Efficient computation of Bézier curves from their Bernstein-Fourier representation

Licio Hernanes Bezerra

Universidade Federal de Santa Catarina, Departamento de Matemática, Florianópolis, SC 88040-900, Brazil

Abstract

This paper presents a new method of computation of Bézier curves of any order. This method is based on the Bernstein-Fourier representation of a Bézier curve and utilizes Fast Fourier Transforms to change from the Bernstein basis to a new one that provides efficient computation. For $2 \leq n \leq 8$, where n is the number of control points, this method is still more rapid than the VS method, which is already very fast.

Keywords: Bernstein matrix, Bézier curve, Vandermonde matrix, FFT

2000 MSC: 15B05, 65D17

1. Introduction

Let $P_0, ..., P_{n-1}$ be n distinct points of the plane. The Bézier curve of degree $n-1$ defined by these points is the set of the points $B(s)$, $s \in [0,1]$, where

$$B(s) = (x(s), y(s)) = \sum_{k=0}^{n-1} \binom{n-1}{k} s^k (1-s)^{n-1-k} P_k.$$

Email address: licio@mtm.ufsc.br (Licio Hernanes Bezerra)

The polynomials $b_{k,n-1}(s) = \binom{n-1}{k} s^k (1-s)^{n-1-k}$ are known as Bernstein basis polynomials of degree n-1. Note that a method to evaluate a Bézier curve is in fact a method to evaluate a polynomial written in the Bernstein basis. Bézier curves play a fundamental role in Computed-Aided Geometric Design area, where the de Casteljau method is the most common method for this evaluation [6]. Numerically speaking, de Casteljau method is very stable. However, it demands $\mathcal{O}(n^2)$ algebraic operations for each s. Recently, several alternative methods have been developed which require fewer operations. Computationally speaking these methods are founded on a change of polynomial basis followed by an efficient algorithm to evaluate the polynomial in the new form. For instance, we have the VS method [8] as well as several methods that have used generalized Ball bases [4, 5, 7, 9], which have arisen after A. Ball had considered the set $\{(1-s)^2, 2s(1-s)^2, 2s^2(1-s), s^2\}$ as a basis for the cubic polynomials [1]. Also there are examples of matrix-oriented methods, where changes of basis involve some matrix theory [2, 3]. If we only consider the time consumption, from [2] we arrive at the conclusion that the VS method [8] is the most efficient when compared with the de Casteljau method, the Wang-Ball method [9], the Said-Ball method [9], the most efficient Pascal matrix method described in [3] and the Hankel matrix method explained in [2]. Here we introduce a new method to evaluate a Bezier curve of order $n-1$ which is even faster. From Fast Fourier Transforms (FFT), the curve becomes as follows:

$$B(s) = \sum_{k=0}^{n-1} Q_k u_k(s),$$

where $u_k(s) = (1 + s(\omega^k - 1))^{n-1}$ for $k = 0 : n-1$, and $\omega = e^{\frac{-2\pi i}{n}}$. Since $B(s)$ is a curve in $\mathbb{R}^2$, we should have $Q_k = \overline{Q_{n-k}}$ for $k \neq 0, n/2 - 1$, if n is even (otherwise, $k \neq 0$). Its performance is discussed here by comparing it with the de Casteljau and the VS methods.

2. Vandermonde matrix as a conversion matrix

Consider a Bézier curve B of degree $n-1$ defined by n given points $P_0 = (x_0, y_0)$, $P_1 = (x_1, y_1)$, ..., $P_{n-1} = (x_{n-1}, y_{n-1})$ in $\mathbb{R}^2$. Let $B_n(s)$ be the $n \times n$ lower triangular matrix such that $(B_n)_{ij}(s) = b_{j-1,i-1}(s)$ for each $n \geq i \geq j \geq 1$, that is,

$$B_n(s) = \begin{pmatrix} 1 & 0 & 0 & ... & 0 \\ 1-s & \binom{1}{1}s & 0 & ... & 0 \\ (1-s)^2 & \binom{2}{1}(1-s)s & \binom{2}{2}s^2 & ... & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ (1-s)^{n-1} & \binom{n-1}{1}(1-s)^{n-2}s & \binom{n-1}{2}(1-s)^{n-3}s^2 & ... & \binom{n-1}{n-1}s^{n-1} \end{pmatrix}.$$

$B_n(s)$ is called a Bernstein matrix. Recall that the Bernstein polynomials $b_{0,n-1}(s), ..., b_{n-1,n-1}(s)$ form a basis of $\mathbb{P}_{n-1}(\mathbb{R})$, the vector space of the real polynomial functions of degree less than or equal to $(n-1)$. Let Z be the $n \times 2$ matrix defined by $Z(k,1) = x_{k-1}$ and $Z(k,2) = y_{k-1}$ for all $k \in \{1, ..., n\}$. Let $e_n = (0 \, ... \, 0 \, 1)^T \in \mathbb{C}^n$. Thus,

$$B(s) = e_n^T B_n(s) Z.$$

Given $(\alpha_0, ..., \alpha_{n-1}) \in \mathbb{C}^n$, let $W = W(\alpha_0, ..., \alpha_{n-1})$ the $n \times n$ Vandermonde matrix defined by

