of three steps. First in preprocessing, we construct a uniform B-spline curve that shares the same boundary with the input curve $\boldsymbol{P}(t)$ and regard it as a new input curve. Second in the main process, we extend the uniform B-spline curve step by step, with one more target point added in each step. Third in postprocessing, we clamp the resulting curve and join it together with $\boldsymbol{P}(t)$ to get the final result.

3.1 Preprocessing

Assume that knot t_n has multiplicity x_n in knot vector T_1. For the input curve $\boldsymbol{P}(t)$, we insert knot t_n for $p - x_n$ times so as to split it into two B-spline sub-curves $\boldsymbol{P}_l(t)$ and $\boldsymbol{P}_r(t)$. $\boldsymbol{P}_l(t)$ is defined over $\{\underbrace{t_p,\ldots,t_p}_{p+1},t_{p+1}\ldots,t_{n-1},\underbrace{t_n,\ldots,t_n}_{p+1}\}$ and $\boldsymbol{P}_r(t)$ is defined over $\{\underbrace{t_n,\ldots,t_n}_{p+1},\underbrace{t_{n+1},\ldots,t_{n+1}}_{p+1}\}$.

Set $\delta = t_{n+1} - t_n$. Since $\boldsymbol{P}_r(t)$ is a Bézier curve, we use curve unclamping to convert it to a uniform B-spline curve $\boldsymbol{C}_0(t)$, with control points $\{\boldsymbol{Q}_i \mid 0 \leqslant i \leqslant p\}$ and knot vector $\{r_0, r_1, \ldots, r_{2p}, r_{2p+1}\}$, where $r_i = t_n + (i-p)\delta$.

$\boldsymbol{C}_0(t)$ is uniform and completely coincides with $\boldsymbol{P}(t)$ at the right-side end point, so from now on we regard $\boldsymbol{C}_0(t)$ as the new input curve.

3.2 Extend the curve to target points

We implement the extension algorithm step by step. In each step, one more target point is added to the current curve. Without loss of generality, we consider step k ($1 \leqslant k \leqslant s$). Before this step, the original curve has become $\boldsymbol{C}_{k-1}(t)$, with control points $\{\boldsymbol{Q}_i \mid 0 \leqslant i \leqslant n_{k-1}\}$ and knot vector $\{r_0, r_1, \ldots, r_{n_{k-1}+p+1}\}$, where $n_{k-1} = p + \sum_{i=1}^{k-1}(m_i+1)$. At step k, we need to extend $\boldsymbol{C}_{k-1}(t)$ to target point $\boldsymbol{R}_k$, with the specified derivatives $\{\boldsymbol{R}_k^{(1)}, \boldsymbol{R}_k^{(2)}, \ldots, \boldsymbol{R}_k^{(m_k)}\}$. The extension result will be $\boldsymbol{C}_k(t)$, with control points $\{\boldsymbol{Q}_i \mid 0 \leqslant i \leqslant n_k\}$ and knot vector $\{r_0, r_1, \ldots, r_{n_k+p+1}\}$, where $n_k = n_{k-1} + m_k + 1$.

Now $m_k + 1$ new control points $\{\boldsymbol{Q}_{n_{k-1}+1}, \boldsymbol{Q}_{n_{k-1}+2}, \ldots, \boldsymbol{Q}_{n_k}\}$ remain unknown. We compute their values from the following derivative constraints:

$$\boldsymbol{C}_k^{(m)}(r_{n_k+1}) = \boldsymbol{R}_k^{(m)}, \quad 0 \leqslant m \leqslant m_k, \tag{4}$$

where $\boldsymbol{C}_k^{(m)}(t)$ is the mth-order derivative of $\boldsymbol{C}_k(t)$.

