BROWN UNIVERSITY
Department of Computer Science
Master's Project

CS-95-M13

"Gestural Controls for Computer Animation"

by

Scott Snibbe

Gestural Controls for Computer Animation

by
Scott Snibbe

Department of Computer Science
Brown University

Submitted in partial fulfillment of the requirements for the degree of Master of Science in the Department of Computer Science at Brown University

September 1994

Professor John F. Hughes
Advisor

Table of Contents

1.0 Introduction

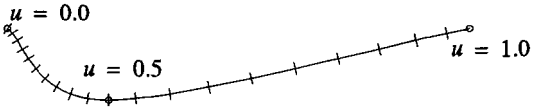
Computer animation systems today fragment the process of animation by unnaturally separating motion control from the visualization of a scene. Spatial and temporal changes are normally modified from separate editing windows — one for modifying an object's current position in space and others for modifying the object's changing path over time. A majority of animation systems only allow precise motion control of single channels of motion. It is left to the skilled animator's mind to integrate these separate channels and views of motion. This research is an attempt to develop interactive techniques for visualizing, creating and modifying motion directly in a two- or three-dimensional scene. The principles of direct manipulation are used to achieve the goal of fluid and natural interaction. Our solution uses existing keyframe and parametric techniques in combination with *displacement functions* inspired by digital signal processing for real-time direct manipulation of spatial and temporal changes. The discussion in this paper will focus almost entirely on the control of the positional attributes of an object. However, the solutions presented here are adaptable to the animation of orientation, color, or any other attribute that varies over time. Expanding this research to these areas is discussed in the section on future work.

1.1 Problems in Existing Animation Systems

Several problems found in a majority of commercial and research animation systems served as the motivation for this research. Not all of these problems are present in all systems, but these are current trends in a large class of existing systems.

- **Animators can completely visualize and edit motion only in separate 2D graphs:** The only means to edit an object's time-varying properties *and* visualize the value of these properties at prior and subsequent times is through 2D graph editors. The 2D or 3D scene view can only be used for viewing and altering an object at a single time point.
- **Editing of motion curves is limited to single channels of motion:** Motion curves are normally limited to representing a one-dimensional parameter vs. time (e.g. x translation vs. time, y rotation vs. time, red color component vs. time). Animators must mentally integrate all of these channels to visualize the animation which they are creating. The best computer animators today are expert at this process. In practice, many animators rely solely on the graph views for many hours without even referencing the scene preview[Chenoweth93][Pasquale93].

- **The natural parameterization of splines does not advance uniformly with respect to distance.** Many systems allow the animator to specify the path of an object through space with a two- or three-dimensional spline curve. Motion along this curve is then described by a single function of u vs. time, where u is the natural parameter of the spline curve. However, equal steps in u result in unequal distances traveled along the curve. In these systems, a graph that appears to indicate constant velocity will actually result in a velocity that varies based on the shape of the curve and the spacing of its control points (figure 1.1). The animator is forced to cancel out the timing induced by the spline before creating the desired motion (figure 1.2).



k-b spline with 3 control points

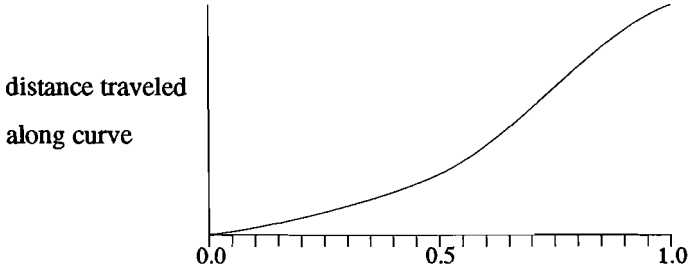


FIGURE 1.1. Parameter value vs. distance traveled for a sample spline curve. A 3 point Kochanek-Bartels spline is shown (top) with tick marks at even parametric intervals. Below, a graph of parameter value (u) vs. actual distance traveled along the curve.

