TechNote TN-05-002

Hermite Curves

Jim Armstrong
Singularity
November 2005

This is the second in a series of TechNotes on the subject of applied curve mathematics in Adobe Flash™. Each TechNote provides a mathematical foundation for a set of Actionscript examples.

Hermite Curves

Hermite curves are a foundation of interactive curve design. All curve design is concerned with the creation of smooth curves based on a small number of user-controlled parameters. Applications of generated curves include cross-sectional elements for extrusion, part of a system for creating 3D surfaces, interpolation between keyframes in animation, and plotting.

It is common to 'anchor' the curve at two endpoints, using other controls to vary the shape of the curve. Cubic polynomials are very popular in curve design. Two of the four conditions required to specify a cubic are taken by placement of 'control points.' The other two conditions are open to adjust the shape of the curve.

Hermite curves are designed using two control points and tangent segments at each control point. Tangent handles may be interactively dragged to adjust the shape of the curve once control points are placed. While very simple to implement, Hermite curves have a number of practical drawbacks. For example, it can be seen from the Actionscript demo that it is difficult to determine how long to make a tangent handle in order to produce a desired shape.

Compensating for these drawbacks leads naturally into Bezier curves. Although cubic Bezier curves are of primary interest, in-depth discussion of quadratic Bezier curves is helpful in understanding the *MovieClip.curveTo()* function.

In addition to deriving the equations to generate Hermite curves, this paper provides background that is used in subsequent TechNotes.

Continuity

It is rare that an application requires the generation of a single curve. Complex curves are often created segment-by-segment, using simpler curves for each segment. The nature of the complete curve is determined by continuity conditions at each join point. If two curve segments have the same value at a join point, the overall curve is said to exhibit G^0 or geometric continuity. G^1 continuity implies that curve segments have matching values at a join point and first derivatives match in direction, but not in magnitude. The overall curve 'hugs' the tangent line more on one side of the join point than the other.

Another type of continuity is called parametric continuity. If two curve segments have matching magnitude and direction of the k-th derivative at a join point, the curve is said to be C^k continuous.

Generally speaking, G^1-continuous curves appear just as smooth as C^1 near join points. The distinction becomes greater further away from join points. If a curve is used to interpolate between keyframes, it is common to consider the animated attribute as parameterized over time, t. As the attribute passes through a keyframe, it is important to maintain consistent velocity and acceleration to avoid jerky motion. C^2 continuity is enforced in these cases even though a C^1-continuous curve may seem to be just as smooth very near join points.

General Form of Parametric Equations

Many curves are defined on a parameter, t. Instead of $y = f(x)$, the curve is defined as $x = f(t)$ and $y = g(t)$, $t \in [0,1]$. In order to quickly compare the mathematical aspects of different curves, it is helpful to have a common framework for deriving the equations of parametric curves. One popular method is to describe the curve as a matrix equation involving a basis matrix, M, a geometry vector, g, and a polynomial vector, p. Define,

$$p_k = [t^k \quad t^{k-1} \quad \dots \quad t \quad 1]^t$$

$$g = [g_1 \quad g_2 \quad \cdots \quad g_k]^t$$

The general form for a parametric curve in t can be written as

$$\theta(t) = [x(t) \quad y(t) \quad z(t)]^t = p_k^t M g \qquad [1]$$

The geometry vector or geometric constraints are determined by endpoints, tangent vectors, or other control points used to define and constrain the curve's shape.

Equation [1] may be a bit difficult for some programmers to understand. Curve equations generally take the same form for x, y, and z-components, so consider only the equation for $x(t)$. Let g_{1x} denote the x-component of the first geometric constraint and consider the equation of the x-component of a cubic curve. Equation [1] simplifies to

$$x(t) = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} g_{1x} \\ g_{2x} \\ g_{3x} \\ g_{4x} \end{bmatrix} \qquad [2]$$

Some readers may find it easier to understand equation [2] in its fully expanded form,

$$\begin{aligned} &(t^3 m_{11} + t^2 m_{21} + t m_{11} + m_{41}) g_{1x} + (t^3 m_{12} + t^2 m_{22} + t m_{32} + m_{42}) g_{2x} \\ &+ (t^3 m_{13} + t^2 m_{23} + t m_{33} + m_{43}) g_{3x} + (t^3 m_{14} + t^2 m_{24} + t m_{34} + m_{44}) g_{4x} \end{aligned} \qquad [3]$$

Notice that the geometric constraints are multiplied (component-by-component) by *blending functions*, represented by the components of $p_k^t M$. The basis matrix, M, determines the coefficients of the blending functions, which give the curve its unique nature or shape.

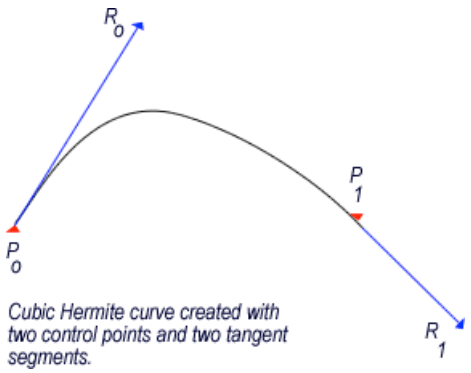
It is common to differentiate between different types of curves based on geometric constraints and the basis matrix. The next section discusses how the basis matrix is derived for Hermite curves.

Hermite Basis Matrix

The Hermite basis matrix is computed by writing a single equation for each of the polynomial coefficients in the blending functions. The geometric constraints can be written in the form,

$$g = [P_0 \quad P_1 \quad R_0 \quad R_1]^t$$

Where P_0 and P_1 are the control points, R_0 represents the tangent vector at the first control point, and R_1 represents the tangent vector at the second control point, as illustrated in the following diagram.