A recurrence formula is required to compute high-order derivatives for ordinary B-spline curves, but it is much easier for uniform B-spline curves. The derivative of a B-spline curve $\boldsymbol{P}(t)$ can be computed through the following formula [13]:

$$\boldsymbol{P}'(t) = \sum_{i=0}^{n-1} N_{i+1,p-1} \frac{p(\boldsymbol{P}_{i+1} - \boldsymbol{P}_i)}{u_{i+p+1} - u_{i+1}},$$

where $\{\boldsymbol{P}_i\}$ are the control points and $\{u_i\}$ are the knots. $\boldsymbol{P}'(t)$ is also a B-spline curve. Since $(u_{i+p+1} - u_{i+1})/p = \Delta$ is a constant value for uniform B-spline curves, we can further compute the high-order derivatives using the formula:

$$\boldsymbol{P}^{(m)}(t) = \sum_{i=0}^{n-m} N_{i+m,p-m}(t)\boldsymbol{P}_i^{(m)},$$

where $0 \leqslant m < p$ and

$$\boldsymbol{P}_i^{(m)} = \frac{1}{\Delta^m}\sum_{j=0}^{m} C_m^j (-1)^{m-j}\boldsymbol{P}_{i+j}.$$

That is to say, the mth-order derivative of a pth-degree uniform B-spline curve $\boldsymbol{P}(t)$ is also a uniform B-spline curve, with degree $(p-m)$ and control points $\{\boldsymbol{P}_i^{(m)} \mid 0 \leqslant i \leqslant n-m\}$.

From above, we know that $\boldsymbol{C}_k^{(m)}(t)$ is also a uniform B-spline curve, with degree $(p-m)$ and control points

$$\left\{ \{\boldsymbol{Q}_i^{(m)} = \frac{1}{\delta^m}\sum_{j=0}^{m} C_m^j (-1)^{m-j}\boldsymbol{Q}_{i+j} \mid 0 \leqslant i \leqslant n_k - m \right\}. \tag{5}$$

Since $\boldsymbol{C}_k(t)$ is uniform, we can use (2) to compute its curve points. Consider a knot r_l that lies in the interval $[r_l, r_{l+1}]$. Set $u = 0$ in (2), and we have

$$\boldsymbol{C}_k(r_l) = \sum_{i=0}^{p} M_p^{p,i}\boldsymbol{Q}_{l-p+i},$$

where $M_p^{p,i}$ is the element of M_p at the pth row and ith column. Notice that r_l also lies in the interval $[r_{l-1}, r_l]$, so the curve point at knot r_l does not depend on $\boldsymbol{Q}_l$, which means the element $M_p^{p,p}$ is always zero.

Similarly, since $\boldsymbol{C}_k^{(m)}(t)$ is uniform, we can compute its value at the end point r_{n_m+1} using

$$\boldsymbol{C}_k^{(m)}(r_{n_k+1}) = \sum_{i=0}^{p-m-1} M_{p-m}^{p-m,i} \boldsymbol{Q}_{n_k-p+i+1}^{(m)}. \tag{6}$$

Combining (4)–(6), we have

$$\boldsymbol{R}_k^{(m)} = \frac{1}{\delta^m} \sum_{l=0}^{p-1} w^{m,l} \boldsymbol{Q}_{n_k-p+l+1}, \tag{7}$$

where

$$w^{m,l} = \sum_{i=\max(0,l-m)}^{\min(l,p-m-1)} M_{p-m}^{p-m,i} C_m^{l-i} (-1)^{m-l+i}. \tag{8}$$

m ranges from 0 to m_k. In (7), we have m_k+1 constraints and m_k+1 unknown control points $\{\boldsymbol{Q}_{n_{k-1}+1}, \boldsymbol{Q}_{n_{k-1}+2}, \dots, \boldsymbol{Q}_{n_k}\}$, which produce a linear system

$$\begin{bmatrix} A^{0,0} & \dots & A^{0,m_k} \\ A^{1,0} & \dots & A^{1,m_k} \\ \vdots & \ddots & \vdots \\ A^{m_k,0} & \dots & A^{m_k,m_k} \end{bmatrix} \begin{bmatrix} \boldsymbol{Q}_{n_{k-1}+1} \\ \boldsymbol{Q}_{n_{k-1}+2} \\ \vdots \\ \boldsymbol{Q}_{n_k} \end{bmatrix} = \begin{bmatrix} \boldsymbol{B}_0 \\ \boldsymbol{B}_1 \\ \vdots \\ \boldsymbol{B}_{m_k} \end{bmatrix},$$