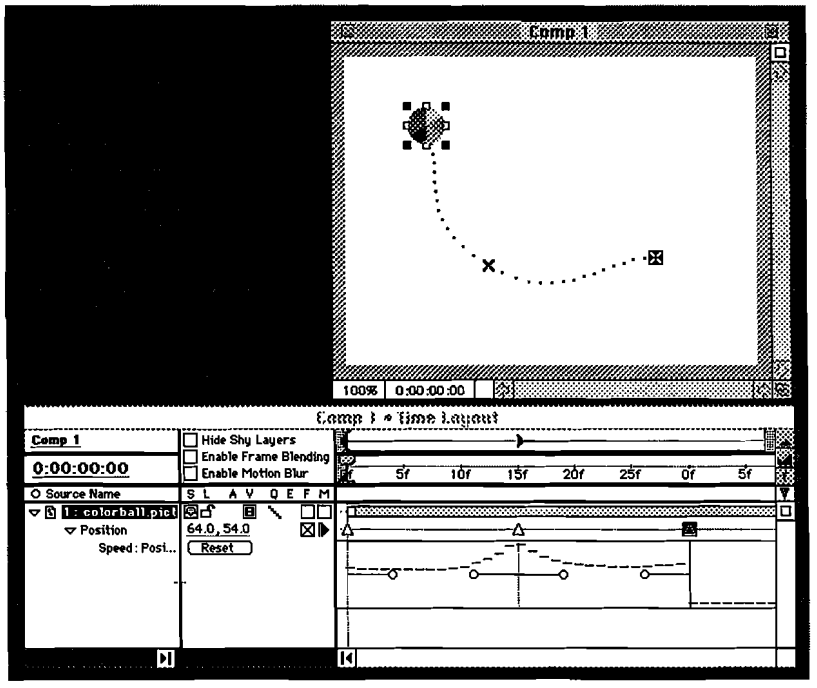


FIGURE 1.2. Timing induced by spline type in AfterEffects [CoSA93]. Three positions for a ball object have been set using a Catmull-Rom spline for spatial interpolation. No changes to the motion have been made; however, the velocity curve for the object indicates velocity increasing around the control points. This is a result of animating with respect to the natural parameterization of the spline. An animator must attempt to first undo the timings induced by the curve, then introduce their own timing changes.

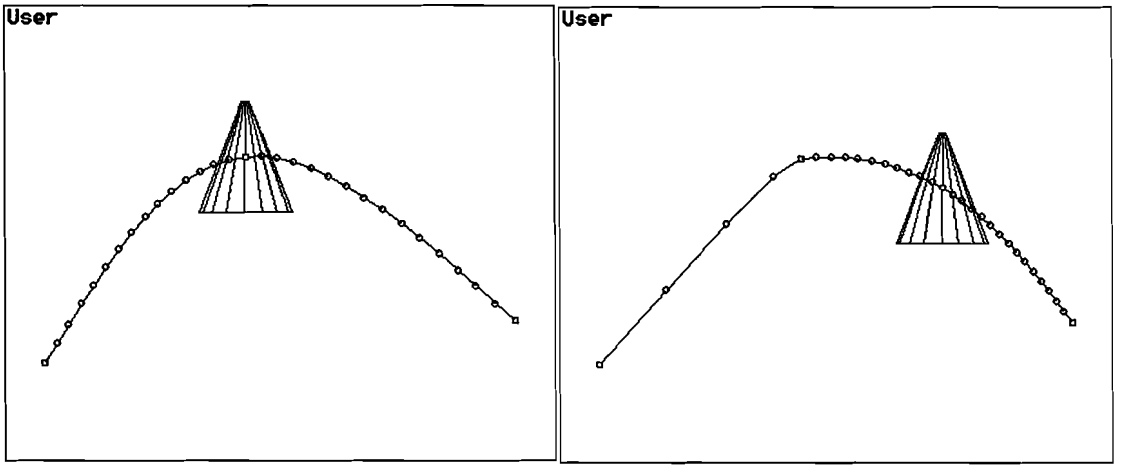


FIGURE 1.3. Altering the shape of a motion curve to achieve timing goals in 3D Studio [Autodesk93]. A short animation along a curve with three control points is shown at left. 3D Studio allows the animator to adjust the timing of the animation by directly dragging the object along the curve, compressing and expanding the surrounding motion (right). However, since motion is tied to the natural parameterization of the curve, their system must also modify the shape of the curve to achieve the temporal goal.

