On Splines and Their Use in Computer Graphics and Generating Curves

Simon Abrelat

002129–0004

Math HL IA

May 2019

Words:

Abstract

Splines are used everywhere around us, from fonts to animations. They are the way that computers deal with curves and because of that have a large range of possible uses. Since they are so useful, there are also a lot of ways to generate these functions and lots of formats they come in. In general, they are fairly computationally cheap which makes them so useful and can be extended in a multitude of ways. In this paper, the goal would be to use splines to smoothly interpolate values either for control or for path generation.

Contents

1 Introduction

Splines are not a specific equation, they are more a class of equations that have similar properties. They are simply piecewise polynomials, and are often parameterized so that they could “loop over themselves” and break the vertical line test. One of the interesting aspects of splines is that they keep this property even at low degrees. The word spline is derived from wooden splines that curve given some constraints and are often used in things like braces for ships. The major benefit of splines is how they are suited for computers. Almost all curves on a computer from fonts to animation paths are described quickly and accurately by Bèzier curves and other such splines. This means that these splines are around us all of the time and are almost invisible if you do not know where to look. There are many different kinds of splines with different methods of formulation. However, this paper will mostly be focused on Hermite and Bèzier curves.

2 Bèzier Curves

2.1 Theory

Bèzier curves can be made with a linear combination of Bernstein polynomials (Casselman, 1984). Bernstein polynomials were first used in the proof of the Stone-Weierstrass theorem which states that every continuous function defined on a closed interval $[a, b]$ can be approximated as closely as desired by a polynomial (Pinkus, 2000). A Bernstein polynomial (2) is defined by a linear combination of Bernstein basis functions (1).

Bernstein Polynomials:

$$b_{\nu,n} = \binom{n}{\nu} x^{\nu}(1-x)^{\nu-n} \tag{1}$$

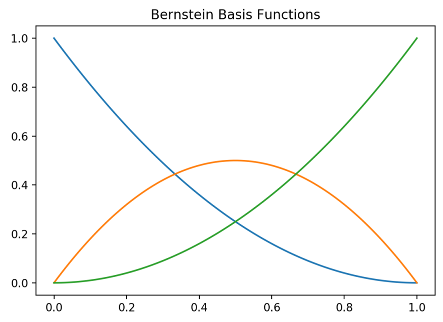
$$B_n(x) = \sum_{\nu=0}^{n} \beta_{\nu} b_{\nu,n}(x) \tag{2}$$

$$\lim_{n\to\infty} B_n(f) = f \tag{3}$$

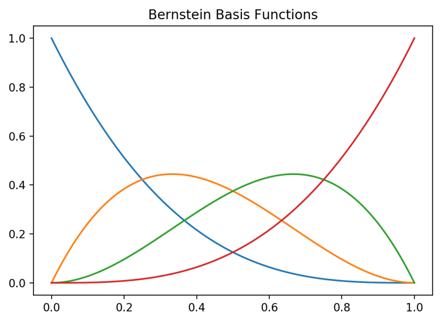
n: is the degree of the Bernstein function

ν: is number of the $n+1$ equations of a function of degree n

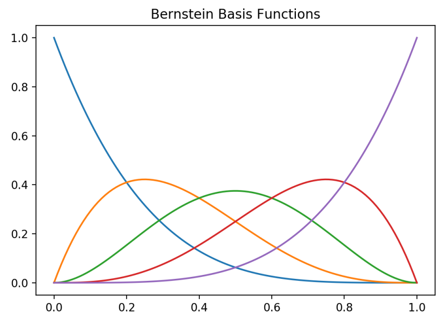
Figure 1: Basis functions for different degrees, Listing 1



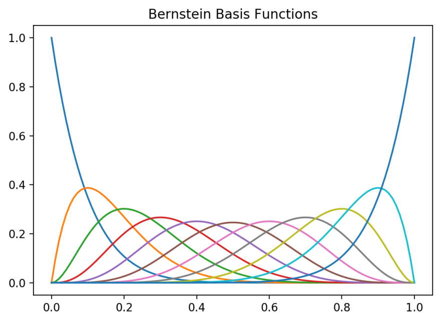
(a) $n = 2$



(b) $n = 3$



(a) $n = 4$



(b) $n = 10$

Bernstein functions are only approximations and to perfectly match the original function you would need an infinite degree Bernstein polynomials similar to Taylor series (3). The algorithm you use to generate a Bézier curve given your control points is called De Castlejau's algorithm. It is a recursive algorithm (4) that takes your control points and uses them for interpolation. This method is a little more complicated than the explicit form (6) which is a summation of the control points times the basis functions of a given degree n. Then to generate any possible curve, parameterize the X and Y axes and fun De Castlejau's algorithm on the scalar x and y quantities.

$$\beta_n^{(0)} = \beta_\nu, i = 0, \ldots, n \tag{4}$$
$$\beta_i^{(j)} = \beta_\nu^{(j-1)}(1 - t_0) + \beta_\nu^{(j-1)} t_0 \tag{5}$$

