Vol. 11, No. 3: 17–26

A Bidirectional Generating Algorithm for Rational Parametric Curves

Zhong Li
Shanghai Jiao Tong University and Zhejiang Sci-Tech University

Lizhuang Ma
Shanghai Jiao Tong University

Abstract. A generating algorithm for rational quadratic Bézier curves is provided. In the algorithm, we find the recurrence relation to get every pixel along the curve, using the bidirectional strategy for rendering and the forward and backward difference method for computing. The algorithm has a fast generating speed, and the curves keep a reasonable rendering accuracy. This algorithm can easily be generalized to render higher-degree rational Bézier curves and other rational parametric curves; it has broad applications. Source code is available online at the website listed at the end of this paper.

1. Introduction

Generating algorithms for curves are important in computer graphics and CAD/CAM. Many efforts have been made to develop efficient generating algorithms. For some simple curves such as lines, circles, ellipses, etc., we have rendering algorithms such as the DDA algorithm for lines, the Bresenham algorithm and midpoint algorithm for circles, the Pitteway algorithm for ellipses, etc. [Bresenham 77, Van Aken 84, Pitteway 85, Foley et al. 90, Yao and Rokne 98].

For rendering rational parametric curves, there seems to be no generally accepted technique for their fast and accurate display. A popular method is

1086-7651/06 $0.50 per page

to compute a set of points along the curves, 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, and high-degree parametric curves are replaced by low-degree parametric curves. Or, we use a pixel-based method to draw: we sample many points along the curves, then round them to the nearest integer and set each pixel to where the computed point falls. This method provides the smoothest curves possible at the expense of computation time since many points have to be computed to ensure that no gaps are created along the curves.

In this paper, we provide a fast generating algorithm for rational quadratic Bézier curves. In the algorithm, we first find the recurrence relation of every point, then use the bidirectional strategy to render and the forward and backward difference method to calculate. The experimental results show that the algorithm has a fast generating speed and the curves keep a reasonable rendering accuracy. This algorithm can also be generalized to higher-degree rational Bézier curves and other rational parametric curves; it has broad applications. Source code is available online at the address listed at the end of this paper.

2. Generating Algorithm for Rational Quadratic Bézier Curves

The standard rational quadratic Bézier curve can be written as

$$P(t) = \frac{(1-t)^2 w_0 P_0 + 2t(1-t) w_1 P_1 + t^2 w_2 P_2}{(1-t)^2 w_0 + 2t(1-t) w_1 + t^2 w_2}, \quad t \in [0,1],$$

where P_0, P_1, P_2 are control points and w_0, w_1, w_2 are weights of corresponding control points. Usually, $w_i > 0$, $w_0 = w_2 = 1$, and $w_1 = w$.

When we render a rational quadratic Bézier curve, we can write it as $x = f(t) = \frac{u(t)}{v(t)}$, $y = g(t) = \frac{r(t)}{v(t)}$, $t \in [0,1]$, where $f(t), g(t)$ would have to be rounded to integers for x, y to be integers. We first discuss $x = f(t)$. Suppose there exists an integer n satisfying

$$n \geq \max_{0 \leq t \leq 1} |f'(t)| .$$

Let t be $\frac{i}{n}$ $(0 \leq i \leq n)$. Correspondingly, x_i becomes $[f(\frac{i}{n})]$, where $[f(\frac{i}{n})]$ means the rounded integer part of $f(\frac{i}{n})$. From the Lagrange mean value theorem, we note that

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

This inequality ensures that the drawing step is not more than one pixel, i.e., the rendered curve does not have gaps (Huang and Zhu, 2001).

