Bézier Curves and De Casteljau's Algorithm

Rhiannon Titcomb

December 11, 2014

The Bézier curve was founded by Pierre Bézier in 1962 and can be described in the most basic way as a kind of parametric curve between two points. Its 'curvature' is defined by its unique 'control points', as illustrated in Figure 1 [1]. To keep things from getting too complex, throughout this presentation I will refer to cubic Bézier curves unless stated otherwise. However note that Bézier curves can be of any degree.

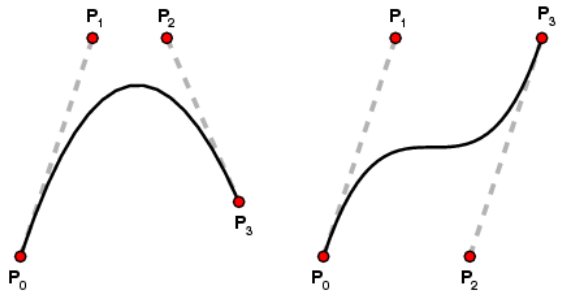

Figure 1: Here are two cubic Bézier curves, both with control points P_0, P_1, P_2, and P_3 but with different curvatures.

Bézier curves are the most commonly used curves in computer graphics and computer design, simply because they are the most intuitive curves to manipulate whilst remaining simple enough to compute quickly. An example of a Bézier path is given in Figure 2, which I plotted using the inbuilt Bézier path function of Sage.

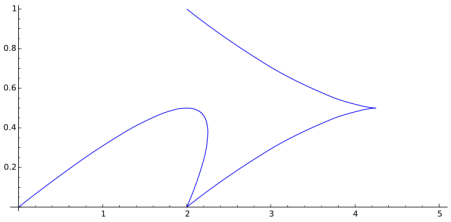

Figure 2: This is a plot of a Bézier Path.

1 The Bernstein Polynomial

The Bézier curve is based on the Bernstein polynomial - a polynomial which is a combination of basis polynomials. Due to limited space, I will not go into detail about how this is derived. For more information on this please refer to [2] and [3].

2 Drawing a Bézier Curve

2.1 De Casteljau's Algorithm

To construct a Bézier curve we use one important property: any Bézier curve can be split into multiple parts to trace out the curve as straight lines [1]. Computers do this using De Casteljau's algorithm. In Section 3 we'll derive a formula to approximate a Bézier curve.

2.2 Recursive Definition of a Bézier Curve

Let us label the control points of a $n-1$ degree Bézier curve as P_0, P_1, P_2, ... ,P_n. We can then describe the curve recursively as

$$B(t) = (1-t)B_{P_0,P_1,P_2,P_{(n-1)}}(t) + tB_{P_1,P_2,P_n}(t)$$

This looks very confusing, but we can re-write it using the binomial theorem [4] to give

$$\sum_{k=1}^{n} \binom{n}{k} (1-t)^{n-k} t^k P_{k-1}$$

For example, we can write the equation of a cubic Bézier curve as $B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3$ [5].

3 Deriving De Casteljau's Algorithm

In Section 2.2 we found a recursive definition for a Bézier curve. This can be used, in a very complex way, to create De Casteljau's Algorithm. De Casteljau's Algorithm takes the control points and finds the midpoints along each line, then joins these midpoints. After that, it takes the midpoints along the newly drawn lines and finds the midpoints again, then draws a line connecting these. By doing this until we are down to only one point, we can approximate the Bézier curve.

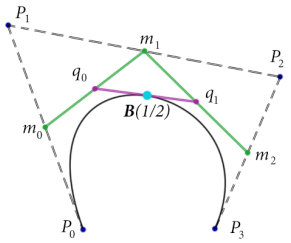

Figure 3: This image shows the midpoints of each line segment in order to give an approximation of the Bézier curve [5].

Note that the higher the degree of the Bézier curve, the more accurate the algorithm value will be, as it will use more line segments. The following code creates a function that will find the midpoints (to save space I have omitted some of the code - the full function can be found at `https://cloud.sagemath.com/projects/001c8810-452b-4b5f-978e-84edfbe12457/files/Individual%20Coursework/DeCasteljau.sagews`).

```
def Casteljau(a = [0,0], b = [0,5], c = [5,5], d = [5,0]):
    #these values for a, b, c, and d can be adjusted to whatever points needed

    #each point is given in the form (x,y)
```

```
a_x = a[0]
a_y = a[1]
print "P_0 = ", (float(a_x), float(a_y))
P_0 = (a_x, a_y)

b_x = b[0]
b_y = b[1]
print "P_1 = ", (float(b_x), float(b_y))
P_1 = (b_x, b_y)

#Points P_2 and P_3 are defined in a similar way

#The midpoint of P_0 and P_1 is M1
M1_x = (a_x + b_x)/2
M1_y = (a_y + b_y)/2
print "M_1 = ", (float(M1_x), float(M1_y))
M_1 = (M1_x, M1_y)

#Points M_2 and M_3 are defined in a similar way

#The midpoint of M_1 and M_2 is Q1
Q1_x = (M1_x + M2_x)/2
Q1_y = (M1_y + M2_y)/2
print "Q_1 = ", (float(Q1_x), float(Q1_y))
Q_1 = (Q1_x, Q1_y)

#Point Q_2 is defined in a similar way

#The midpoint of Q_1 and Q_2 is B1
B1_x = (Q1_x + Q2_x)/2
B1_y = (Q1_y + Q2_y)/2
print "B_1 = ", (float(B1_x), float(B1_y))
B_1 = (B1_x, B1_y)
```

These midpoints can then be used to plot an approximation of the curve in Sage, in a similar style to Figure 3.

References

[1] Casselman, B., "from bézier to bernstein", 2014, http://www.ams.org/samplings/feature-column/fcarc-bezier [Accessed 8 December 2014].

[2] Joy, K. I., "bernstein polynomials", 1996, http://graphics.idav.ucdavis.edu/education/CAGDNotes/Bernstein-Polynomials.pdf [Accessed 8 December 2014].

[3] Weisstein, E. W., "bernstein polynomial", 2014, http://mathworld.wolfram.com/BernsteinPolynomial.html [Accessed 9 December 2014].

[4] Prautzsch, H., Boehm, W., and Paluszny, M., *"Bézier and B-Spline Techniques"*, Springer, 2002.

[5] Kun, J., "bezier curves and picasso", 2013, http://jeremykun.com/2013/05/11/bezier-curves-and-picasso/ [Accessed 11 December 2014].