$j = 1, \ldots, n$

$\nu = 0, \ldots, n - j$

$$B(t) = \sum_{\nu=0}^{n} \beta_\nu b_{\nu,n}(t) \tag{6}$$

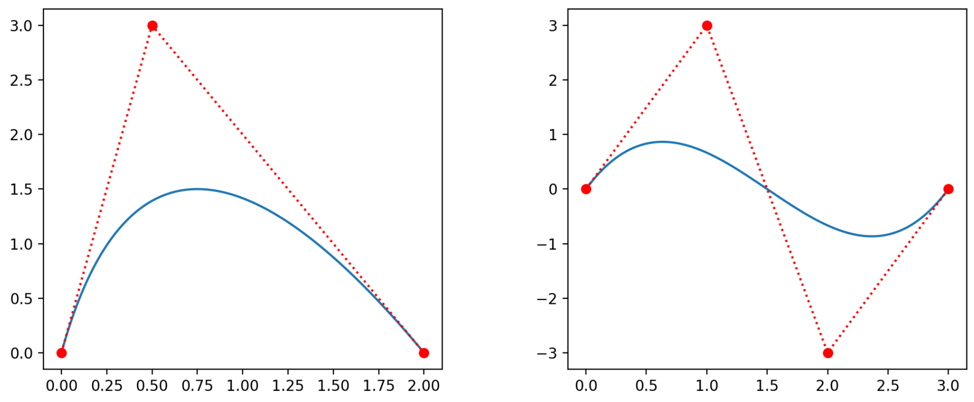
β_ν: is the ith control point

$b_{\nu,n}(t)$: is the Bernstein basis function for the number and degree

2.2 Application

There are a practically infinite number of uses for Bèzier curves. They and other similar methods are used to display all curves in computers which means that anything make industrially with a curve has a Bèzier curve or other spline involved. However, the hidden ubiquitous use of Bèzier curves in particular are fonts. All PostScript fonts, the basis of PDFs, use cubic Bèzier curves (a) and TrueType fonts, abbreviated to .ttf, use quadratic curves (b).

Figure 3: Cubic and Quadratic Curves, Listing 2



(a) Cubic curve, 3 control points (b) Quadratic curve, 4 control points

In fact, most of the computer generated graphics like advertisements contain Bèzier curves. The popular software used for such things is Adobe Illustrator®which has tools for generating curves with specific parameters being 2 anchor and 2 control points which is exactly the quadratic Bèzier curve.

The other use for Bèzier curves are in optimization problems. Since they are used to approximate a curve and the control point can be related to the derivative of a function, they are very useful for optimization problems. They can be used to minimize path lengths (Jusko, 2016) and can even have restrictions such as a maximum acceleration or velocity (G. Jolly et al., 2009). The acceleration and velocity limits are caused by the control points which can be controlled since they are tangential to their anchor points meaning the angle is related to the derivative of the function and the magnitude of the control point is related to the acceleration or multiple control points. These values can even be used to model resistance in a parameterized manner so yet another use of Bèzier curves would be the generation of an optimal airfoil (Rogalsky et al., 2000).

References

Casselman, B. (1984). From bézier to bernstein. *American Mathematical Society*, 39(6):1087–1097.

G. Jolly, K., Sreerama Kumar, R., and Vijayakumar, R. (2009). A bezier curve based path planning in a multi-agent robot soccer system without violating the acceleration limits. 57:23–33.

Jusko, T. (2016). Scalable trajectory optimization based on bézier curves. Deutscher Luft- und Raumfahrtkongress.

Pinkus, A. (2000). Weierstrass and approximation theory. *Journal of Approximation Theory*, 107:1–66.

Rogalsky, T., W. Derksen, R., N, R., and Kocabiyik, S. (2000). Differential evolution in aerodynamic optimization. *Canadian Aeronautics and Space Journal*.

3 Appendix

Listing 1: Bernstein Basis Functions

```python
from numpy import linspace as lin  # linespace
from scipy.special import binom  # binomial coefficient
import matplotlib.pyplot as plt  # plotting
from math import pow # power function

x = lin(0, 1, 100)  # Input as 100 floats between 0 and 1
num: int = 10  # The degree

# returns the value from the Bernstein basis function at the point x
def bbf(i: int, n: int, x: float) -> float:
    return binom(n, i) * pow(x, i) * pow((1 - x), (n-i))

# Iterates through all of the functions for the degree
for c in range(num + 1):
    # Plots all of the functions
    plt.plot(x, [bbf(c, num, i) for i in x])

plt.title("Bernstein Basis Functions")  # Title
plt.show()  # displays graphs
```

Listing 2: Bèzier Curves

```python
from numpy import linspace as lin  # linespace
from scipy.special import binom  # binomial coefficient
import matplotlib.pyplot as plt  # plotting
from math import pow # power function

cpx: [float] = [0.0, 0.5, 2.0]
cpy: [float] = [0.0, 3.0, 0.0]

t = lin(0, 1, 100)  # Input as 100 floats between 0 and 1
num: int = 2  # The degree 0 indexed

# returns the value from the Bernstein basis function at the point x
def bbf(i: int, n: int, x: float) -> float:
    return binom(n, i) * pow(x, i) * pow((1 - x), (n-i))

# Generalized De Casteljau's Explicit formula
def f(a: [float], t: float) -> float:
    ret = 0
    for i in range(num + 1):
        ret += a[i] * bbf(i, num, t)
    return ret

def x(t: float) -> float: return f(cpx, t) # implemented on X's
    control points
def y(t: float) -> float: return f(cpy, t) # implemented on Y's
    control points
```

```

plt.plot([x(i) for i in t], [y(i) for i in t]) # plots X and Y
plt.plot(cpx, cpy, 'ro') # plots the control points
plt.plot(cpx, cpy, 'r:') # plots lines between control points
plt.show()
```