- **The shape of a motion curve is altered to achieve timing goals:** Some systems alter the shape of a motion path when users edit the timing of an animation. This problem is also a result of tying motion to the *u*-parameter of a spline. The actual shape of the curve must be changed in order to alter the distance travelled over equal time steps (figure 1.3).
- **Direct manipulation of the animated object is allowed only at control points:** When a spline curve is used as the underlying representation of spatial change, most systems only allow the animator to change the object at the spline control points [ElectricImage93][TDI93]. If the animator wants to alter a position between control points, she must either work indirectly, altering surrounding control points and tangents, or she must add a new control point.
- **Animations with densely spaced keyframes are difficult to modify:** Most production quality animations end up being specified by very densely spaced keyframes (10-15 keyframes/second is normal) [Chenoweth93][Pasquale93]. If an animator decides that part of the motion should be changed, she must individually change a wide range of control points surrounding the specific change in order to blend it with the surrounding motion. Many animators find it faster to re-do the animation from scratch in this situation (figure 1.4).

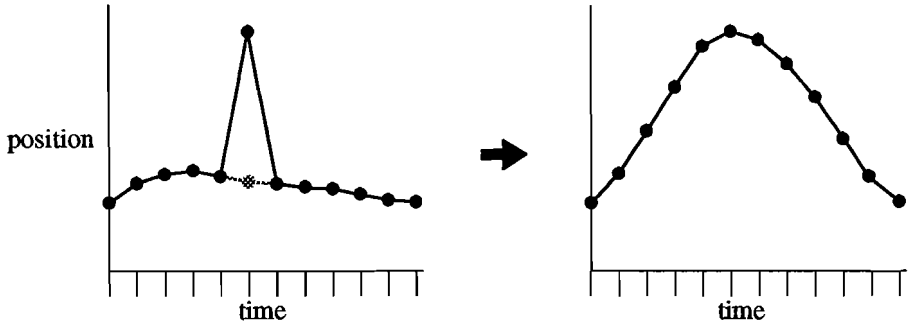


FIGURE 1.4. Modifying densely spaced keyframes. A single change (left) must be blended into the surrounding keyframes by individually adjusting several neighboring keyframes on either side.

1.2 Prior Work

There is a large body of previous published research on kinematic animation and motion control. This section will only address those prior models that incorporated splines to control spatial interpolation and motion control.

A significant problem in computer animation for many years involved finding a type of spline that would pass through a given set of control points smoothly. Kochanek and Bartels invented a spline type which achieved this goal, and also incorporated timing controls into the model [Kochanek84]. Their model uses three parameters (*tension, bias* and

continuity) whose values are set at each control point. These parameters alter both the shape of the curve and the spacing of u around each control point. An animator using the system first gets an intuitive feeling for the effects of these controls, then attempts to achieve her animation goals using these controls. The effects of these parameters are often intuitive, changing the shape and timing in a manner consistent with many naturalistic motions. However, intimately connecting shape and timing has severe drawbacks. Animators often end up in a tug-of-war in which they achieve the desired timing at the expense of the motion curve's shape, or vice-versa.

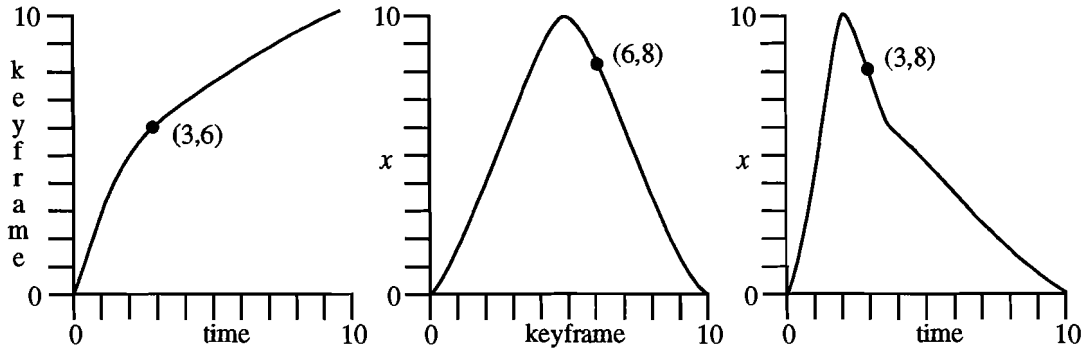


FIGURE 1.5. Double-interpolant method (adapted from [Steketee85]). The leftmost curve of keyframe vs. time is composed with the x vs. keyframe graph (center) to determine the actual value of x over time (right). As an example, a single point is traced through the three graphs. At time t=3 the keyframe number is 6 (left). Looking up keyframe 6 in the middle graph gives an x value of 8. Therefore, in the final graph (right) we see that time t=3 corresponds to an x coordinate of 8.

