Journal of Zhejiang University SCIENCE
ISSN 1009-3095
http://www.zju.edu.cn/jzus
E-mail: jzus@zju.edu.cn

A fast direct point-by-point generating algorithm for B Spline curves and surfaces*

LI Zhong (李 重)†[1,3], HAN Dan-fu (韩丹夫)[2]

([1]*Department of Information and Computing Science, Zhejiang University of Sciences, Hangzhou 310033, China*)

([2]*Department of Mathematics, Zhejiang University, Hangzhou 310027, China*)

([3]*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200030, China*)

†E-mail: lizhongzju@hotmail.com

Received Feb. 2, 2004; revision accepted July 1, 2004

Abstract: Traditional generating algorithms for B Spline curves and surfaces require approximation methods where how to increment the parameter to get the best approximation is problematic; or they take the pixel-based method needing matrix transformation from B Spline representation to Bézier form. Here, a fast, direct point-by-point generating algorithm for B Spline curves and surfaces is presented. The algorithm does not need matrix transformation, can be used for uniform or nonuniform B Spline curves and surfaces of any degree, and has high generating speed and good rendering accuracy.

Key words: Point-by-point generating algorithm, B Spline curve, B Spline surface, Pixel
doi:10.1631/jzus.2005.A0502 **Document code:** A **CLC number:** TP391

INTRODUCTION

Curve generating algorithms are important in computer graphics and CAD/CAM. For some simple curves such as line, circle, ellipse, etc., we have some rendering algorithms such as DDA algorithm for line, Bresenham algorithm and midpoint algorithm for circle, Pitteway's algorithm for ellipse, etc. (Bresenham, 1977; Pitteway, 1985; Ammeraal, 1987; Foley *et al.*, 1990; 1993).

For rendering parametric curves and surfaces, a popular method is to compute a set of points along the curves and surfaces, then join them by line segments in a smooth approximation. This algorithm needs floating point arithmetic; how to increment the parameter to get the best approximation possible is not obvious. Low degree curves and surfaces are used to approximate high degree parametric curves and surfaces.

We also can use pixel-based method to draw: we sample many points along the curves and surfaces, round them to the nearest integer and set each pixel to where the computed point falls in. This method provides the smoothest curves and surfaces possible at the expense of computation time as many points have to be computed to ensure that no gaps are created along the curves and surfaces (Anantakrishnan and Piegl, 1992).

In order to speed up the computation, a fast point-by-point generating algorithm for parametric curves and surfaces is introduced (Huang and Zhu, 2001). But this algorithm is based on Bézier curves and surfaces; B Spline curves and surfaces must first be decomposed into piecewise Bézier form, after which the algorithm can be used to render. This algorithm cannot directly render B Spline curves and surfaces well.

In recent years, B Spline curves and surfaces, rational B Spline curves and surfaces, especially nonuniform rational B Spline curves and surfaces (NURBS) have become standard tools for representing many geometric entities used in a variety of visualization oriented fields such as Computer

*Project (No. G1998030401) supported by the National Natural Science Foundation of China

Graphics, CAD/CAM and Vision/Image Processing, but there seems to be no generally accepted technique for fast, direct and accurate display of them. So results of study on B Spline curves and surfaces' generating algorithm will have an important influence in these fields. In this paper, we provide a direct point-by-point generating algorithm for B Spline curves and surfaces. The algorithm does not need matrix transformation from B Spline representation to Bézier form, and uses integer equation and difference method to simplify the computation. These measures can improve the algorithm's effect. The algorithm can be used for uniform or nonuniform B Spline curves and surfaces of any degree. As we know, Bézier curves and surfaces is a special formation of B Spline curves and surfaces as long as we adjust knot vectors; this algorithm can also draw Bézier curves and surfaces and has broad applications.