where A is a $(m_k+1) \times (m_k+1)$ matrix with elements

$$A^{m,j} = w^{m,j+p-m_k-1}, \tag{9}$$

and $\boldsymbol{B}$ is a (m_k+1)-dimensional vector with elements

$$\boldsymbol{B}_m = \delta^m \boldsymbol{R}_k^{(m)} - \sum_{l=0}^{p-m_k-2} w^{m,l} \boldsymbol{Q}_{n_k-p+l+1}. \tag{10}$$

Solve this linear system to get the values of control points $\{\boldsymbol{Q}_i\}$. Then, the resulting curve $\boldsymbol{C}_k(t)$ is determined.

3.3 Postprocessing

After s steps, all the target points are added to the curve, and the right segment $\boldsymbol{P}_r(t)$ of $\boldsymbol{P}(t)$ becomes the extension result $\boldsymbol{C}_s(t)$. Now we join it together with the left segment $\boldsymbol{P}_l(t)$ to get the final result.

Clamp curve $\boldsymbol{C}_s(t)$ to knot vector $\{\underbrace{r_n, \dots, r_n}_{p+1}, r_{n+1} \dots, r_{n_s}, \underbrace{r_{n_s+1}, \dots, r_{n_s+1}}_{p+1}\}$, join the curve $\boldsymbol{P}_l(t)$ and $\boldsymbol{C}_s(t)$ together, and remove the joint knot t_n for $(p-x_n)$ times. Then, the final curve is determined, and its knot vector is

$$\{\underbrace{t_p, \dots, t_p}_{p+1}, t_{p+1}, \dots, t_{n-1}, t_n = r_n, r_{n+1}, \dots, r_{n_s}, \underbrace{r_{n_s+1}, \dots, r_{n_s+1}}_{p+1}\},$$

where $n_s = \sum_{i=1}^{s}(m_i+1)$. The curve has $n+n_s+1$ control points.

The overall algorithm is shown in Algorithm 1.

Algorithm 1 B-spline curve extension to multiple target points

```
Require: A pth-degree B-spline curve $\boldsymbol{P}(t)$ defined over knot vector $T$ (in (3)), and target points with specified high-order
    derivatives $\{\{\boldsymbol{R}_i^{(m)} \mid 0 \leqslant m \leqslant m_i\} \mid 1 \leqslant i \leqslant s\}$.
Ensure: A B-spline resulting curve.
 1: Curve splitting. Assume knot value $t_n$ has multiplicity $x_n$ in $T$. Insert knot $t_n$ for $(p - x_n)$ times and split $\boldsymbol{P}(t)$ into
    two segments $\boldsymbol{P}_l(t)$ and $\boldsymbol{P}_r(t)$.
 2: Curve unclamping. Let $\delta = t_{n+1} - t_n$. Unclamp the right segment $\boldsymbol{P}_r(t)$ to get a uniform B-spline curve $\boldsymbol{C}_0(t)$.
 3: Iteration steps for extension. Set $n_k = p + \sum_{i=1}^{k}(m_i + 1)$ and $r_i = t_n + (i - n)\delta$. In each step $k$ $(1 \leqslant k \leqslant s)$, target
    point $\boldsymbol{R}_k$ is reached, and curve $\boldsymbol{C}_{k-1}(t)$ is extended to $\boldsymbol{C}_k(t)$, with control points $\{\boldsymbol{Q}_i \mid 0 \leqslant i \leqslant n_k\}$ and knot vector
    $\{r_0, r_1, \ldots, r_{n_k+p+1}\}$. The new control points are computed through the linear system
        $A \times [\boldsymbol{Q}_{n_{k-1}+1}\boldsymbol{Q}_{n_{k-1}+2} \cdots \boldsymbol{Q}_{n_k}]^{\mathrm{T}} = \boldsymbol{B}$,
    where $A$ and $\boldsymbol{B}$ are determined from (8)–(10).
 4: Curve clamping. Clamp the resulting curve $\boldsymbol{C}_s(t)$.
 5: Curve joining. Join the two curves $\boldsymbol{P}_l(t)$ and $\boldsymbol{C}_s(t)$ together, and remove the joint knot $t_n$ for $(p - x_n)$ times to get the
    final result.
```