Obviously, $u(t)$ and $v(t)$ are degree-2 polynomials. If m is the least common multiple of all the denominator coefficients of $(1-t)^2, t(1-t), t^2$, multiply $u(t)$ and $v(t)$ by $n^2 \cdot m$ to get

$$f\left(\frac{i}{n}\right) = \frac{n^2 \cdot m \cdot u(\frac{i}{n})}{n^2 \cdot m \cdot v(\frac{i}{n})} \equiv \frac{\bar{u}(i)}{\bar{v}(i)},$$

where $\bar{u}(i), \bar{v}(i)$ are polynomials in i whose coefficients are integers.

So $x = f(t)$ can change to the integer equation

$$\bar{v}(i) \cdot x_i = \bar{u}(i) + z_i, |z_i| \le |\bar{v}(i)|/2, \tag{2}$$

where x_i, z_i are integers.

If x_i is known, then x_{i+1} should be satisfied with

$$\bar{v}(i+1) \cdot x_{i+1} = \bar{u}(i+1) + z_{i+1}, |z_{i+1}| \le |\bar{v}(i+1)|/2.$$

Denoting $\Delta\varphi(i) = \frac{\bar{u}(i+1)}{\bar{v}(i+1)} - \frac{\bar{u}(i)}{\bar{v}(i)}$, thus

$$x_{i+1} = (x_i + \Delta\varphi(i) - \frac{z_i}{\bar{v}(i)}) + \frac{z_{i+1}}{\bar{v}(i+1)},$$

where

$$|\Delta\varphi(i)| = |\frac{\bar{u}(i+1)}{\bar{v}(i+1)} - \frac{\bar{u}(i)}{\bar{v}(i)}| = |f(\frac{i+1}{n}) - f(\frac{i}{n})| \le 1, |\frac{z_i}{\bar{v}(i)}| \le \frac{1}{2}.$$

Hence,

$$|\Delta\varphi(i) - \frac{z_i}{\bar{v}(i)}| \le \frac{3}{2}.$$

Therefore, x_{i+1} can only be $x_i - 1, x_i, x_i + 1$; we can compute x_{i+1} by the following recursive relation:

$$x_{i+1} = \begin{cases} x_i - 1 & \text{when} \quad \frac{\bar{u}(i+1)}{\bar{v}(i+1)} - \frac{\bar{u}(i)}{\bar{v}(i)} - \frac{z_i}{\bar{v}(i)} < -\frac{1}{2}, \\ x_i & \text{when} \quad -\frac{1}{2} \le \frac{\bar{u}(i+1)}{\bar{v}(i+1)} - \frac{\bar{u}(i)}{\bar{v}(i)} - \frac{z_i}{\bar{v}(i)} < \frac{1}{2}, \\ x_i + 1 & \text{when} \quad \frac{\bar{u}(i+1)}{\bar{v}(i+1)} - \frac{\bar{u}(i)}{\bar{v}(i)} - \frac{z_i}{\bar{v}(i)} \ge \frac{1}{2}, \end{cases}$$

and

$$z_{i+1} = \begin{cases} z_i + x_i(\bar{v}(i+1) - \bar{v}(i)) + \bar{u}(i) - \bar{u}(i+1) - \bar{v}(i+1) \\ \qquad\qquad \text{when } x_{i+1} = x_i - 1, \\ z_i + x_i(\bar{v}(i+1) - \bar{v}(i)) + \bar{u}(i) - \bar{u}(i+1) \\ \qquad\qquad \text{when } x_{i+1} = x_i, \\ z_i + x_i(\bar{v}(i+1) - \bar{v}(i)) + \bar{u}(i) - \bar{u}(i+1) + \bar{v}(i+1) \\ \qquad\qquad \text{when } x_{i+1} = x_i + 1. \end{cases}$$

Similarly, letting t be $\frac{i}{n}$, we can change $y = g(t)$ to

$$\bar{v}(i) \cdot y_i = \bar{r}(i) + q_i, |q_i| \leq |\bar{v}(i)|/2. \tag{3}$$

So n should be considered in terms of x, y, namely $n = \max\limits_{0 \leq t \leq 1} \{|f'(t)|, |g'(t)|\}$.