PARAMETER CURVES' POINT-BY-POINT GENERATING ALGORITHM (HUANG AND ZHU, 2001)

When we render parameter curves such as Bézier curves, we can write them as: $x=f(t)$, $y=g(t)$, $t\in[0,1]$. $f(t)$, $g(t)$ would have to be rounded to integers in order for x, y to be integers. We first discuss $x=f(t)$. Supposing there exists an integer n satisfying a condition on the maximum of $|f'(t)|$ and $|g'(t)|$. Let t be i/n ($0\le i\le n$); correspondingly x_i becomes $[f(i/n)]$. Here $[f(x)]$ means the rounded integer part of $f(x)$. From the Lagrange mean value theorem, we can get

$$\left|f\left(\frac{i+1}{n}\right)-f\left(\frac{i}{n}\right)\right|=\left|\frac{f'(\theta)}{n}\right|\le 1.$$

This inequality ensures that the draw step is not more than one pixel, i.e., the rendered curve has no gap.

If $f(t)$'s coefficients are rational numbers, we always change $f(t)$ to an integer equation. For example, if m is the least common multiple of all coefficient's denominators, $f(t)$'s highest degree is k, then let $N=n^k m$. Multiplication of $x=f(t)$ by N changes $x=f(t)$ into an integer equation

$$Nx_i=\phi(i)+z_i, \tag{1}$$

where $\phi(i)=Nf(i/n)$, x_i, z_i are integers, and $|z_i|\le N/2$.

We can get all the pixels of the curve point-by-point from Eq.(1). We denote $\Delta\phi(i)=\phi(i+1)-\phi(i)$. If x_i is known, then x_{i+1} should be satisfied with

$$\begin{aligned}Nx_{i+1}&=\phi(i+1)+z_{i+1}=\phi(i)+\Delta\phi(i)+z_{i+1}\\&=(Nx_i-z_i+\Delta\phi(i))+z_{i+1}\end{aligned}$$

$$\because|\Delta\phi(i)|=N\left|f\left(\frac{i+1}{n}\right)-f\left(\frac{i}{n}\right)\right|\le N,\ \text{and } |z_i|\le N/2.$$

$$\therefore|\Delta\phi(i)-z_i|\le 3N/2.$$

Hence, x_{i+1} can only be x_i-1, x_i, or x_i+1. We get x_{i+1} by the following recursive relation

$$x_{i+1}=\begin{cases}x_i-1 & \text{when } \Delta\phi(i)-z_i<-1/2N\\ x_i & \text{when } -1/2N\le\Delta\phi(i)-z_i<1/2N\\ x_i+1 & \text{when } \Delta\phi(i)-z_i\ge 1/2N,\end{cases} \tag{2}$$

and

$$z_{i+1}=\begin{cases}z_i-\Delta\phi(i)-N, & \text{when } x_{i+1}=x_i-1\\ z_i-\Delta\phi(i), & \text{when } x_{i+1}=x_i\\ z_i-\Delta\phi(i)+N, & \text{when } x_{i+1}=x_i+1.\end{cases} \tag{3}$$

For $y=g(t)$, we use similar method. Let t be i/n, $g(t)$ change to

$$My_i=\varphi(i)+q_i,\ |q_i|\le M/2. \tag{4}$$

n should be considered in terms of x, y, namely $n=\max\limits_{0\le t\le 1}\{|f'(t)|,|g'(t)|\}$. For Bézier curves, n can be calculated by theorem, so we can draw Bézier curves point-by-point.

ALGORITHM FOR DIRECTLY GENERATING B SPLINE CURVES AND SURFACES

The B Spline curve of order k (degree $k-1$) is a vector-valued piecewise polynomial function of the form

$$P(t)=\sum_{i=0}^{l}P_iN_{i,k}(t),\ t\in[t_{k-1},t_{l+1}].$$

where P_i denote the control points. $N_{i,k}(t)$ are the B Spline functions defined over the knot vector

$\boldsymbol{T}_{l,k}=\{t_i\}_{i=0}^{l+k}$.

The main rendering problem for B Spline curves is that the B Spline functions are different in different knot vector interval $t\in[t_{j-1},t_j]$, $0\le t_{j-1},t_j\le 1$, $j=k, k+1, \ldots, l+1$. If we draw B Spline curves using the above algorithm, the current method must first decompose them to piecewise Bézier form by matrix transformation.

