Key Frame Interpolation via Splines and Quaternions

David Eberly
Magic Software, Inc.
http://www.magic-software.com

Created: March 2, 1999
Modified: October 28, 2002

1 Introduction

This document illustrates how to interpolate key frames using Kochanek–Bartels splines and quaternion calculus. You should read the documents *Kochanek–Bartels Cubic Splines* (`KBSplines.pdf`) and *Quaternion Algebra and Calculus* (`Quaternions.pdf`) first. A computer implementation is provided for key frame interpolation where both position and orientation are interpolated via splines. A simple application to computer animation is also provided.

2 Squad using KB Splines

The "squad" construction for quaternions can be modified to support the ideas of Kochanek–Bartels splines. While those splines were defined in terms of positional quantities (an additive system), they are easily extended to quaternions (a multiplicative system). In this context, the splines are of the Catmull–Rom type where the derivatives at the control points are centered finite differences,

$$T_n = \frac{\log(q_n^{-1}q_{n+1}) + \log(q_{n-1}^{-1}q_n)}{2}.$$

For tension τ, continuity γ, and bias β, and for a single spline segment between q_n and q_{n+1}, the tangent at $t = 0$ (the "outgoing" tangent) is

$$T_n^0 = \frac{(1-\tau)(1-\gamma)(1-\beta)}{2}\log(q_n^{-1}q_{n+1}) + \frac{(1-\tau)(1+\gamma)(1+\beta)}{2}\log(q_{n-1}^{-1}q_n) \tag{1}$$

and the tangent at $t = 1$ (the "incoming" tangent) is

$$T_n^1 = \frac{(1-\tau)(1+\gamma)(1-\beta)}{2}\log(q_n^{-1}q_{n+1}) + \frac{(1-\tau)(1-\gamma)(1+\beta)}{2}\log(q_{n-1}^{-1}q_n). \tag{2}$$

The intermediate terms for a_n and b_n are constructed as earlier. The equation $T_n^0 = \log(q_n^{-1}q_{n+1}) + 2\log(q_n^{-1}a_n)$ leads to

$$a_n = q_n \exp\left(\frac{T_n^0 - \log(q_n^{-1}q_{n+1})}{2}\right) \tag{3}$$

and $T_n^1 = \log(q_{n-1}^{-1} q_n) - 2\log(q_n^{-1} b_n)$ leads to

$$b_n = q_n \exp\left(\frac{\log(q_{n-1}^{-1} q_n) - T_n^1}{2}\right). \tag{4}$$

Adjustments can be made to the tangents in equations (1,2) to take into account non–uniform sample times, just as in the interpolation of positional data. The multiplier of equation (1) is $2\Delta_{n-1}/(\Delta_{n-1} + \Delta_n)$ and the multiplier of equation (2) is $2\Delta_n/(\Delta_{n-1} + \Delta_n)$ where $\Delta_n = s_{n+1} - s_n$ and s_n is the time value of the sample corresponding to q_n.

3 Implementation

3.1 Position Splines

The files `PosSpline.h` and `PosSpline.cpp` are straightforward implementations of the Kochanek–Bartels splines. The public interface is An application creates position key values and assigns tension, continuity, and bias per key. A sequence of interpolating polynomials and other data required by the interpolation are constructed from the keys. It is assumed that the key times are ordered, `key[i].t < key[i+1].t`, for all i. Since the interpolation requires boundary conditions, an application should choose `key[0]` and `key[numKeys-1]` appropriately. A reasonable choice is to set `key[0] = key[1]` and `key[numKeys-1] = key[numKeys-2]`.

A sample WinMain driver is provided in the source file. There are eight control points, each of the middle six which can be selected with the mouse. The tension, continuity, and bias parameters for the selected control point are displayed in the upper left corner of the window. The keys 't' and 'T' decrease and increase tension, respectively. The keys 'c' and 'C' control continuity and the keys 'b' and 'B' control bias. After each selection, a new spline is built and the spline is drawn.