4 Generalization for NURBS curves

Our algorithm can be generalized to handle NURBS curve extension. A pth-degree NURBS curve $\boldsymbol{P}(t)$ is defined as

$$\boldsymbol{P}(t) = \frac{\sum_{i=0}^{n} N_{i,p}(t) w_i \boldsymbol{P}_i}{\sum_{i=0}^{n} N_{i,p}(t) w_i} = \frac{\boldsymbol{A}(t)}{w(t)}, \tag{11}$$

where $\{N_{i,p}(t)\}$ are B-spline basis functions, $\{\boldsymbol{P}_i\}$ are the control points, and $\{w_i\}$ are the weights. We need to extend it to target points $\{\{\boldsymbol{R}_i^{(m)} \mid 0 \leqslant m \leqslant m_i\} \mid 1 \leqslant i \leqslant s\}$.

A rational curve in n-dimensional space can be represented as a polynomial curve in $(n+1)$-dimensional space using homogeneous coordinates [13]. So, our basic idea is to call our B-spline curve extension algorithm in homogeneous space. A NURBS curve $\boldsymbol{P}(t)$ can be represented as a B-spline curve $\boldsymbol{P}^w(t)$ in homogeneous space as

$$\boldsymbol{P}^w(t) = (\boldsymbol{A}(t), w(t))^{\mathrm{T}} = \sum_{i=0}^{n} N_{i,p}(t) \boldsymbol{P}_i^w,$$

where $\boldsymbol{P}_i^w = (w_i \boldsymbol{P}_i, w_i)^{\mathrm{T}}$ are the homogeneous coordinates.

We need to find the corresponding target points in homogeneous space. Assume that the target point $\{\boldsymbol{R}_i^{(m)} \mid 0 \leqslant m \leqslant m_i\}$ corresponds with the point $\{(\boldsymbol{X}_i^{(m)}, v_i^{(m)})^{\mathrm{T}} \mid 0 \leqslant m \leqslant m_i\}$ in homogeneous space. Then, at the point $\boldsymbol{R}_i$ the following equation should hold:

$$\boldsymbol{P}^{(m)}(t) = \boldsymbol{R}_i^{(m)}, \quad \boldsymbol{A}^{(m)}(t) = \boldsymbol{X}_i^{(m)}, \quad w^{(m)}(t) = v_i^{(m)}. \tag{12}$$

The mth-order derivatives of the NURBS curve $\boldsymbol{P}(t)$ are obtained (see [13]) through

$$\boldsymbol{P}^{(m)}(t) = \frac{\boldsymbol{A}^{(m)}(t) - \sum_{l=1}^{m} C_m^l w^{(l)}(t) \boldsymbol{P}^{(m-l)}(t)}{w(t)}. \tag{13}$$

Thus, substitute (12) into (13), and we have

$$\boldsymbol{X}_i^{(m)} = \sum_{l=0}^{m} C_m^l v_i^{(l)} \boldsymbol{R}_i^{(m-l)}. \tag{14}$$

We assign values for $\{v_i^{(m)} | \, 0 \leqslant m \leqslant m_i\}$ in advance, and use (14) to compute the corresponding values of $\boldsymbol{X}_i$. Then, the target points in homogeneous space are determined, and we can call the extension algorithm for B-spline curves to get the extension result. The overall process is shown in Algorithm 2.