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_0 & \alpha_1 & \cdots & \alpha_{n-1} \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_0^{n-1} & \alpha_1^{n-1} & \cdots & \alpha_{n-1}^{n-1} \end{pmatrix}.$$

Suppose that $\alpha_i \neq \alpha_j$ for $0 \leq i < j \leq n-1$. Then W is an invertible matrix. Therefore, we have

$$B(s) = e_n^T B_n(s) W \left(W^{-1} Z\right).$$

It is not difficult to see that

$$e_n^T B_n(s) W = \left((1 - s + \alpha_0 s)^{n-1} \cdots (1 - s + \alpha_{n-1} s)^{n-1}\right).$$

A good choice for W is the Fourier matrix: $W = W(1, \omega, ..., \omega^{n-1})$, where $\omega = e^{\frac{-2\pi i}{n}}$. Hence, the operation $W^{-1}Z$ can be done in an accurate way and in $\mathcal{O}(n \log n)$ arithmetic operations. In MATLAB the matrix $U = W^{-1}Z$ is easily computed: $U = ifft(Z)$.

Now, let $Q_0 = (r_0, s_0)$, $Q_1 = (r_1, s_1)$, ..., $Q_{n-1} = (r_{n-1}, s_{n-1})$, where $r_k = U(k+1, 1)$ and $s_k = U(k+1, 2)$ for all $k \in \{0, ..., n-1\}$. Hence, we can conclude that

$$B(s) = \sum_{k=0}^{n-1} Q_k (1 + s(\omega^k - 1))^{n-1}, \qquad s \in [0, 1]. \tag{1}$$

3. Numerical Results

Let $P_0 = (x_0, y_0)$, ..., $P_{n-1} = (x_{n-1}, y_{n-1})$ be n distinct points.

The de Casteljau algorithm evaluates polynomials in the form

$$\sum_{k=0}^{n-1} \binom{n-1}{k} P_k s^k (1-s)^{n-k-1}.$$

It demands exactly $n(n-1)$ products and $2n(n-1)$ additions for each $s \in (0,1)$ as we can see in the following lines of MATLAB code:

```
function [C1 C2] = casteljau(Z,s)
%CASTELJAU(Z,s) De Casteljau method.
%   CASTELJAU(Z,s) is the Bezier curve defined by Z at t=s
n = size(Z,1);
b1 = x;
b2 = y;
for k = 2:n
    for r = n:-1:k
        b1(r) = b1(r-1) + s * ( b1(r) - b1(r-1) );
        b2(r) = b2(r-1) + s * ( b2(r) - b2(r-1) );
    end
end
C1 = b1(n);
C2 = b2(n);
```

The VS algorithm evaluates polynomials in the form

$$\sum_{k=0}^{n-1} V_k s^k (1-s)^{n-k-1},$$

where $V_k = \binom{n-1}{k} P_k$. This algorithm consists of $2n-1$ products and $2n-1$ additions for each $s \in (0,1)$ (see the algorithm in [5]).

Let Z be the $n \times 2$ matrix such that $Z(k,:) = (x_{k-1}\, y_{k-1})$. Then, our algorithm first calculates $U = ifft(Z)$, which requires $\mathcal{O}(n \log n)$ operations, and afterwards it evaluates $B(s)$ according to Equation 1. Since the nth power of a number can be found in $\mathcal{O}(\log n)$ operations, all computation requires $\mathcal{O}(n \log n)$ operations. However, we can reduce the time of computation of Equation 1 as follows. Suppose, for instance, that $s = 0 : 1/128 : 1$ is a partition of the interval [0,1] in 128 equal subintervals, that is, $s(j) = (j-1)/128$ for $j = 1 : 129$. Let $t = 1 : -1 : 1/2 + 1/128$, that is, the elements of s that are greater than $1/2$ in reverse order. Consider an integer k such that $0 < k < n/2$. Since $\omega^{n-k} = \overline{\omega^k}$ we have for all $s < 1/2$ that

$$\left(\omega^k s + (1-s)\right)^{n-1} = \omega^{kn-k}\left(s + (1-s)/\omega^k\right)^{n-1} =$$

$$= \overline{\omega^k}\left(s + \overline{\omega^k}(1-s)\right)^{n-1} = \overline{\omega^k}\left(\overline{\omega^k}t + (1-t)\right)^{n-1}.$$

Therefore, since $Q_{n-k} = \overline{Q_k}$ for $0 < k < n/2$, the calculation of $B(s)$ for $s = 0 : 1$ depends basically on the computation of $\left(\omega^k s + (1-s)\right)^{n-1}$ for $0 \leq s < 1/2$ and for $0 < k < n/2$, which yields a significant reduction of the computational time.

Without loss of generality, we have used the MATLAB function *rand* in order to generate n test control points: $A = rand(n, 2)$. Hence, the coordinates of the control points are all positive, and also less than or equal to 1. In Table 1 we can compare the average times to compute Bézier curves of several orders by using our method, VS method and de Casteljau method. The results have been averaged over 1000 runs. The experiments have been run in a MacBook Pro 5.5 with a 2.26 GHz Intel Core 2 Duo processor under a 64-bit MATLAB, version R2011b.

