# Motion Curves

Robert Bridson

September 12, 2011

# 1 Modeling, Animation, and Rendering

In the first course in graphics, we were mostly concerned with *rendering*, producing a 2D image of a "model" consisting of 3D geometry, appearance functions (like colours, shininess coefficients, textures), lighting, a camera pose, etc. If we represent all the information in that model with the label $\mathcal{M}$ and the 2D image as $\mathcal{I}$, then we can express rendering as evaluating a function $\mathcal{R}$:

$$\mathcal{I} = \mathcal{R}(\mathcal{M}).$$

The model will naturally depend on many parameters, which we can express with a functional relationship too:

$$\mathcal{M}(p_1, p_2, \ldots, p_n).$$

For example, parameter $p_1$ might be the $x$-coordinate of the camera, $p_{43}$ the radius of a sphere that appears in the scene, and $p_{3187}$ the red component of one of the lights. Many stages in the film pipeline contribute to creating this model, this massive function $\mathcal{M}$: modeling, rigging, etc.

Continuing in this vein, we can abstractly view the process of *animation* as assigning a value to each parameter $p_i$ in the model for each moment in time. Again, in functional terms, we're coming up with a vector-valued function

$$\mathcal{P}(t) = [p_1(t), p_2(t), \ldots, p_n(t)],$$

and then generating a frame at time $t$ with

$$\mathcal{I}(t) = \mathcal{R}\left(\mathcal{M}(\mathcal{P}(t))\right).$$

This is a very abstract way of thinking about animation, but it does show how much freedom we have in animating—any parameter in the model can be "animated" by making it a function of time.

## 2 Motion Curves

A motion curve is just one of the animated parameters, considered as a function of time. For example, if $p_1(t)$ is indeed our function mapping from time to the $x$-coordinate of the camera in a scene, we can plot it to see it as a curve. If we felt something was wrong with the camera motion, looking at that plot could instantly clue us in to moments where it might be too jerky or overly smooth.

Even better, most animation programs also enable the user to edit a motion by directly adjusting the motion curves in a *curve editor*. This isn't always the most intuitive control—we'll return to that in a bit—but it is a very explicit and transparent mechanism that often is the best way to go.

This brings us to the big question: how do we express easily editable curves? We want to be able to approximate just about any smooth (or mostly smooth) function, we want convenient and intuitive control, reasonably efficient evaluation. We don't necessarily care about exactly matching every function anyone could dream up—as long as the visible error can be made arbitrarily small, approximation is just fine.

This problem has been extensively studied in the context of geometric modeling, in particular CAD (Computer Aided Design).

## 3 Splines

A spline is a *piecewise-polynomial* curve, usually with some constraints on continuity or differentiability, and also usually expressed in a particularly convenient form based on *control points*.

Let $f(t)$ be the motion curve in question (one of the $p_i(t)$'s from above, but I'm changing the notation to avoid ambiguity when we introduce specific values $f_i$, where the subscript index has a very different meaning). We'll only look at $f$ defined on an interval of time $t \in [0, T]$.

Piecewise-polynomial means that $[0, T]$ can be split into subintervals, say $0 = t_0 < t_1 < t_2 < \cdots < t_n = T$, and on each of these subintervals $[t_i, t_{i+1}]$, the function $f(t)$ is equal to a polynomial—but each subinterval's polynomial could have different coefficients.

Polynomials are convenient for designing curves partly because they're so easy to work with (just needing multiplication and addition in their evaluation, produce other polynomials when differentiated), partly because they're infinitely differentiable and have no singularities of any sort on their own, and partly because there are strong theorems like Taylor's indicating their power at locally approximating smooth functions. Low-degree polynomials, cubic and less in particular, also have special physical significance: they can accurately approximate how thin materials naturally bend in the real world.[1]

## 3.1 Lerping

While the simpelst polynomial of all is a constant, and so the simplest spline is a piecewise constant function, a piecewise constant function isn't particularly useful for motion curves—it can't represent continuous motion.[2] Therefore the minimum-degree spline we have interest in is the