To directly draw B Spline curves point-by-point, we must consider two conditions. First, the B Spline curve segment should be continuous at the inner knot vector; this can be guaranteed by the B Spline curve's property. Second, in order not to create a gap in the curve, it should always satisfy $\left|f\left(\frac{i+1}{n}\right)-f\left(\frac{i}{n}\right)\right|\le 1$ whether $(i+1)/n$, i/n are in the same knot interval or not in the same knot interval. In fact, for $x=f(t)$, $t\in[0,1]$. Supposing $n=\max\limits_{0\le t\le 1}|f'(t)|$, when $(i+1)/n$, i/n are in the same knot vector interval $[t_{j-1},t_j]$, $f(t)$ are in the same format. Obviously $\left|f\left(\frac{i+1}{n}\right)-f\left(\frac{i}{n}\right)\right|\le 1$. When $(i+1)/n$, i/n are not in the same knot vector interval, supposing dividing knot is t_j, parameter functions that are on the two sides of this knot are function $f_1(t)$, $f_2(t)$ each; when $f_1(t)$, $f_2(t)$ are continuous at this knot, we have $f_1(t_j)=f_2(t_j)$; we can know that

$$\begin{aligned}\left|f\left(\frac{i+1}{n}\right)-f\left(\frac{i}{n}\right)\right|&=\left|f_2\left(\frac{i+1}{n}\right)-f_1\left(\frac{i}{n}\right)\right|\\&\le\left|f_2\left(\frac{i+1}{n}\right)-f_2(t_j)\right|+\left|f_1(t_j)-f_1\left(\frac{i}{n}\right)\right|\\&\le n\left(\frac{i+1}{n}-t_j\right)+n\left(t_j-\frac{i}{n}\right)\le 1.\end{aligned}$$

So we can draw B Spline curves directly. From the computation result, we found we can use the same recursive Eqs.(2), (3) to compute x_{i+1} and z_{i+1}.

A key point in using this algorithm drawing for B Spline curves is to compute $|f'(t)|$ and $|g'(t)|$'s upper bound value. From the experiment results, we found that a somewhat large upper bound value increases the loop numbers and causes drawing of the same pixel many times so that it adversely influences the algorithm's effect. We should get the minimum possible upper bound.

For popular uniform cubic B Spline curve

$$\begin{aligned}f(t)=&\frac{(-t^3+3t^2-3t+1)}{6}X_0+\frac{(3t^3-6t^2+4)}{6}X_1\\&+\frac{(-3t^3+3t^2+3t+1)}{6}X_2+\frac{t^3}{6}X_3\end{aligned}$$

From the computation, we can get $f'(t)$'s extreme value is $[X_2-X_0-V^2/W]/2$ at $t=-V/W$, where $V=X_2-2X_1+X_0$, $W=X_3-3(X_2-X_1)-X_0$.

At the endpoints, $|f'(t)|$'s value are $|X_2-X_0|/2$ and $|X_3-X_1|/2$. So,

$$nx=\begin{cases}\max(|X_2-X_0|,|X_2-X_0-V^2/W|,|X_3-X_1|)/2\\\qquad\text{when } -V/W\in[0,1]\\\max(|X_2-X_0|,|X_3-X_1|)/2\\\qquad\text{when } -V/W\notin[0,1]\end{cases}$$

In the same way, we get minimum upper bound ny, then let $n=\max(nx, ny)$, so we get the value n.

For nonuniform cubic B Spline curves, because its derivative equation is degree 2, we can get $|f'(t)|$ and $|g'(t)|$'s minimum upper bound like uniform cubic B Spline curves.

For quadratic B Spline curves, we can easily get $|f'(t)|$ minimum upper bound.

Normally we seldom use B Spline curves whose degree is above 3 in the curve design. If there is a need to draw high degree B Spline curves by this algorithm, we can get the minimum upper bound value via numerical methods.

When we use surfaces in computer graphics, ray-tracing does not require discretizing surfaces via a set of integer points in 3D space. If there is a need to represent B Spline surfaces in this manner, we can generalize the algorithm to approximate surfaces.