Table 1: Mean run time of computation in seconds of 129 points of a Bézier curve of degree N-1 by three different methods: our Algorithm (F), VS (V), and de Casteljau (C). *NC* means non-convergence.

N	Time (F)	Time (V)	Time (C)
2	6.7169e-05	1.8267e-04	3.5330e-04
3	1.1026e-04	1.9017e-04	3.8008e-04
4	1.5732e-04	2.4685e-04	4.4755e-04
5	1.7473e-04	2.5436e-04	5.1001e-04
6	2.1299e-04	2.6647e-04	5.8470e-04
7	2.3539e-04	2.7248e-04	6.5651e-04
8	2.7500e-04	2.8284e-04	7.5417e-04
9	2.9945e-04	2.9454e-04	8.4648e-04
16	5.3785e-04	3.6302e-04	0.0018
32	0.0011	5.2298e-04	0.0059
64	0.0024	8.5407e-04	0.0215
128	0.0050	0.0017	0.0874
256	0.0106	0.0039	0.3428
512	0.0229	0.0123	1.3698
1024	0.0519	0.0400	5.4491
2048	0.1171	0.1391 (NC)	21.9483

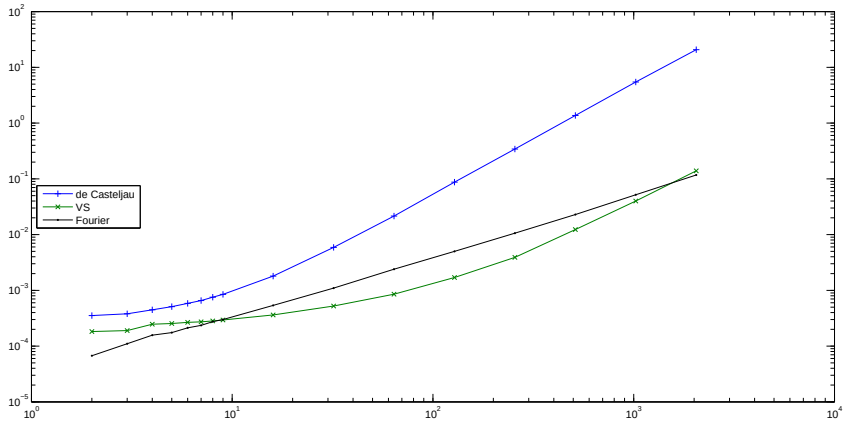

Figure 1: Fourier versus VS versus de Casteljau

4. Conclusions

The data in Table 1 can also be seen in Figure 1, where they are represented in logarithmic scale. There we see that our method works better than VS for $n = 2 : 8$ and better again for $n = 2048$. We have done tests for various $n > 2048$, and so far our method has worked better than the VS method. However, the VS method has not converged for $n > 1024$. This is certainly due to the fact that some binomial coefficients needed to convert P_k into V_k are greater than 10^{15}. MATLAB gives a warning message saying that those coefficients are accurate to 15 digits. On the other hand, our method is as accurate as the de Casteljau method. For instance, if F and C are two sets of 129 points obtained from the computation of a Bézier curve of order 511 by our method and the de Casteljau method respectively, the norm of the difference between F and C is about 10^{-14}. For future work we intend to examine shape preserving properties and other properties associated with

these bases formed by pairs of complex conjugate polynomials (see [2]).

[1] A. A. Ball, CONSURF, Part one: Introduction to conic lofting tile, Comput. Aided Design 6 (1974) 243-249.

[2] L. H. Bezerra, Vandermonde Factorizations of a Regular Hankel Matrix and Their Application to the Computation of Bézier Curves, SIAM J. Matrix Anal. & Appl. 33 (2012) 411-432.

[3] L. H. Bezerra and L. K. Sacht, On computing Bézier curves by Pascal matrix methods, Appl. Math. Comput. 217 (2011) 10118-10128.

[4] J. Delgado and J. M. Peña, A shape preserving representation with an evaluation algorithm of linear complexity, Comput. Aided Geom. D. 20 (2003) 1–10.

[5] J. Delgado and J. M. Peña, On efficient algorithms for polynomial evaluation in CAGD, Monogr. Semin. Mat. García de Galdeano 31 (2004) 111–120.

[6] G. Farin, Curves and Surfaces for Computed Aided Geometric Design: A Practical Guide, 3rd ed.. Academic Press, New York, 1993.

[7] H. B. Said, Generalized Ball curve and its recursive algorithm, ACM Trans. Graph. 8 (1989) 360–371.

[8] L. L. Schumaker and W. Volk, Efficient evaluation of multivariate polynomials, Comput. Aided Geom. D. 3 (1986) 149–154.

[9] H. Shi-Min, W. Guo-Zhao and J. Tong-Guang, Properties of two types of generalized Ball curves, Comput. Aided Design 28 (1996) 125–133.