TechNote TN-05-003

Quadratic Bezier Curves

Jim Armstrong
December 2005

This is the third in a series of TechNotes on the subject of applied curve mathematics in Macromedia Flash^{TM}. Each TechNote provides a mathematical foundation for a set of Actionscript examples.

Quadratic Curves

From the TechNote on cubic Hermite curves, it should be evident from the Actionscript example that adjusting the shape of the curve using tangent handles is somewhat clumsy. This difficulty increases when creating a user-interface for piecewise cubic Hermite curves.

Several types of curves attempt to retain the general properties of Hermite curves with implicit methods for generating tangents. One such approach was independently discovered by two French automotive engineers, working on ways to interactively create cross-sections for automobile parts.

Bezier curves can be described with a geometric construction, requiring little or no math background. Bezier curves may also be formulated mathematically, using an approach similar to that described in the Hermite curve TechNote. This approach illustrates the mathematical similarity between Bezier and Hermite curves.

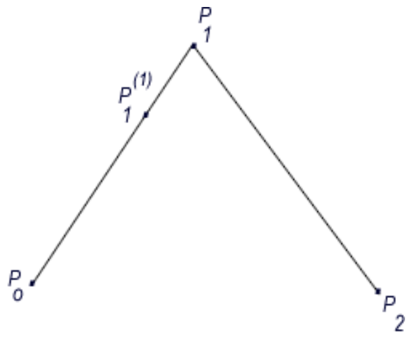
The current discussion separates quadratic, cubic, and general Bezier curves into three separate TechNotes. The geometric construction for quadratic curves is emphasized in the current TechNote. A more formal mathematical derivation is supplied in the upcoming TechNote on cubic curves. The upcoming TechNote on general Bezier curves will discuss implementation and coding techniques.

Since the *MovieClip.curveTo()* function implements a quadratic Bezier curve, this TechNote provides the opportunity to investigate exactly what is happening ‘behind the scenes’ with that function.

deCasteljau’s Algorithm

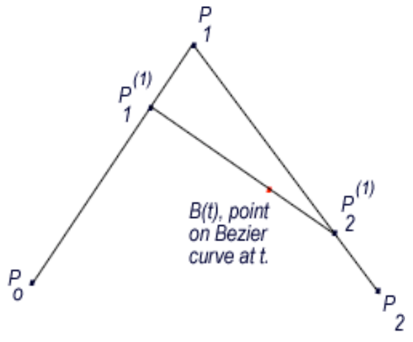
A quadratic Bezier curve is described with three control points. The first and third control points are considered to be ‘anchors.’ The middle control point influences the shape of the curve. The generated curve interpolates the first and third control points and approximates the middle control point.

Points on the curve, $B(t)$, t in [0,1] can be derived via a geometric construction. Consider the three control points in the diagram, below. On the line segment, P_0P_1 mark the point, $P_1^{(1)}$ in the ratio, $(1-t):t$, as shown.



Control cage for quadratic Bezier curve and
first segment division in deCasteljau's algorithm

Find the point, $P_2^{(1)}$, at the same ratio on the segment, P_1P_2,then draw a line from $P_1^{(1)}$ to $P_2^{(1)}$. Locate the point, $B(t)$ at the same ratio on that line segment. This is the point on the curve at the specified value of t, as shown below.



Control cage for quadratic Bezier curve and
second segment division in deCasteljau's algorithm

By varying t from 0 to 1, one can trace the quadratic Bezier curve using this approach. In fact, the approach can be extended to curves of arbitrary degree.