ALGORITHM

When we render the curves and surfaces, we can use the difference method to improve the algorithm's effect (Rappoport, 1991; Klassen, 1991). In this algorithm, when we compute $\Delta\phi(i)=\phi(i+1)-\phi(i)$ we can use difference formula by addition arithmetic:

$$\Delta^{k+1}\phi(i)=\Delta^{k}\phi(i+1)-\Delta^{k}\phi(i).$$

We notice that degree h polynomial's order h difference is constant, so when we know all order differences $\Delta^{k}\phi(i)$, we can get all order difference $\Delta^{k}\phi(i+1)$ by h times' addition arithmetic according to difference formula

$$\Delta^{k-1}\phi(i+1)=\Delta^{k-1}\phi(i)+\Delta^{k}\phi(i),\ k=1, 2, \ldots, h.$$

where order 0 differencing of ϕ is function ϕ itself.

We can use similar method for computing $\Delta\varphi(i)$ in $y=g(t)$.

In the experiment, changing parametric equations to integer form tended to make the coefficients very large, so we used multiple precision integers. Or we drew the whole curves and surfaces by some segments divided by knot vector interval, since we always made sure that the segments are continuous at the inner knot vectors and that a gap did not appear in the segments. In the algorithm, in order to keep the curve 'smooth' and avoid rendering too many redundant points caused by 'corner points' (Fig.1), we can use two variables *xprev*, *yprev* to detect and eliminate cases where a pixel has two or more subsequent pixels in its immediate neighborhood. By the Knuths's diamond rule (Donald, 1986), when a pixel centered at some point (x, y) is turned on if the curve contains a point (xx, yy) such that

$$-1/2 < xx + yy - x - y < 1/2$$

and

$$-1/2 < yy - xx + y - x < 1/2$$

So, the 'corner point' may be the pixel that is turned on; but in many experiments, we find it was very rare that the corner point must be rendered and the haphazard effect of corner point removal could be ignored which kept the curve's shape accurate.

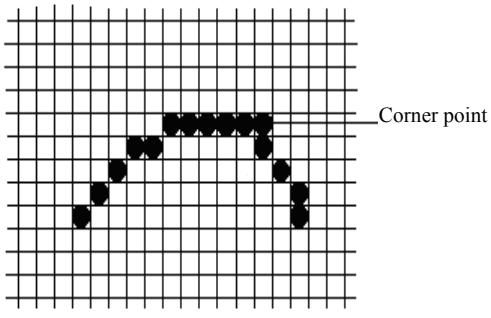

Fig.1 Corner point

The step in drawing the B Spline curves were as follows:

Step 1: Choose proper value N, M, compute the value n.

Let B Spline curves change to integer Eqs.(1), (4).

Step 2: From $\phi(i)$ and $\varphi(i)$, $i=0, 1, \ldots, n$.

Compute ϕ and φ's all order differences a_k, b_k at $i=0$.

Step 3: From $Nx=\phi(0)+z_1$, $|z_1|\le N/2$ and $My=\varphi(0)+q_1$, $|q_i|\le M/2$,

Compute first pixel point (x, y) and z_1, q_1 at $i=0$.

Step 4: Draw the first pixel (x, y), and let *xold*=*x*, *yold*=*y*, *xprev*=*x*, *yprev*=*y*.

Step 5: For $i=1, 2, \ldots, n$ loop.

Using Eqs.(2), (3) and difference formula, we compute x_{i+1}, z_{i+1}, y_{i+1}, q_{i+1}.

```
If ((|x_{i+1}–xold| >1) or (|y_{i+1}–yold | >1))
    {
      Draw pixel (xprev, yprev),
      Let xold=xprev, yold=yprev.
      }
   xprev=x_{i+1}, yprev=y_{i+1}.
```

Step 6: Draw the last point (x, y).

EXPERIMENTAL DETAILS

This algorithm was used to render B Spline curve and surface by VC6.0 language and OpenGL Graphics tool where the computer was CPU P4/1.8 G and EMS memory was 256 MB. Fig.2 shows a cubic B Spline curve where $P(t)=\sum_{i=0}^{n} P_i N_{i,k}(t)$, $n=3$, $k=4$ (degree 3), knot vectors t_j are nonuniformly located in [0,1], and control points are $P_0(100, 160)$, $P_1(140, 90)$, $P_2(240, 190)$, $P_3(280, 140)$.

