Bézier Curves

Joel Barrett
University of Abertay Dundee
2012

1 Introduction

Bézier curves inherit their name from Pierre Bézier, a French engineer who popularised their use during his tenure at Renault (Bézier 1972). As with a number of important scientific discoveries, Bézier curves obey Stigler's law of eponymy in that they aren't named after their original discoverer, Paul de Casteljau, who was under an NDA at the time that he discovered them, so wasn't able to publish his work.

A Bézier curve is defined by a set of control points, known collectively as the control polygon – or convex hull – where the first and last points are the end points of the curve, and the middle points manipulate the shape of the curve by acting as attractors. If we were to connect these points with lines they would form a pen containing the Bézier curve.

2 Mathematics

Given the control points $P_0, P_1, \ldots P_n$, the nth degree Bézier curve can be given by:

$$\boldsymbol{B}_n(t) = \sum_{r=0}^{n} P_r \, \boldsymbol{B}_{r,n}(t), \qquad t \in [0,1]$$

Where $\boldsymbol{B}_{r,n}(t)$ is a Bernstein polynomial equivalent to:

$$\binom{n}{r} t^r (1-t)^{n-r} \equiv \frac{n!}{r!\,(n-r)!} t^r (1-r)^{n-r}$$

2.1 Cubic Bézier Curves

In the context of this application, we are interested in the case $n = 3$ of the above equation – i.e. cubic Bézier curves with the parametric equation:

$$\boldsymbol{B}(t) = (1-t)^3 \boldsymbol{P}_0 + 3t(1-t)^2 \boldsymbol{P}_1 + 3t^2(1-t)\boldsymbol{P}_2 + t^3 \boldsymbol{P}_3$$

2.2 Piecewise Bézier Curves

Bézier curves become computationally expensive as their degree increases above cubic, so it is preferable to join a series of lower degree curves together should more control points be desired. In order to obtain a smooth path for an object to follow, C1 continuity is required. C0 continuity is achieved when $b_3 = c_0$, i.e. the last point of the first curve shares its coordinates with the first point of the adjoining curve. For C1 continuity to exist, the curves must also share a common tangent; which is to say that $B'(1) = C'(0)$.

2.3 Frenet-Serret Equations

To orient objects moving around a curve, the Frenet-Serret formulas can be used:

$$\frac{d\boldsymbol{T}}{ds} = \rho \boldsymbol{N}, \qquad \frac{d\boldsymbol{N}}{ds} = -\rho \boldsymbol{T} + \tau \boldsymbol{B}, \qquad \frac{d\boldsymbol{B}}{ds} = -\tau \boldsymbol{N}$$

Where d/ds is the derivative with respect to arc length, ρ is curvature, τ is the torsion of the curve and:

- $\boldsymbol{T}$ is the unit vector tangent to the curve, pointing in the direction of motion.
- $\boldsymbol{N}$ is the derivative of $\boldsymbol{T}$ with respect to the arc length parameter of the curve, divided by its length.
- $\boldsymbol{B}$ is the cross product of $\boldsymbol{T}$ and $\boldsymbol{N}$.

Collectively, these are known as the TNB or Frenet-Serret frame.

3 Numerical Analysis

In order to render a Bézier curve using current rasterisation methods, we must segment it into a series of connected lines known as a polyline. To compute the vertices comprising this polyline, we must evaluate the curve at different values of the parameter t. This section will describe different means by which to do this, with a view to selecting the most optimal.

3.1 Cubic Interpolation

This is the most intuitive method for computing a point on the curve at parameter t given the cubic Bézier equation – however, it is also the slowest, and basically the brute-force approach to rendering a curve as it's just a direct translation of the given equation without consideration for computational efficiency.