A key point in using this algorithm is to compute the upper-bound value of $|f'(t)|$ and $|g'(t)|$. Suppose the rational Bézier curve is written as

$$C(t) = \frac{\sum\limits_{i=0}^{l} c_i w_i B_{i,n}(t)}{\sum\limits_{i=0}^{l} w_i B_{i,n}(t)},$$

where $B_{i,n}(t)$ are Bernstein polynomials, c_i are control points, and w_i are the weights.

Floater gave two estimates [Floater 92]:

$$|C'(t)| \leq l \cdot \frac{Q}{q} \max_{i,j=0,1,\cdots,l} |c_i - c_j|, \quad t \in [0,1] \tag{4}$$

and

$$|C'(t)| \leq l \cdot \frac{Q^2}{q^2} \max_{i=0,1,\cdots,l-1} |c_{i+1} - c_i|, \quad t \in [0,1], \tag{5}$$

where $Q = \max\limits_i \{w_i\}, q = \min\limits_i \{w_i\}$.

For the special case of rational quadratic Bézier curves, Hermann gave a better estimate [Hermann 99]:

$$|C'(t)| \leq 2 \max_{i=0,1} |c_{i+1} - c_i| \max\{\bar{w}, \frac{2}{1+\bar{w}}\} \max\{\sqrt{\frac{w_2}{w_0}}, \sqrt{\frac{w_0}{w_2}}\}, \tag{6}$$

where $\bar{w} = \frac{w_1}{\sqrt{w_0 w_2}}$.

We can get the upper bound of $|f'(t)|$ and $|g'(t)|$ from Equations (4), (5), and (6), but from the experimental 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 influences the efficiency of the algorithm. So, for rational quadratic Bézier curves, we can get the smallest possible upper bound from Equation (6).

3. Improvement in the Generating Algorithm

In order to speed up the generating time, we can use the bidirectional rendering method [Yao and Rokne 96] to draw rational Bézier curves. When x_i is known, x_{i-1} can be derived by the similar recursive relation

$$x_{i-1} = \begin{cases} x_i - 1 & \text{when} \quad \frac{\bar{u}(i-1)}{\bar{v}(i-1)} - \frac{\bar{u}(i)}{\bar{v}(i)} - \frac{z_i}{\bar{v}(i)} < -\frac{1}{2}, \\ x_i & \text{when} \quad -\frac{1}{2} \leq \frac{\bar{u}(i-1)}{\bar{v}(i-1)} - \frac{\bar{u}(i)}{\bar{v}(i)} - \frac{z_i}{\bar{v}(i)} < \frac{1}{2}, \\ x_i + 1 & \text{when} \quad \frac{\bar{u}(i-1)}{\bar{v}(i-1)} - \frac{\bar{u}(i)}{\bar{v}(i)} - \frac{z_i}{\bar{v}(i)} \geq \frac{1}{2}, \end{cases}$$

and

$$z_{i-1} = \begin{cases} z_i + x_i(\bar{v}(i-1) - \bar{v}(i)) + \bar{u}(i) - \bar{u}(i-1) - \bar{v}(i-1) \\ \qquad \text{when } x_{i-1} = x_i - 1, \\ z_i + x_i(\bar{v}(i-1) - \bar{v}(i)) + \bar{u}(i) - \bar{u}(i-1) \\ \qquad \text{when } x_{i-1} = x_i, \\ z_i + x_i(\bar{v}(i-1) - \bar{v}(i)) + \bar{u}(i) - \bar{u}(i-1) + \bar{v}(i-1) \\ \qquad \text{when } x_{i-1} = x_i + 1. \end{cases}$$