Fig.3 shows a bi-cubic B Spline surface rendered by this algorithm (the surface had been treated for the lighting effect). Tables 1, 2 give time comparisons used by some different algorithms to draw B Spline curve and surface and Figs.2, 3 show B Spline curve and B Spline surface.

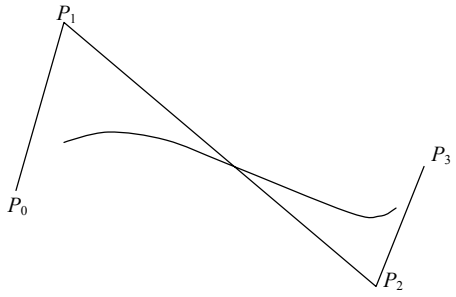

Fig.2 B Spline curve

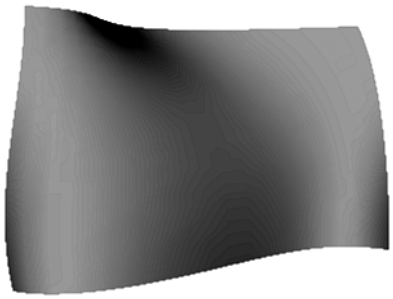

Fig.3 B Spline surface

Table 1 Time comparison with different algorithms drawing B Spline curve for 10000 times

Algorithm	Time (s)
Algorithm based on approximation	38.27
Algorithm requiring matrix transformation	46.48
Algorithm in this paper	30.51

Table 2 Time comparison with different algorithms drawing B Spline surface for 10000 times

Algorithm	Time (s)
Algorithm based on approximation	88.43
Algorithm requiring matrix transformation	107.50
Algorithm in this paper	83.81

From the rendering time comparison, because we do not need matrix transformation from B Spline representation to Bézier form; change the parameter equation to integer equation; use the difference method to calculate by addition arithmetic; all these measures can make the algorithm faster.

On the other side of considering curve and surface's shape, because we use some variables to get rid of redundant points, the curve and surface generated by this algorithm are some 'smooth' and have an accurate shape.

CONCLUSION

This paper's fast and direct point-by-point generating algorithm for B Spline curves and surfaces does not need matrix transformation, uses integer equation and difference method to compute $\Delta\phi(i)$ and $\Delta\varphi(i)$; all these measures can speed up the generating time while keeping the curve and surface's accuracy. This point-by-point generating algorithm can be directly used for uniform or nonuniform B Spline curves and surfaces of any degree, even for Bézier curves and surfaces, and has broad applications.

References

Ammeraal, L., 1987. Computer Graphics for IBM PC. John Wiley & Sons, New York.

Anantakrishnan, N., Piegl, L., 1992. Integer de casteljau algorithm for rasterizing NURBS curves. *Computer Graphics Forum*, **11**(2):151-162.

Bresenham, J., 1977. A linear algorithm for incremental digital display of circular arcs. *Communications of ACM*, **20**(2):100-106.

Donald, E., 1986. METAFONT the Program. Addison-Wesley, Reading, Massachusetts.

Foley, J., Dam, A., Feiner, S., Hughes, J., 1990. Computer Graphics: Principles and Practice. Addison-Wesley, Reading, Massachusetts.

Foley, J., Dam, A., Feiner, S., Hughes, J., 1993. Introduction to: Computer Graphics. Addison-Wesley, Reading, Massachusetts.

Huang, Y., Zhu, G., 2001. A fast point-by-point generating algorithm for rational parametric curve. *Chinese Journal of Computer*, **24**(8):809-814 (in Chinese).

Klassen, R., 1991. Integer forward differencing of cubic polynomials: Analysis and algorithms. *ACM Transactions on Graphics*, **10**(2):152-181.

Pitteway, M., 1985. Algorithms of Conic Generation. Fundamental Algorithms for Computer Graphics, NATO ASI Series, p.219-237.

Rappoport, A., 1991. Rendering curves and surfaces with hybrid subdivision and forward differencing. *ACM Transactions on Graphcis*, **10**(4):323-341.