Mathematical Approach

A simple means to derive the equations for a quadratic Bezier curve follows directly from the steps in deCasteljau's algorithm. Every point in the construction is parameterized on t. Equations for each point can be easily written as functions of t. Beginning with $B(t)$,

$$B(t) = (1-t)P_1^{(1)} + tP_2^{(1)} = (1-t)[(1-t)P_0 + tP_1] + t[(1-t)P_1 + tP_2]$$

Collecting terms and simplifying results in

$$B(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2 \qquad [1]$$

With Hermite curves, geometric constraints were the two anchor points and two tangents. With Bezier curves, tangents are implicitly specified. The geometric constraints are the individual control points. These constraints are weighted by blending functions,

$$b_0(t) = (1-t)^2$$
$$b_1(t) = 2t(1-t)$$
$$b_2(t) = t^2$$

These blending functions happen to be the second-order Bernstein polynomials. Higher-order Bezier curves use higher-order Bernstein polynomials as their blending functions.

Bernstein polynomials have some very unique qualities. You are invited to explore some of the many online references, including

http://mathworld.wolfram.com/BensteinPolynomial.html

For example, what is the sum of the three blending functions for the quadratic Bezier curve?

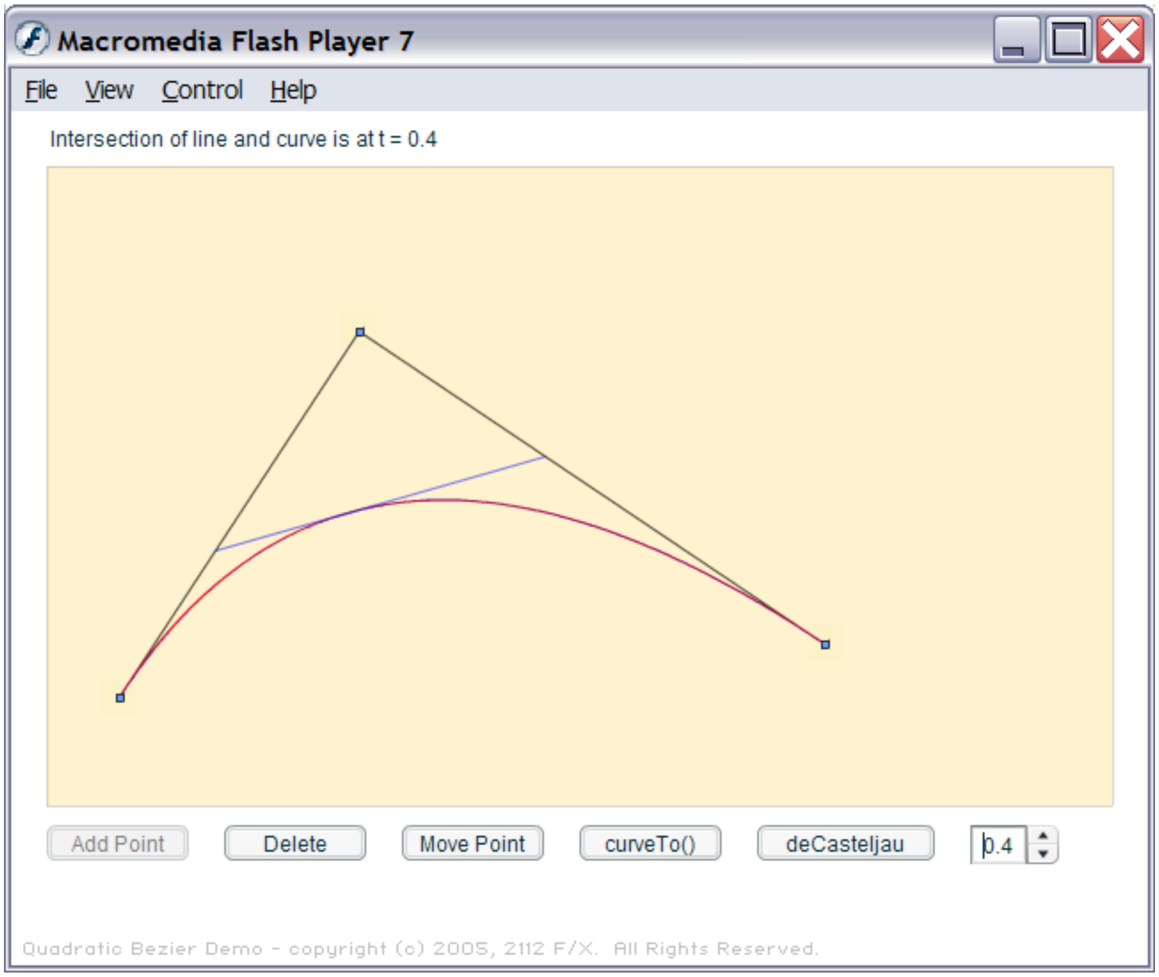
$$b_0(t) + b_1(t) + b_2(t) = 1 - 2t + t^2 + 2t - 2t^2 + t^2 = 1$$