3.2 Rotation Splines

The files `RotSpline.h` and `RotSpline.cpp` implement rotations in axis–angle format and use squad with the Kochanek–Bartels extension. The interface is identical to that for position splines, except that the basic element of interpolation is a rotation which is represented by an axis of rotation and an angle of rotation about that axis. Use of the functions is the same as in the position case.

A sample WinMain driver is provided in the source file. There are eight control points, each of the middle six which can be selected with the mouse. The tension, continuity, and bias parameters for the selected control point are displayed in the upper left corner of the window. The keys 't' and 'T' decrease and increase tension, respectively. The keys 'c' and 'C' control continuity and the keys 'b' and 'B' control bias. After each selection, a new spline is built and the spline is drawn.

The input quaternions have $z = 0$, so they lie on the 3D sphere which is the intersection of the 4D unit hypersphere and a hyperplane. The control points are drawn on the 2D screen by projecting out the w component (the remaining components are x and y). The quaternion spline curve is a curve on the 4D unit hypersphere. The curve that is drawn is a projection of the original by setting the w and z. It is not the case

that the z components of the squad interpolation are 0. Moreover, the projection into 3D is not necessarily a curve on the 3D unit sphere. The projection into 2D screen space is just to give you an idea of the behavior of the interpolated curve in higher dimensions.

4 Application

A simple WinMain application to animation is provided in `KeyframeAnimation.cpp`. A stick figure representing a leg (hip joint, thigh, knee joint, calf, ankle joint, foot) is animated from a sequence of five key frames (first and last being the same frame, boundary conditions using repeated values at the end points). The thigh, calf, and foot are assumed to be fixed lengths. The hip location, hip angle (measured from the vertical axis), knee angle, and ankle angle are freely chosen parameters. The knee, ankle, and foot positions are determined by inverse kinematics.

More precisely, the free parameters are chosen with respect to the left– handed screen coordinates x and y. All z values are set to zero and all rotations are about the axis $(0, 0, 1)$. Let $\vec{H}$, $\vec{K}$, $\vec{A}$, and $\vec{F}$ be the hip, knee, ankle, and foot positions, respectively. Let θ be the angle formed by the leg $\vec{K} - \vec{H}$ with the y–axis (which points downwards). Let ϕ be the angle between $\vec{H} - \vec{K}$ and $\vec{A} - \vec{K}$. Let ψ be the angle between $\vec{K} - \vec{A}$ and $\vec{F} - \vec{A}$. Let $L_{hk} = |\vec{H} - \vec{K}|$, $L_{ka} = |\vec{K} - \vec{A}|$, and $L_{af} = |\vec{A} - \vec{F}|$ be the specified lengths of the thigh, calf, and foot, respectively. Some trigonometry will show that

$$\begin{aligned} \vec{K} &= \vec{H} + L_{hk}(\sin\theta, \cos\theta) \\ \vec{A} &= \vec{K} + (L_{ka}/L_{hk})R_{\phi}(\vec{H} - \vec{K}) \\ \vec{F} &= \vec{A} + (L_{af}/L_{ka})R_{\psi}(\vec{K} - \vec{A}) \end{aligned}$$

The angles ϕ and ψ are assumed to be acute. The rotation matrix R_{α} has first row $(\cos\alpha, \sin\alpha)$ and second row $(-\sin\alpha, \cos\alpha)$.

The application uses a position spline for $\vec{H}$ and rotation splines for θ, ϕ, and ψ. The key frames are initially drawn, but can be toggled py pressing 'k' and 'K'. Single–stepping through the animation is done by pressing 'a' and 'A'. The correction of the interpolated angles is used since the interpolated axis may be $(0, 0, -1)$. For applications which convert the interpolated axis and angle to a rotation matrix for use in a hierarchical transformation scheme, the flipped axis is irrelevant since the angle has a sign change at the same time.