Steketee and Badler [Steketee85] published an early paper describing a method for separating temporal control from spatial control in parametric animation. Their work uses B-splines for graphing object attributes. One set of curves represents position vs. keyframe for individual object attributes (e.g. x, y, z). A second curve of keyframe vs. time is used to separately modify the timing of the object attributes. The composition of the two curves results in the final value for a given attribute (figure 1.5). These timing curves can be applied to multiple parameters at once, so that adjusting a single timing curve modifies the timing of several different related object attributes. The position and timing of the object being controlled are visualized in the final view by placing tick-marks at even intervals of time along the composite spatial curve. A drawback of their system is that the spa-

tial attributes are separated into multiple channels and no direct manipulation of temporal information is allowed. Control of motion involves editing two separate graph views (the object attribute graph and the timing curve), then viewing the actual spatial path to evaluate changes. Furthermore, the effective range of a change in any graph is limited by the number and spacing of control points in the given curve and cannot be arbitrarily controlled by the animator.

The Menv system developed at Pixar uses only a single graph of value vs. time for object attributes [Reeves90]. Key values along these curves are connected by spline segments. The shape and type of these splines can be modified in a piece-wise manner, choosing the best shape and type for each segment of time (figure 1.6). The system has the drawback that the x, y and z positions must be edited separately, and timing curves can only be modified at their control points by using their tangents. However, the system was developed in cooperation with traditional animators who find it a convenient way to specify and visualize motion. Once an animator gets used to the idea of viewing changes in graph form, the Pixar system provides an efficient way to create motion. The Pixar model is the method used by most commercial animation systems today.

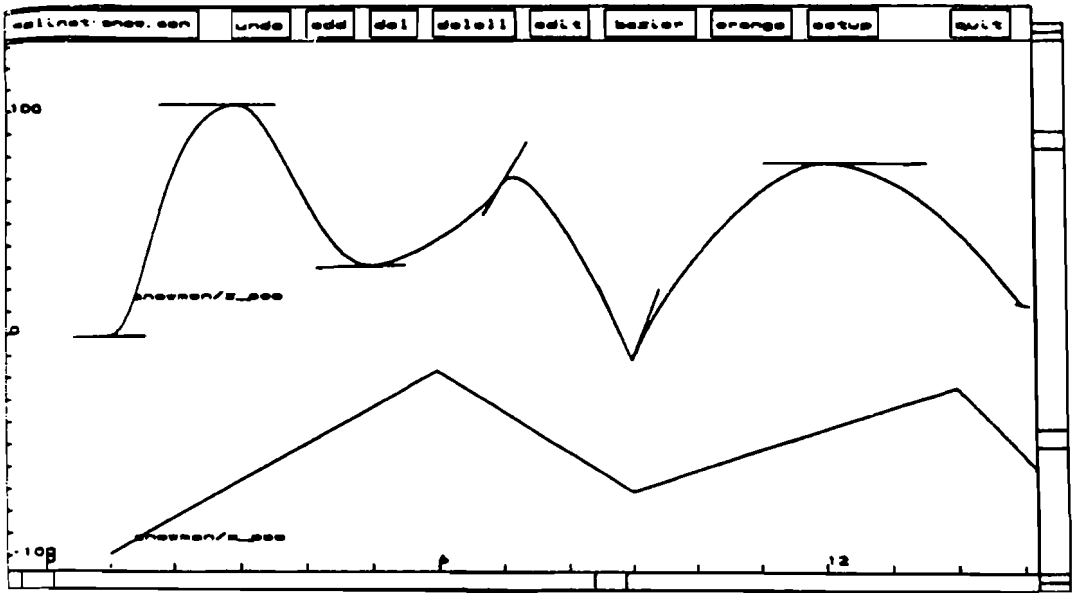


FIGURE 1.6. Editing motion curves in the Menv animation system. Pixar's animation system allows the editing of single parameters vs. time. Keyframes are connected by spline segments which the animator can modify directly by altering the tangents or changing the spline type. Note that only one channel can be edited at a time. The top curve represents the z position of the object, while the curve below shows its x position. Diagram from [Reeves90].