As it happens, this is the case for Bernstein polynomials of any degree.

MovieClip.curveTo() Function

Equation [1] illustrates the fundamental operation performed by the *curveTo()* function of the *MovieClip* class. The parameter, t, is varied from 0 to 1 and a smooth curve is drawn. The Actionscript demo supplied with this TechNote illustrates the creation and evaluation of a quadratic Bezier curve via the *Bezier2* class.

The demo (bezier2demo.fla) allows the result of *curveTo()* to be drawn on top of the Bezier curve (in red) to illustrate the equivalence of the two methods. It also illustrates the deCasteljau construction for different values of t. This is illustrated in the following digram with $t = 0.4$.



Intersection of line and curve is at t = 0.4

Quadratic Bezier Demo - copyright (c) 2005, 2112 F/X. All Rights Reserved.

From the code, you may deconstruct how to apply *MovieClip.curveTo()* to create a quadratic Bezier segment as described in the previous section. Once you understand the application of *curveTo(),* it can be used instead of the *Bezier2* class.

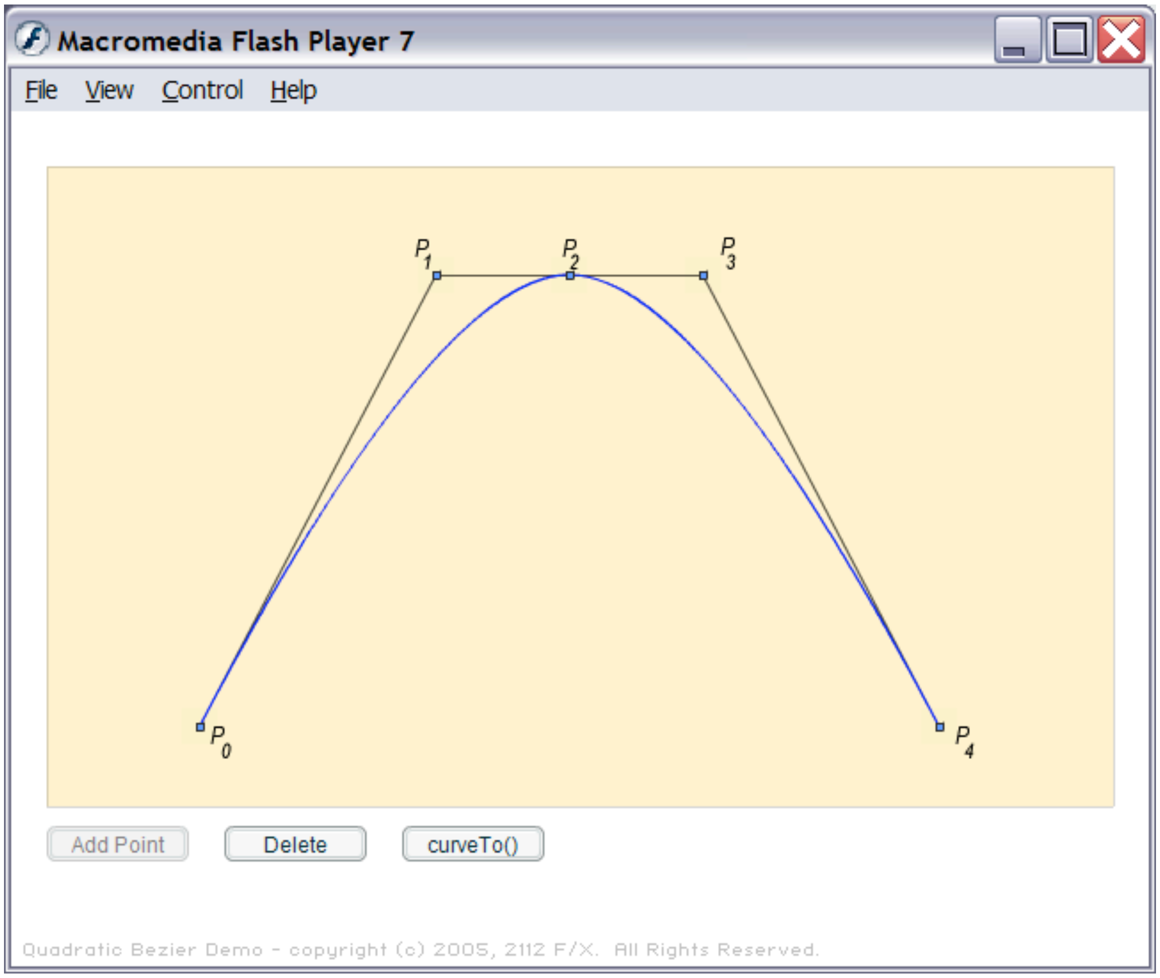
Piecewise Bezier Curve Segments

The Macromedia documentation provides a picture of a control cage (and the resulting curves) for both a quadratic and cubic Bezier segment. The second and third control points for the cubic curve lie on the same horizontal line. The claim has been made that the *curveTo()* function can emulate the cubic curve by essentially dividing the horizontal segment (from the second to the third control point) in half, thus creating two control cages for a quadratic curve. The two cages are symmetric about a vertical line passing through the point of division.