[1]Loosely speaking, a thin strip of elastic materal such as wood will settle into a shape which minimizes its curvature in a least-squares sense, subject to constraints such as having its ends fixed or being in contact with some other solid. Modeling the curvature of a function as being its second derivative leads, with some manipulation called "the calculus of variations", to the statement that curvature is minimized when the fourth derivative is zero. A curve with zero fourth derivative is exactly a cubic polynomial. In fact, the word "spline" originally referred to a thin piece of wood or rubber which was constrained by a draughtsman to pass over certain points on a drawing, and its curve could then be traced to get a smooth function—which is naturally pretty close to a cubic curve for these reasons.

[2]Of course, if there is a separate constant value for every single frame it could give the illusion of continuous motion—except there would be no notion of motion blur, which we have already seen is very important.

piecewise-linear spline.

While you could write a piecewise-linear spline in terms of a standard polynomial of degree one on each subinterval, something like this,

$$f(t)|_{[t_i,t_{i+1}]} = a_i t + b_i \qquad i = 0, \dots, n-1, \tag{1}$$

this has many disadvantages. First and foremost, to enforce continuity we have a constraint that the value at $t_i$ approached from the left on interval $[t_{i-1}, t_i]$ has to equal the value approached from the right on interval $[t_i, t_{i+1}]$:

$$a_{i-1}t_i + b_{i-1} = a_i t_i + b_i \qquad i = 1, \dots, n-1. \tag{2}$$

Now we have a system of equations to satisfy, and the user isn't free to directly manipulate the coefficients—which in of themselves aren't very intuitive controls.

A much better, and more natural approach, is to let the user specify the function values at the endpoints between subintervals:

$$f(t_i) = f_i \qquad i = 0, \dots n. \tag{3}$$

In other words, the user gives us the times they care about ($t_i$) and the values of the function there ($f_i$), and it's up to us to construct the piecewise-linear function which passes through those points.

A bit of terminology: we call the values $t_i$ the *knots*; the pair of knot and function values $(t_i, f_i)$ is a *control point* (since that's how the user will control the function), and the requirement that the function passes though the control points is called *interpolation*.

You should already have seen how to do linear interpolation—or *lerping* for short. Consider interval $[t_i, t_{i+1}]$. We need a linear polynomial that equals $f_i$ at the left endpoint and $f_{i+1}$ at the right endpoint. This is fairly easy to work out with a formula $at+b$ as before, but I will give you a more pleasant form which highlights the two control points:

$$f(t)|_{[t_i,t_{i+1}]} = f_i \left[\frac{t_{i+1} - t}{t_{i+1} - t_i}\right] + f_{i+1}\left[\frac{t - t_i}{t_{i+1} - t_i}\right]. \tag{4}$$

It should be obvious that the two terms here are both linear polynomials in $t$ so their sum is, and that the appropriate values are achieved at the endpoints, so this has to be the answer.

## 3.2 Smoothness

Piecewise-linear curves are tremendously useful throughout animation, but the one problem with them is that they aren't very smooth. Obviously a straight line is perfectly smooth, but at the knots, the junctions between one interval and the next, the piecewise-linear curve can have a sharp "kink".

One of the technical ways to define smoothness in mathematics is by classifying functions as $C^0$ if they are continuous, $C^1$ if they are differentiable and the first derivative is continuous, and in general $C^k$ if the function has a $k$'th derivative which is continuous. Obviously every $C^1$ function is also $C^0$, and more generally you can think of these as nested spaces of functions. Polynomials, amongst many other functions such as $\sin(t)$, $e^t$, etc., have infinitely many derivatives and thus are called $C^\infty$.

A piecewise-linear spline is always $C^0$ (continuous) but in general can't be $C^1$. If it represents a position, its velocity (the first derivative) isn't well defined at the knots $\{t_i\}$, where the velocity instantaneously changes value. (In fact, the first derivative of a piecewise-linear function is a piecewise-constant function, with undefined values at the jumps between constant values.)