5 Analysis and examples

First, we analyze the time cost of our algorithm. For each target point $\boldsymbol{R}_i$, evaluation of the matrices A and $\boldsymbol{B}$ costs $\Theta((m_i + 1)^2 p)$, and solving the linear system costs $\Theta((m_i + 1)^2)$. The preprocessing

```
Algorithm 2 NURBS curve extension to multiple target points
Require: A pth-degree NURBS curve $\boldsymbol{P}(t)$, and target points with specified high-order derivatives $\{\{\boldsymbol{R}_i^{(m)} \mid 0 \leqslant m \leqslant m_i\} \mid 1 \leqslant i \leqslant s\}$.
Ensure: A NURBS resulting curve.
1: Get the corresponding B-spline curve $\boldsymbol{P}^w(t)$ in homogeneous space.
2: Compute the corresponding target points $\boldsymbol{R}_i^{w(m)} = (\boldsymbol{X}_i^{(m)}, v_i^{(m)})^{\mathrm{T}}$ in homogeneous space. Assign values for $\{v_i^{(m)} \mid 0 \leqslant m \leqslant m_i\}$ (user defined), and then compute the values of $\{\boldsymbol{X}_i^{(m)} \mid 0 \leqslant m \leqslant m_i\}$ using (14).
3: Take $\boldsymbol{P}^w(t)$ as the original curve, and $\boldsymbol{R}_i^{w(m)}$ as the target points. Call Algorithm 1 to get the B-spline resulting curve $\boldsymbol{Q}^w(t)$ in homogeneous space.
4: Get the resulting NURBS curve $\boldsymbol{Q}(t)$ from $\boldsymbol{Q}^w(t)$.
```

Table 1 Time comparison between two methods

Curve degree	Point count	Execution time (μs)			Curve degree	Point count	Execution time (μs)		
		Hu's	Ours	Ratio			Hu's	Ours	Ratio
4	8	1.19	1.10	92%	5	8	1.58	1.29	82%
	16	1.92	1.23	64%		16	2.65	1.45	55%
	32	3.39	1.54	45%		32	4.79	1.74	36%
	64	6.28	2.06	32%		64	9.01	2.25	25%

and postprocessing steps consist of a curve splitting, a curve unclamping, a curve clamping, and a curve joining, respectively, among which each operation costs $\Theta(p^2)$. So, the whole time cost of our algorithm is $\Theta(p^2 + p\sum_{i=0}^{s}(m_i+1)^2)$.

Compared with the method of Hu et al. [1], our method has a lower time complexity. Hu et al. [1] used the inverse process of the de Boor algorithm to evaluate the new control points, which costs $\Theta(p^2)$ in every step. Assuming that the count of target points is s, their method has a time complexity of $\Theta(sp^2)$. Meanwhile, if derivative constraints are not demanded at every target point, the time complexity of our method will be only $\Theta(p^2 + sp)$, which is better than theirs.

Table 1 shows the time comparison between our method and Hu et al. [1]. The degree of original B-spline curves ranges from 4 to 5, and the count of target points ranges from 8 to 64. The points are randomly located in 3D space. We run the process 10000 times for each case and record the average execution time. As shown in Table 1, as the degree of curves and the count of target points increase, the time efficiency of our method also rises.

We illustrate several results of our method as the following. A simple example is shown in Figure 1. We extend the initial curve to multiple target points so as to construct the outer contour of a sport utility vehicle (Figure 1(a)).

In Figure 2, we use our method to draw the shape of character '@'. In Figure 3, we use our method to reconstruct the shape of a human face profile. In these examples, we first manually specify some simple initial curves, and then sample the target points from original pictures. Tangent vectors are also specified to improve the shapes of resulting curves.

Figure 4 shows an example of constructing a closed curve. We extend the original curve from one end to the other, such that the resulting curve is closed.

6 Conclusion

This paper presents an algorithm for B-spline curve extension. We construct a uniform B-spline curve from the original curve and use it to generate extension curves. Operations of uniform B-spline curves are more efficient than normal B-spline curves, so our method is simple and fast. Assuming that the curve degree is p and the count of target points is s, our method only costs a time complexity of $\Theta(p^2 + sp)$, while the method of Hu et al. [1] costs a time complexity of $\Theta(sp^2)$. Arbitrary-order derivative constraints at the target points can be specified in our method. Generalization for NURBS curve extension is also discussed.