If you understand the quadratic Bezier curve discussed in this TechNote, then you should realize that such a claim is false. Although dividing the horizontal segment in half is a way of creating a piecewise quadratic curve that is C^1 continuous at the join point, Bezier curves interpolate at each endpoint. Dividing the cubic curve's control cage at the middle of the horizontal segment creates two control cages, one from $P_0 - P_1 - P_2$ and another from $P_2 - P_3 - P_4$. Two Bezier segments are generated with a join point at P_2. The first segment interpolates P_0 and P_2. The second curve interpolates P_2 and P_4.

The cubic Bezier curve should interpolate P_0 and P_4, while approximating P_1 and P_3. Since P_2 is on the same line segment as P_1P_3, the cubic curve will not pass through P_2. This is further evidenced by the fact that the cubic Bezier curve is known to be contained within the convex hull of the control cage.

This observation is easier to understand from an actual demo. The second file associated with this TechNote (piecewisequad.fla) illustrates how to construct a symmetric control cage similar to that in the Macromedia documentation. Two quadratic Bezier segments are generated with a join point at P_2. The demo also illustrates how to draw the same piecewise curve using the *curveTo()* function. The result is illustrated below.


Macromedia Flash Player 7
File View Control Help
P_1
P_2
P_3
P_0
P_4
Add Point
Delete
curveTo()
Quadratic Bezier Demo - copyright (c) 2005, 2112 F/X. All Rights Reserved.

Notice that the piecewise curve looks nothing like a cubic Bezier segment for the control cage $P_0 - P_1 - P_3 - P_4$.