Similarly, y_{i-1}, q_{i-1} can be derived by the recurrence relation. So when we draw rational Bézier curves, we can render two endpoints $(x_0, y_0), (x_n, y_n)$ at first, then use the x_{i+1}, y_{i+1} and x_{i-1}, y_{i-1} recursive relation to get the next pixels (x_1, y_1) and (x_{n-1}, y_{n-1}) simultaneously, until two rendered points meet. Here, two rendered points meeting means that the X-value distance and Y-value distance of two rendered points should be less than or equal to 1 at the same time. This measure can reduce the loop number nearly by a half so that it improves the algorithm efficiency greatly. This judgment is suitable for lower-degree curves that do not contain loops; for higher-degree curves that contain loops, it will lead to early termination of the algorithm. One solution is that we can use the rational de Casteljau algorithm to divide higher-degree curves into some subcurves that do not include loops and then use the bidirectional generating algorithm to draw every subcurve.

In order to simplify the computation when using the recurrence relation, we can calculate $\bar{u}(i), \bar{v}(i), \bar{r}(i)$ helped by the difference method [Klassen 91, Rappoport 91]. The forward difference formulas are

$$\Delta\bar{u}(i) = \bar{u}(i+1) - \bar{u}(i) \text{ and } \Delta^{k+1}\bar{u}(i) = \Delta^k\bar{u}(i+1) - \Delta^k\bar{u}(i).$$

We notice that the order h difference of a degree h polynomial is a constant, so when we know all order differences $\Delta^k\bar{u}(i)$, we can get the differences of each order $\Delta^k\bar{u}(i+1)$ by h additions according to the formula

$$\Delta^{k-1}\bar{u}(i+1) = \Delta^{k-1}\bar{u}(i) + \Delta^k\bar{u}(i), k = 1, 2,$$

where the order 0 difference of $\bar{u}$ is the function $\bar{u}$ itself.

Similarly, there is a backward difference formula for which we use the notation

$$\nabla\bar{u}(i) = \bar{u}(i) - \bar{u}(i-1).$$

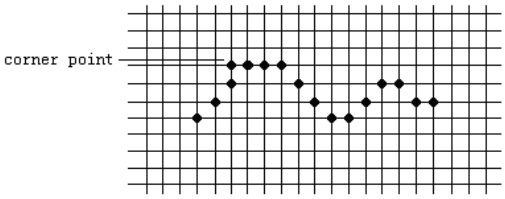

Figure 1. Corner point.

To compute for $\bar{v}(i), \bar{r}(i)$, we can use similar forward and backward difference methods.

In order to keep the curve "smooth" and avoid rendering too many redundant "corner points" (see Figure 1), we can use some variables to detect and eliminate cases where a pixel has two or more subsequent pixels in its immediate neighborhood. When we use the recurrence relation to get the pixel A (x_0,y_0), supposing the previous pixel of A is B (x_1,y_1) and the previous pixel of B is C (x_2,y_2), we can judge whether pixel B is a corner point as follows:

```
if ($|x_0 - x_2| > 1$) or ($|y_0 - y_2| > 1$) then
    Draw pixel ($x_1$, $y_1$);
    Let $x_2 = x_1$, $y_2 = y_1$; $x_1 = x_0$; $y_1 = y_0$
else
    Let $x_1 = x_0$; $y_1 = y_0$.
end if
```

By the diamond rule [Knuth 86], in general terms, a pixel centered at some point (x, y) is turned on if the curve contains a point (xx, yy) such that

$$-\frac{1}{2} < xx + yy - x - y < \frac{1}{2} \text{ and } -\frac{1}{2} < yy - xx + y - x < \frac{1}{2}.$$

So, the "corner point" may be the pixel that is turned on, but in many experiments, we find the chance that the corner point must be rendered to be very small. The haphazard effect of corner point removal can be ignored; it can keep the curve's accurate shape.

The algorithm for drawing rational quadratic Bézier curves can be written as follows:

Step 1. Choose the desired value n.