Cubic Hermite curve created with two control points and two tangent segments.

Considering only the x-component of the Hermite curve (as the equations for other components are the same), the required equations are for $x(0), x(1), x'(0), x'(1)$.

The general form for $x(t)$ is $a_0 + a_1 t + a_2 t^2 + a_3 t^3$ which is compared to equation [1] at $t = 0$ and $t = 1$ to yield

$$x(0) = [0 \quad 0 \quad 0 \quad 1]^t M g_x$$

$$x(1) = [1 \quad 1 \quad 1 \quad 1]^t M g_x$$

Since $x'(t) = [3t^2 \quad 2t \quad 1 \quad 0]^t M g_x$,

$$x'(0) = [0 \quad 0 \quad 1 \quad 0]^t Mg_x$$

$$x'(1) = [3 \quad 2 \quad 1 \quad 0]^t Mg_x$$

In matrix form, these equations can be written as

$$g^* = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} Mg^*$$

Where $g^* = [x(0) \quad x(1) \quad x'(0) \quad x'(1)]^t$

The only way this equation can be satisfied is

$$M = A^{-1} \quad , \quad A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}$$

The coefficients of M are easily solved for by setting $AA^{-1} = I$, from which

$$M = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

This yields the blending functions

$$b_1(t) = 2t^3 - 3t^2 + 1$$
$$b_2(t) = -2t^3 + 3t^2$$
$$b_3(t) = t^3 - 2t^2 + t$$
$$b_4(t) = t^3 - t^2$$

A simple way to express the x-component of the Hermite curve is

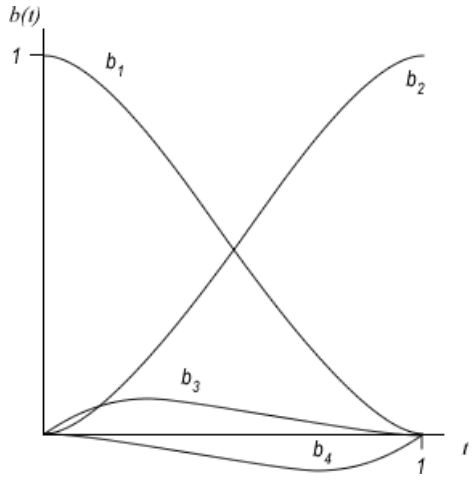
$$(2t^3 - 3t^2 + 1)P_{0x} + (-2t^3 + 3t^2)P_{1x} + (t^3 - 2t^2 + t)R_{0x} + (t^3 - t^2)R_{1x} \qquad [4]$$

An alternate expression that is a bit more computationally efficient is

$$(1 - b_2(t))P_{0x} + b_2(t)P_{1x} + (b_4(t) - t^2 + t)R_{0x} + b_4(t)R_{1x} \qquad [5]$$

An even more efficient approach is to collect t^3, t^2, t and constant terms and then evaluate the polynomial with nested multiplication.

A graph of the blending functions is shown below.



Notice that the blending functions weighting the control points tend to overwhelm those weighting the tangent vectors. This is more noticeable in the Actionscript example.

Piecewise Hermite Curve Segments

Drawing a single segment of a Hermite curve is relatively straightforward. If desired, a more complex shape can be created with piecewise Hermite curve segments. It is necessary to decide whether or not to enforce geometric or (more restrictive) parametric continuity. If G^1 continuity is enforced, then the geometric constraints of the two curve segments at a joint point are

$$\begin{bmatrix} P_0 \\ P_1 \\ R_0 \\ R_1 \end{bmatrix}$$ *curve segment left of join*

$$\begin{bmatrix} P_0 \\ P_1 \\ \alpha R_2 \\ R_3 \end{bmatrix}$$ *curve segment right of join*

where α is a scalar multiplier. If C^1 continuity is enforced, then $\alpha = 1$.

The user interface becomes more complicated with G^1 continuity as the direction of the first tangent handle of the right curve (right, relative to join point) must maintain a fixed direction. Its magnitude only affects the right curve segment.

You can imagine how tedious it might be for a user to construct a complex shape from a series of Hermite curves. In fact, it is not easy to construct desired shapes for a single segment as the tangent handles have to be moved very far away from the control points to produce significant bends (this is evident from studying the graph of the blending functions). For some curves, obtaining a desired shape requires dragging tangent handles outside the available display area.

An old trick to alleviate this difficulty was to artificially weight the blending functions for the tangent vectors. Increasing weight causes the curve to 'hug' each tangent segment more, but producing a desired shape requires even more work as two sets of inputs (tangent handles and weights) must be adjusted.

It is possible to relax the need for tangent handles by converting a Hermite curve into a cardinal spline. A popular form is the TCB (Tension-Continuity-Bias) spline, featured in most modern 3D packages. The formulas for automatically computing tangents were introduced in [2].

Another type of curve that is closely related to Hermite is Bezier. Bezier curves are computed using nothing but control points and a basis of Bernstein polynomials. This is the subject of the next TechNote in this series.

References

1) Ahlberg, J.H., Nielson, E.N., and Walsh, J.L., "The Theory of Splines and Their Applications," Academic Press, NY, 1967.

2) Kochaneck, D. and Bartels, R., "Interpolating Splines with Local Tension, Continuity, and Bias control," Proceedings of the 11th International Conference on Computer Graphics and Interactive Techniques, ACM Press, pp. 33-41, 1984.