3.2 De Casteljau's Method

An improvement over the brute-force approach, de Casteljau's method makes use of the control polygon in order to evaluate points on the enclosed curve at a time complexity of $O(n^2)$. The following recurrence relation exists for a cubic Bézier curve:

$$b_i^{(j)} = (1-t)b_i^{(j-1)} + t b_{i+1}^{(j-1)}, \quad (j = 1,2,3 \; and \; i = 0,1,\ldots,3-j)$$

Which is in fact a linear interpolation over multiple iterations in order to compute the point $B(t)$. The collective results from these iterations can be visualised as a table.

De Casteljau's method isn't the fastest means of computing single points on a curve, but it is the most stable, and the results of the table can also be used to compute tangents and split a curve; the latter of which is utilised in this program to add a new curve to the track. If a single table is used to compute points, tangents and to split the curve, de Casteljau's method becomes more interesting.

3.3 Horner's Scheme

Given the polynomial:

$$p(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_nx^n$$

Horner's scheme is concerned with rearrangement of the terms to the following form:

$$p(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + a_nx)\ldots))$$

One can observe that multiplication operations are kept to a minimum by grouping terms in parentheses. This of course is favourable for computation – but first, the cubic Bézier equation must be converted to monomial form, yielding:

$$\boldsymbol{B}(t) = b_0 + (3b_1 - 3b_0)t + (3b_0 - 6b_1 + 3b_2)t^2 + (-b_0 + 3b_1 - 3b_2 + b_3)t^3$$

In the application, this equation is further rearranged to reduce operations, with the most important observation being the fact that the coefficients only need to be computed at initialisation of the curve class, and when a control point is translated. All-in-all, Horner's method involves 3 vector additions and multiplications to compute a single point on the curve.

Horner's method is optimal both in the number of additions and multiplications involved; however, the expression tree produced is linear in disposition, and as such is unsuited to parallelisation. Worthy of note is Estrin's scheme (Knuth 1997), which is almost as efficient as Horner in regard to operations, but also has a wider expression tree suited to modern architectures. However, it only proves worthwhile for evaluation of higher degree polynomials (circa 7th degree and above), and as such, isn't beneficial for evaluation of Bézier curves. Another alternative is the E-method, but this seems to be targeted at fixed-point hardware, so also wasn't investigated further.

3.4 Forward Differences

Whilst Horner's method proves optimal in this scenario for one-off computations of points on the curve, there is still room for improvement when it comes to computing a polyline. The main principle of forward differences is to find some value $\boldsymbol{dB}$ such that $\boldsymbol{B}(t + \Delta t) = \boldsymbol{B}(t) + \boldsymbol{dB}$ for any value of t. If this can be found, then it is sufficient to calculate $\boldsymbol{B}(0)$ and take forward differences in order to find, in general:

$$\boldsymbol{B}(i\Delta t) = \boldsymbol{B}\big((i-1)\Delta t\big) + \boldsymbol{dB} = \boldsymbol{B}(0) + i\boldsymbol{dB}$$

We find the value $\boldsymbol{dB}$ through Taylor series - i.e.:

$$f(t + \Delta t) = f(t) + f'(t)\Delta t + \frac{f''(t)\Delta^2 t}{2!} + \frac{f'''(t)\Delta^3 t}{3!} + \cdots$$

Although in this case we limit the sum to cubic Bézier curves. Upon evaluating Taylor series for the coefficients given above in the monomial form of a Bézier curve, we find the forward differences to be:

$$f_i = at_i^3 + bt_i^2 + ct_i + d$$
$$\Delta f_i = (3a\varepsilon)t_i^2 + (3a\varepsilon^2 + 2b\varepsilon)t_i + (a\varepsilon^3 + b\varepsilon^2 + c\varepsilon)$$
$$\Delta^2 f_i = (6a\varepsilon^2)t_i + (6a\varepsilon^3 + 2b\varepsilon^2)$$
$$\Delta^3 f_i = 6a\varepsilon^3$$

Once these are computed, a point on the curve can be found with just 3 vector additions and no multiplications. The computation of forward differences is further optimised in the application by reversing the order in which they appear, as it can be observed that past results are reusable in the future.