Step 2. Let the rational quadratic Bézier curve change to Equations (2) and (3).

```
Step 3. From $\bar{v}(0)x_0 = \bar{u}(0) + z_0$, $|z_0| \le |\bar{v}(0)|/2$, $\bar{v}(0)y_0 = \bar{r}(0) + q_0$,
$|q_0| \le |\bar{v}(0)|/2$, compute $x_0, y_0, z_0, q_0$; draw the first point $(x_0, y_0)$.
From $\bar{v}(n)x_n = \bar{u}(n) + z_n$, $|z_n| \le |\bar{v}(n)|/2$, $\bar{v}(n)y_n = \bar{r}(n) + q_n$,$|q_n| \le$
$|\bar{v}(n)|/2$, compute $x_n, y_n, z_n, q_n$; draw the last point $(x_n, y_n)$.

Step 4. Denoting $j$=0, $k = n$.
while two pixels $(x_j, y_j)$ and $(x_k, y_k)$ don't meet do
  Use the recurrence relation and difference formula to compute
  $x_{j+1}, z_{j+1}, y_{j+1}, q_{j+1}$.
  if pixel $(x_j, y_j)$ is not a corner point then
    draw this pixel.
  end if
  Use the recurrence relation and difference formula to compute
  $x_{k-1}, z_{k-1}, y_{k-1}, q_{k-1}$.
  if pixel $(x_k, y_k)$ is not a corner point then
    draw this pixel.
  end if
  j++; k--;
end while
if pixel $(x_j, y_j)$ is not a corner point then
  draw this pixel.
end if
if pixel $(x_k, y_k)$ is not a corner point then
  draw this pixel.
end if
```

4. Generalization of the Generating Algorithm to Higher-Degree Rational Parametric Curves

This generating algorithm for rational quadratic Bézier curves can easily be generalized to higher-degree rational Bézier curves. We first change them to the integer equations (Equations (2) and (3)), then use the recurrence relation to get every pixel point-by-point. The key point in using this algorithm is to compute the $|f'(t)|$ and $|g'(t)|$ upper-bound values. For the rational Bézier curves, the upper-bound value can be derived from Equation (4) or (5). This algorithm can also be applied to other rational parameter curves such as rational B-spline curves as long as we convert them to rational Bézier curve form. And in the algorithm, we can still apply the bidirectional strategy with the forward and backward difference methods to improve the efficiency of the generating algorithm.

In the generating algorithm for higher-degree rational parametric curves, we need to judge whether the curve includes loops that can lead to early termination of the algorithms. If the curve includes loops, we use the rational de Casteljau algorithm to divide it into some subcurves that do not contain loops. Then, we use this bidirectional generating algorithm to draw them.

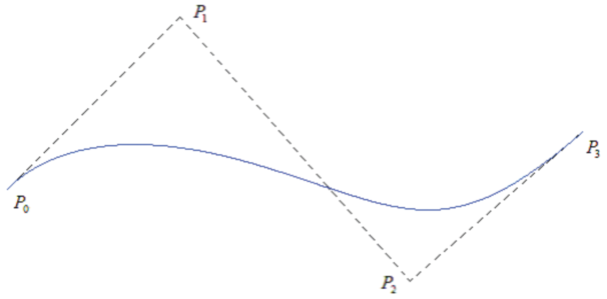

Figure 2. Rational cubic Bézier curve.

5. Experimental Results

We used this algorithm to render some rational Bézier curves in the Microsoft VC 6.0 development environment. The source code is available online at the website listed at the end of this paper. Here, the computer configuration is CPU P4/1.8 GHz, EMS memory 512 MB. Since rational quadratics can produce straight lines and circles, we have the generating time comparison between the bidirectional algorithm and Bresenham routines for those cases in Table 1. Table 2 is the generating time comparison between the bidirectional algorithm and recursive subdivision algorithm for the rational cubic Bézier curve in Figure 2.