We know from physics that most real motion is smoother than that. Arguably all motion has to be at least $C^1$ from Newton's Law $F = ma$, in fact. Very rapid collisions are usefully modeled as being an instantaneous change in velocity (corresponding to infinite forces over an infinitesimal period of time)—but those $C^0$ events are the exception to the rule.

Splines are always $C^\infty$ between knots, due to the infinite smoothness of polynomials. For a piecewise-polynomial to be $C^0$, the values at the knots approached from the left and right have to be equal. For a spline to be $C^1$, the first derivatives or slopes at the knots approach from the left and right have to match. For a spline to be $C^2$ it's that as well as matching the second derivatives.

Looking at piecewise-linear functions we see we can't possibly do better than $C^0$ in general. Interpolating the values at the endpoints uniquely specifies the linear polynomial on any interval, thus there are no more degrees of freedom to modify to achieve any other desired quality such as slopes matching at knots. The only solution, then, is to use a higher degree polynomial.

## 3.3 Representation

A polynomial can have many representations. For example, all of these formulas represent the same cubic:

$$t^3 + 2t^2 - t + 3, \tag{5}$$

$$((t+2)t-1)t+3, \tag{6}$$

$$(t-1)^3 + 5(t-1)^2 + 6(t-1) + 5, \tag{7}$$

$$3\left[(1-t)^3\right] + \tfrac{8}{3}\left[3(1-t)^2 t\right] + 3\left[3(1-t)t^2\right] + 5\left[t^3\right]. \tag{8}$$

The first form (5) is probably what you're most familiar with; the second (6) is almost equivalent but rewritten in a way that is usually more efficient to evaluate (*Horner's Rule*) since it involves less multiplies. The third form (7) is written in powers of $(t-1)$ instead of $t$ but evaluates to the same values: however, whereas the coefficients in the first two forms are $[1 \quad 2 \quad -1 \quad 3]$, the coefficients for the third form are $[1 \quad 5 \quad 6 \quad 5]$. The last form (8) is an example of a Bézier spline, which we will discuss below.

Before getting into a selection of the "best" representation to use, we should take a second to think clearly about how to switch between representations, and even what a representation really means. We'll declare that (5) and (6) in the example really are the same representation: the defining coefficients are the same even though the exact sequence of arithmetic operations implied are slightly different.

We can view the standard form of a polynomial,

$$a_0 + a_1 t + a_2 t^2 + \dots a_k t^k, \tag{9}$$

as one way of identifying a "vector" from the $(k+1)$-dimensional vector space of degree $k$ polynomials. In particular, we can express that vector space as the span of a set of basis vectors,

$$\text{span}(1, t, t^2, \dots, t^k), \tag{10}$$

and equation (9) is just expressing a linear combination of those basis vectors with coefficients $a_0, a_1, \dots, a_k$.

There are many other choices for a basis for any vector space, of course. For example, for degree one polynomials, the standard basis is

$$\{1, t\}, \tag{11}$$

but an equally valid basis is

$$\{1 - t, t\}. \tag{12}$$

Any degree one polynomial represented in the first basis as

$$a_0 + a_1 t \tag{13}$$

can be represented with different coefficients in the second basis as

$$a_0[1 - t] + (a_1 + a_0)[t]. \tag{14}$$

Formally we could write:

$$\begin{bmatrix} 1 & t \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 1-t & t \end{bmatrix} \begin{bmatrix} a_0 \\ a_0 + a_1 \end{bmatrix}. \tag{15}$$

We can even express this change of basis with a matrix:

$$\begin{bmatrix} a_0 \\ a_0 + a_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \quad \Leftrightarrow \begin{bmatrix} 1 & t \end{bmatrix} = \begin{bmatrix} 1-t & t \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{16}$$

So the question for us becomes how to select a useful basis, and conversion between representations is simply multiplication by a change-of-basis matrix which can be pre-computed.