3.5 Arc Length

The arc length of a parametric curve between t_0 and t_1 is given by the definite integral:

$$l = \int_{t_0}^{t_1} \sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2}\, dt$$

There are numerous methods available in which to approximate this integral, such as Trapezium rule, Simpson's rule and Riemann sums. If accuracy is a consideration, a more sophisticated approach such as Gaussian or Clenshaw-Curtis quadrature may be desirable. In the context of this program, speed is the main factor, and as such, a more intuitive geometric stance to arc length computation was taken.

An arc length approximation by summing the distances between each vertex was implemented, but the final method used was proposed by Jens Gravesen (1997), and involves computing half of the distance around the control polygon multiplied by the chord length – i.e. the distance between the end points. While this is inaccurate, it is sufficient in the context of the program, where arc length is used to determine where a new curve should be added to the circuit.

4 Physics

To determine the speed of an object moving around the curve we take the sum of kinetic and potential energy in the system (i.e. mechanical energy) and the start and end of a specified time interval. The energy equation of a system is given by:

$$E = \frac{1}{2}mv^2 + mgh$$

Which is $E = KE + PE$. We assume no air resistance (drag) or friction – thus allowing perpetual motion to be realised – and take the level of PE to be the plane. Mass is ignored in the application – i.e. just set to one so that the m's cancel – and Earth's gravity is used ($9.81 m/s^2$), leaving velocity as the unknown.

By the principle of conservation of energy we know that for a system acted on by conservative forces only, the total mechanical energy remains constant. Using this property, we can then take the energy at the end of a time interval to find the value of v. We then substitute v into the equation $distance = speed \times time$ along with delta time in order to determine the distance an object should travel along the curve.

5 Conclusion

As far as mathematical operations are concerned, the methods employed by the application are efficient. However, there is room for further optimisation, such as in storing the polylines in display lists or vertex buffers, and utilising the GPU to compute curves

via geometry or tessellation shaders. Also, forward differences could be used to compute an array of tangents at each vertex, so that they aren't evaluated "on the fly". This could be implemented in conjunction with moving objects around the curve based on vertex positions, rather than computing an independent value of t, which would mean that smoothness of object movement around the curve would be dependent on resolution.

Taking the concept of precomputation further: as there are no external forces acting on the physics of entity movement, the speed of object travel at each vertex of the curve could also be computed upon initialisation, and simply retrieved at each iteration of the game loop. This would shift computation to initialisation and picking, rather than object state updates.

References

BÉZIER, P.E. 1972. Emploi des machines à commande numérique. (Translation: Numerical control, mathematics and applications.) London: John Wiley and Sons.

GRAVESEN, J. 1997. Adaptive subdivision and the length and energy of Bézier curves. [online]. Available from: http://portal.acm.org/citation.cfm?id=256526 [Accessed 23 January 2012]

KNUTH, D. 1997. The art of computer programming, volume 2: seminumerical algorithms. 3rd ed. United States of America: Addison-Wesley.

Bibliography

BOEBERT, E. [no date]. Computing the arc length of cubic Bézier curves. [online]. Available from: http://steve.hollasch.net/cgindex/curves/cbezarclen.html [Accessed 24 January 2012]

KANKAANPÄÄ, H. [no date]. Calculating Bézier curves fast via forward differencing. [online]. Available from: http://www.niksula.hut.fi/~hkankaan/Homepages/Bézierfast.html [Accessed 25 January 2012]

MULLER, J.M. 2006. Elementary functions: algorithms and implementation. 2nd ed. United States of America: Birkhauser.

SALOMON, D. 2006. Curves and surfaces for computer graphics. United States of America: Springer.

WANG, W. ET AL. 2008. Computation of rotation minimizing frames. ACM Transactions on Graphics. 27(1): pp.23-41.