In Table 1, there seems to be no apparent generating time difference, but in Table 2, we find that because the new algorithm uses the bidirectional

Algorithm	**Time(s)**
Bresenham algorithm for line	10
Bidirectional algorithm for line	12
Bresenham algorithm for circle	15
Bidirectional algorithm for circle	14

Table 1. Time comparison with different algorithms drawing rational quadratic Bézier curves 10,000 times.

Algorithm	**Time(s)**
Recursive subdivision algorithm	31
Bidirectional algorithm	17

Table 2. Time comparison with different algorithms drawing rational cubic Bézier curves 10,000 times.

strategy to render, which reduces the loop number nearly by a half, and uses the recurrence relation with forward and backward difference method to calculate which simplifies the computation, these measures can make this bidirectional algorithm faster than recursive subdivision algorithm.

Considering the curve shape, because we use some variables to get rid of redundant points, rational Bézier curves generated by this algorithm are "smooth" and have a reasonably accurate shape.

Acknowledgments. The author is very grateful to the anonymous referees for the useful comments and suggestions which improved this paper considerably. This paper was supported by the National Science Foundation of China (No. 60373070 and No. 60573147), Postdoctor Foundation of Shanghai (05R214129), and Zhejiang Education Foundation of China (20050786).

References

[Bresenham 77] J.E. Bresenham. "A Linear Algorithm for Incremental Digital Display of Digital Arcs." *Communications of ACM* 20:2 (1977), 100–106.

[Floater 92] M. Floater. "Derivatives of Rational Bézier Curves." *Computer Aided Geometric Design* 9 (1992), 161–174.

[Foley et al. 90] J.D. Foley, A. Van Dam, S.K. Feiner, *et al. Computer Graphics: Principles and Practice.* Reading, MA: Addison-Wesley, 1990.

[Hermann 99] T. Hermann. "On the Derivatives of Second and Third Degree Rational Bézier Curves." *Computer Aided Geometric Design* 16 (1999), 157–163.

[Huang and Zhu 01] Y.D. Huang and G.Q. Zhu. "A Fast Point-by-Point Generating Algorithm for Rational Parametric Curves." *Chinese Journal of Computers* 24:8 (2001), 809–814. In Chinese.

[Klassen 91] R. Klassen. "Integer Forward Differencing of Cubic Polynomials: Analysis and Algorithms." *ACM Transactions on Graphics* 10:2 (1991), 152–181.

[Knuth 86] Knuth, D.E. METAFONT the Program. Reading, MA: Addison-Wesley, 1986.

[Pitteway 85] M. Pitteway. "Algorithms of Conic Generation." In *Fundamental Algorithms for Computer Graphics*, NATO ASI Series, Vol. F17, pp. 219–237. Berlin: Springer Verlag, 1985.

[Rappoport 91] A. Rappoport. “Rendering Curves and Surfaces with Hybrid Subdivision and Forward Differencing.” *ACM Transactiona on Graphics* 10:4 (1991), 323–341.

[Van Aken 84] J. Van Aken. “An Efficient Ellipse-Drawing Algorithm.” *IEEE Computer Graphics & Applications* 4:9 (1984), 24–35.

[Yao and Rokne 96] C. Yao and J. Rokne. “Bi-Directional Incremental Linear Interpolation.” *Computers and Graphics* 20:2 (1996), 295–305.

[Yao and Rokne 98] C. Yao and J. Rokne. “Run-Length Slice Algorithm for the Scan-Conversion of Ellipses.” *Computers & Graphics* 22:4 (1998), 463–477.

Web Information:

Source code in C, written using the Microsoft VC 6.0 development environment, is available online at http://jgt.akpeters.com/papers/LiMa06.

Zhong Li, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, 200030 (lizhongzju@hotmail.com)

Lizhuang Ma, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, 200030 (ma-lz@cs.sjtu.edu.cn)

Received September 27, 2004; accepted in revised form August 7, 2005.