The Inkwell system developed at the Apple Advanced Technology Group includes a set of tools for digitally filtering densely spaced control points in timing curves [Litwinowicz91]. The authors interpret each hand-animated or sampled motion curve as a digitally sampled signal. Animators can modify the gain, decay or oscillation of their motion curve by filtering with an infinite impulse response filter(figure 1.7). To the animator, these parameters are understood intuitively as magnitude, wiggle and lag. Their system also has a cosine blending function to blend changes from a single frame into surrounding control points.

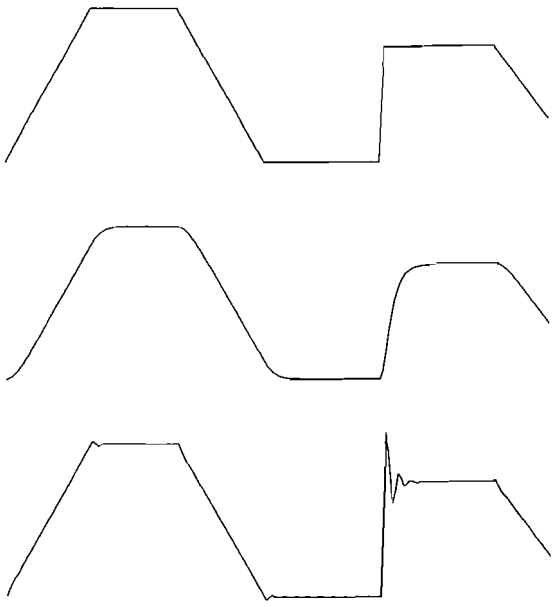


FIGURE 1.7. IIR filter applied to a motion curve. A digital IIR filter can be applied to motion curves to achieve the intuitive animation effects of wiggle and lag. The filter is controlled by parameters representing gain, decay and oscillation. The top curve is the original unmodified graph. The middle curve is the result of applying a filter with gain = 1, delay = 0.625 and osc = 0.0. The filter smooths and delays the original signal. The bottom curve results from applying the filter (gain = 1, delay = 0.67, osc = 1.5132). This results in both wiggle and lag of the underlying parameter.

Several commercial programs allow the spatial path of an object to be described by a three-dimensional spline curve. Timing is then specified by a graph of u vs. time, where u is the underlying parameter of the spline. These systems all have the drawbacks mentioned in section 1.1: Equal steps in u do not result in equal distance traveled, and manipulation of the object is only supported at the spatial control points [TDI93][ElectricImage93] [CoSA93].

Many researchers from fields outside of computer animation are working towards better direct manipulation of spline curves and surfaces. Least-squares techniques [Fowler93], constraint-based techniques [Welch92] and oriented particle systems [Szeliski93] are the major areas currently being explored for directly manipulating spline

shape and complexity at any point on the curve. These approaches will be more fully explored in section 3.0.

2.0 Goals for Gestural Animation Controls

The following set of goals serve as the basis of this research. They were arrived at through evaluation of the problems in existing commercial and research animation systems. They attempt to describe an interface that will be more intuitive and productive for animators.

- **Create a parametric animation system that visualizes temporal and spatial information in a single view.** Achieving this goal should reduce the cognitive load on an animator, enabling her to directly view changes without mentally integrating separate graphs and scene views.
- **Allow direct manipulation of the object being animated while maintaining a parametric curve as the underlying representation.** An animator should be able to directly manipulate the actual object at the current time rather than editing a spline curve and tangents to indirectly achieve motion goals. For animators skilled in keyframe animation, this goal will allow them to exploit their existing skills without learning new interaction techniques. Maintaining a parametric curve as the underlying representation allows control of spatial and temporal continuity and changes in resolution impossible with other representations.
- **Parametrize the spatial curve by arc-length so that equal steps along the curve's new parameter *s* result in an equal distance travelled.** Animators should be able to think in the natural terms of distance or velocity vs. time when editing and visualizing motion. Mathematically, this is achieved by re-parameterizing the curve by arc-length.
- **Create a series of gestural tools for modifying spatial and temporal information.**

  - "Reach this point at this time" while maintaining the shape of the motion curve, but changing the speed at which the object travels along the given path. This is accomplished by compressing or expanding surrounding temporal information.

  - "Reach this point at this time" by modifying the spatial curve but maintaining either the duration or velocity of the given segment.

  - "Make this segment longer, shorter, or of a specific duration".