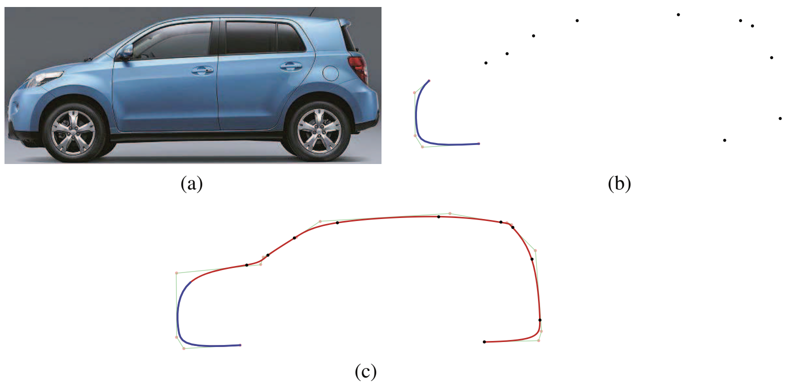

Figure 1 (Color online) Constructing outer contour of a sport utility vehicle. (a) The original picture, (b) the initial B-spline curve and sampling points, and (c) the resulting curve.

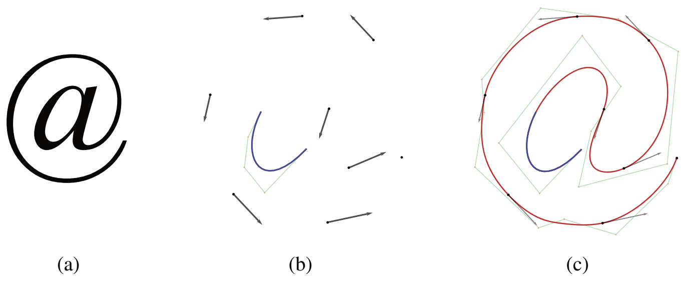

Figure 2 (Color online) Drawing the shape of character '@'. (a) The standard shape of the symbol "@", (b) the initial curve, target points, and tangent vectors, and (c) the resulting curve.

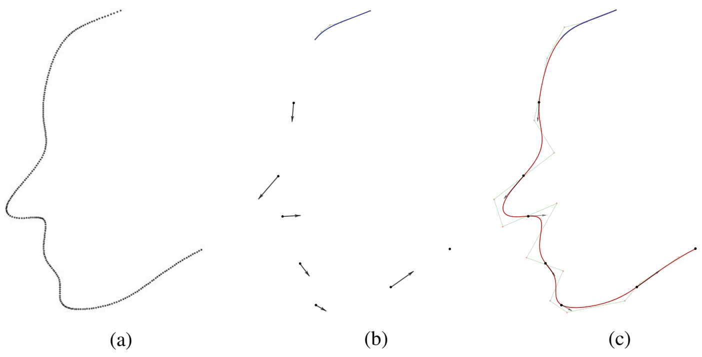

Figure 3 (Color online) Reconstructing human face profile. (a) The discrete points from original input data; (b) the target points sampled from (a) and the estimated tangent vectors; (c) the resulting curve.

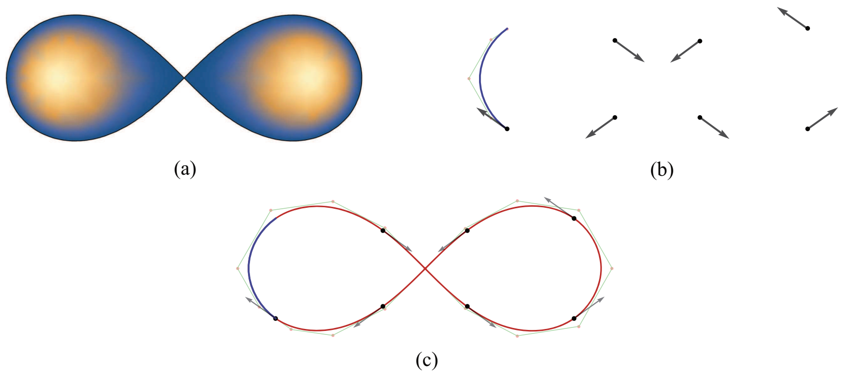

Figure 4 (Color online) Example of a closed curve. (a) illustrates an ideal dumb-bell shape. (b) shows the sampled target points and the estimated tangent vectors. (c) shows the resulting curve of our method, whose shape is close to (a).