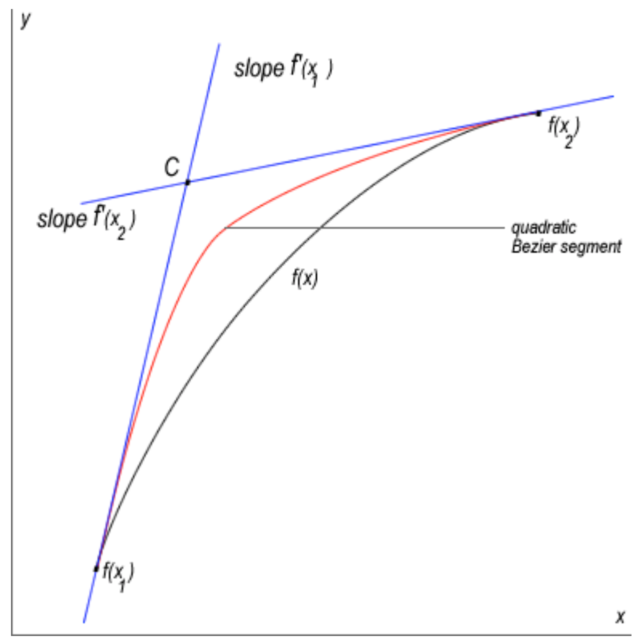
Plotting

As suggested by others in the Flash community, *curveTo()* can be used as a replacement for line-to-line plotting of functions. One advantage of this approach is that the *curveTo()* function provides better anti-aliasing than a line-to-line plot. Generally speaking, a quadratic approximation should work across a larger interval than linear, so an adequate graph might be obtained with fewer intervals and thus less computation.

This approach is not without issues, one of which is the requirement of derivative evaluation in order to construct a control cage. Some people might wish to plot functions but not understand how to code a method to evaluate that function's derivative at a point. While the derivative can be approximated, this process introduces another set of issues. It is also the case that some functions are better plotted in parametric form.

As a prelude to a future TechNote, the following diagram illustrates one method for constructing a control cage from two points of a continuous function with known derivative at each point. The idea is to approximate a function $y = f(x)$ between two points x_1 and x_2 with a quadratic Bezier segment. If the derivative of f is known at the two endpoints, a simple control cage produces a curve that interpolates at the two endpoints and has G^1 continuity at each endpoint. The quality of the approximation depends on the nature of the curve and the separation between endpoints.

The control cage consists of the endpoints $f(x_1)$ and $f(x_2)$ along with the point, C -- the intersection of lines with slopes $f'(x_1)$ and $f'(x_2)$, respectively, as illustrated in the following diagram.



The approximating Bezier segment from the control points $f(x_1) - C - f(x_2)$ is illustrated in red. It is drawn a bit further away from the curve than normal to make the diagram easier to read and to reinforce that the quadratic segment is an approximation to the original function, not an exact fit. The method shown was designed to achieve G^1 continuity at the interpolation points

(x_1 and x_2). If only G^0 continuity is desired, there are many other options for generating an approximating curve segment.

As each line segment passes through a known point with known slope, both equations are well defined. The general equation of each line is of the form

$y - f(x^*) = f'(x^*)(x - x^*)$, where the line passes through the point $(x^*, f(x^*))$. The point, C, can be found by computing the intersection of the two lines.

Further discussion of these equations, solution, limitations, and the overall method of approach is deferred to a subsequent TechNote.

Example Code

(March 2007) The quadratic Bezier code is now part of the AS 3 parametric curve library. The code base requires a development environment capable of compiling Actionscript 3. FlexBuilder 2 is used for the current demos and sample MXML files are provided with the code distribution.

The quadratic Bezier also contains three-point interpolation with an automatic chord-length parameterization.

Cubic Bezier Curves

The next TechNote in this series will discuss the mathematical foundation for cubic Bezier curves. The relationship between the Hermite and Bezier basis matrices is derived. Relevant properties of Bezier curves and the Bernstein polynomials are discussed. The basis for approximation and faster drawing methods is discussed, although detailed examination of these topics is deferred until a later TechNote. That TechNote will feature a method known as recursive subdivision.

Until then, I hope you have achieved a greater understanding of the mathematics behind the *MovieClip.curveTo()* function.

References

1) Farin, G, “Curves and Surfaces for Computer Aided Geometric Design,” Academic Press, San Diego, 1993.

2) Bezier, P, “Numerical Control – Mathematics and Applications,” translated by Forrest, A.R. and Pankhurst, A. F., from “*Emploi des Machines a Commande Numerique*,” Wiley London, 1972.