  - "Go faster", "Go slower", or "Reach a specific velocity" at a given point, while maintaining the shape of the spatial curve and the duration of the temporal segment.
- **Allow an arbitrary range over which changes in space or time will be smoothly blended into surrounding motion.** Animators should be able to change a portion of the animation without re-doing a large amount of the surrounding motion. Changes should be smoothly integrated in space or time, maintaining the character of the original motion to the greatest extent possible.

- **Provide real-time performance for spatial and temporal curves with more than 300 control points.** An animator is routinely dealing with motion curves containing control points at nearly every frame. For a 10 second scene at 30 fps, this means 300 temporal control points and 300 spatial control points. Response time for this type of scene should be a maximum of 200 ms, preferably much faster, allowing immediate interaction and time for other computation and rendering. In general, this will preclude using iterative constraint solving techniques for any of these tools.
- **Provide direct editing of motion graphs or spatial curves using control points.** The new techniques should not preclude hand adjustment of individual temporal or spatial control points. Animators should be able to freely switch between the gestural integrated tools, direct manipulation of the objects, and editing of control points and graphs in separate views.
- **Develop techniques that are naturally extensible to orientation, scale and any other animatable parameters.**

3.0 Implementation of Gestural Animation Controls

Our approach to meeting these goals is to control an object's movement through space by using two spline curves for each object. The first spline curve, Q, is the *motion path*, describing a path through space along which an object travels:

$$Q : \Re \rightarrow \Re^3$$

We express Q as a function of the parameter u such that

$$Q(u) = (x(u), y(u), z(u))$$

The second curve, S, is a function of distance vs. time and is referred to as the *motion graph*:

$$S : \Re \rightarrow \Re$$

This function maps from a time value t to a distance travelled along the curve, s:

$$S(t) = s$$

Since the curve Q is parametrized by its natural parameter, u, and we wish to parametrize it by arc-length (s), we must create a mapping from s to u and vice-versa. This problem is addressed in section 3.1 where the following function A, which maps from u to s, is defined:

$$A(u) = s$$

Our goals for motion control require that we satisfy mathematically precise goals for the shape and derivative of both the motion graph and the motion path. We will discuss two separate approaches to achieving these goals. The first approach uses popular constraint-based techniques, building on the work developed by Gleicher, Welch, Moreton and other authors [Gleicher91][Welch92][Moreton92]. This approach is discussed in section 3.2 along with its significant drawbacks. A more successful approach uses displacement functions to solve the curve-manipulation problem, which is discussed in section 3.3. In section 3.4 we show how to achieve direct manipulation of temporal information in a single view. Both the constraint and displacement techniques are applicable to a wide range of spline types; in section 3.5 we discuss the particular representations we use for our implementation. Section 3.6 discusses techniques for visualizing motion in a single two or three-dimensional view.

3.1 Separating Spatial Control from Temporal Control in Splines

In order to separate spatial and temporal control in splines, we first re-parametrize the motion path by arc-length. Suppose we have a function

$$A : \Re \rightarrow \Re$$

where $A(u)$ is the distance travelled along Q from 0 to u. We can use this function to map between the motion graph curve $S(t)$ and the motion path curve $Q(u)$. The function A gives us the arc-distance travelled, s, given a parameter value u:

$$s = A(u)$$

However, to determine a point along Q for a given time t, we must map from the distance value, s given by $S(t)$ to the u value over which the curve $Q(u)$ is parametrized. This can be accomplished by inverting the function A:

$$u = A^{-1}(s)$$

To determine the u parameter value along the curve Q for a given time t, we evaluate this function at the s value given by the function S at time t:

$$u = A^{-1}(S(t))$$

Now we can determine an object's position at a given time with the following equation:

$$P(t) = Q(A^{-1}(S(t)))$$

The entire process is illustrated in figure 3.1.

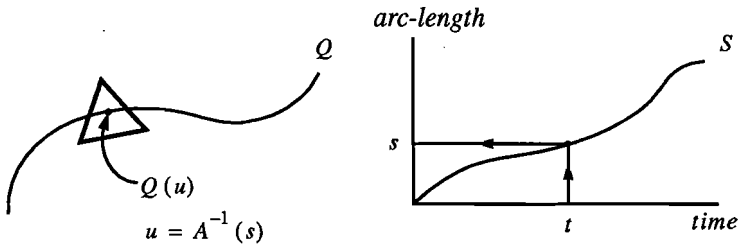


FIGURE 3.1. Motion control with separate space and time curves. The triangular object is driven along the curve Q by S, a function of distance vs. time. The function S at right gives us the distance travelled, s, for a given time, t. The parametric distance along the curve, u, is then determined by A^{-1}. The actual position of the objects is determined by evaluating $Q(u)$.