7 Future work

In some cases where input points are irregularly distributed, our extension method may produce unexpected shapes as we use global uniform knot values. The method of Hu et al. [1] used the chord-length parameterization to generate new knots and also suffers from this problem. In this case, it is still a challenging problem to generate a reasonable shape with a low time cost.

An important application of curve extension is B-spline curve approximation of other known curves. The basic idea is taking input points sampled from the input curve and extending the initial B-spline curve to these points. Chen et al. [3, 4] have done some research on this topic and used the tangent directions of sampled points for curve extension. Since our extension method can specify high-order derivatives at each point, these approximation methods may be further improved.

Acknowledgements The research was supported by International Science & Technology Cooperation Program of China (Grant No. 2013DFE13120). Kanle SHI was supported by National Natural Science Foundation of China (Grant No. 61272235) and Open Funding Project of State Key Laboratory of Virtual Reality Technology and Systems, Beihang University (Grant No. BUAA-VR-14KF-05). Junhai YONG was supported by National Natural Science Foundation of China (Grant No. 91315302). Haichuan SONG was supported by National Natural Science Foundation of China (Grant No. 61173077).

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Hu S M, Tai C L, Zhang S H. An extension algorithm for B-splines by curve unclamping. Comput Aid Des, 2002, 34: 415–419
2 Liu Y J, Qiu R Q, Liang X H. NURBS curve blending using extension. J Zhejiang Univ Sci A, 2009, 10: 570–576
3 Chen X D, Ma W. Geometric point interpolation method in $\mathbb{R}^3$ space with tangent directional constraint. Comput Aid Des, 2012, 44: 1217–1228
4 Chen X D, Ma W, Paul J C. Cubic B-spline curve approximation by curve unclamping. Comput Aid Des, 2010, 42: 523–534
5 Shetty S, White P. Curvature-continuous extensions for rational B-spline curves and surfaces. Comput Aid Des, 1991, 23: 484–491
6 Zhou Y F, Zhang C M, Gao S S. Extension of B-spline curves with G^2 continuity. In: Proceedings of Advances in Visual Computing. Berlin: Springer, 2008. 1096–1105
7 Fan H, Zhang C M, Li J J. Extension algorithm for B-splines with GC^2-continuous (in Chinese). Chin J Comput, 2005, 28: 933–938
8 Mo G L, Zhao Y N. A new extension algorithm for cubic B-splines based on minimal strain energy. J Zhejiang Univ Sci A, 2006, 7: 2043–2049
9 Xu G, Wang G Z. Extensions of uniform cubic B-spline curve with local shape parameters (in Chinese). J Comput Res Develop, 2007, 44: 1032–1037
10 Xu J. Smooth B-spline curves extension with ordered points constraint. Adv Mater Res, 2011, 311: 1439–1445
11 Zhang T, Wang X, Jiang Q, et al. G^2-continuity extension algorithm for disk B-spline curve. In: Proceedings of Computer-Aided Design and Computer Graphics (CAD/Graphics), Guangzhou, 2013. 413–414
12 Jiang Q, Wu Z, Zhang T, et al. An extension algorithm for ball B-spline curves with G^2 continuity. In: Proceedings of International Conference on Cyberworlds, Yokohama, 2013. 252–258
13 Piegl L A, Tiller W. The NURBS Book. Berlin: Springer-Verlag, 1995
14 Mortenson M E. Geometric Modeling. New York: Wiley, 1985