In practice, a velocity curve $V(t)$ is more commonly used as the motion graph, since subtle changes of velocity can be more easily visualized. However, $V(t)$ can be integrated without difficulty to determine $S(t)$. For ease of manipulation, the graph $S(t)$ is often represented as a two-dimensional spline curve. Animators find this representation easier to manipulate and capable of finer control of motion with fewer control points. However, interpreting a two-dimensional spline curve as a function of one variable involves some subtleties, which are discussed in section 3.1.2. In this paper we will usually refer to $S(t)$ as a one-dimensional function for simplicity. However, keep in mind that S may in fact be represented parametrically as a two-dimensional spline. There turns out to be no analytic expression for the arc-length function A for an arbitrary spline Q, so the computation and inversion of this function are accomplished using numerical methods, which are discussed in the next section.

3.1.1 Reparametrizing Splines by Arc-length

In order to parametrize a spline curve by arc-length we must take the original curve $Q(u)$ and find a formulation for the arc-length s along this curve with respect to the original parameter u:

$$s = A(u)$$

The spatial curve's parametric formulation in 3-space can be described as follows:

$$Q(u) = (x(u), y(u), z(u))$$

A segment of arc-length ds between points $Q(u)$ and $Q(u + du)$ should be familiar from vector calculus:

$$ds = \sqrt{dx^2 + dy^2 + dz^2}$$

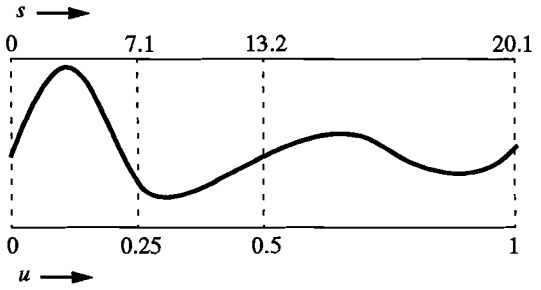
$$= \sqrt{\left(\frac{dx}{du}\right)^2 + \left(\frac{dy}{du}\right)^2 + \left(\frac{dz}{du}\right)^2}\,du$$

To determine the arc-length from the start of the curve to a point at parameter value u, we integrate this equation:

$$A(u) = \int_0^u \sqrt{\left(\frac{dx}{du}\right)^2 + \left(\frac{dy}{du}\right)^2 + \left(\frac{dz}{du}\right)^2}\,du$$

In general there is no analytic expression for this integral, so instead we use numerical techniques to evaluate it. An adaptive technique for evaluating this integral is presented in [Guenter90]. The method recursively subdivides the curve and evaluates the integral of the halves using Gaussian quadrature (a numerical integration scheme described below). A table of (u_i, s_i) pairs is built up through the recursive evaluation of the integral. At each recursive step, the interval is divided into two equal length subhalves. The integrals over these intervals are then evaluated using quadrature and the sum of the values for the two regions is compared with the value calculated for the entire interval. Once the difference between these values passes below the desired accuracy, the subdivision process halts for a given interval. The resulting table is then used to rapidly find the arc-length value for an arbitrary u value. The table is searched for the values u_i, u_{i+1} such that $u_i \le u \le u_{i+1}$.

The arc-length from u_i to u is then computed using quadrature. An illustration of this process is shown in figure 3.2.



u	s
0.0	0.0
0.25	7.1
0.5	13.2
0.5	20.1

FIGURE 3.2. Arc-length computation using adaptive subdivision (adapted from [Guenter90]). To approximate the arc-length of a curve, we subdivide the curve and evaluate the sub-intervals using Gaussian quadrature (left). When the difference between the sum of 2 sub-intervals and the value calculated for the entire interval passes below a given threshold, values are entered into a table (at right). The table is then used to find the arc-length for arbitrary points along the curve, as described in the text.

The Gaussian quadrature integration technique works by taking the sum of a set function values at specific points, multiplied by appropriate weights. The quadrature technique for integration requires that the arbitrary interval $[a,b]$ is first mapped to the interval [-1,1]. A linear transformation

$$\tau = \frac{2x-a-b}{b-a}$$

can be used to accomplish this remapping. The integral of a function f can then be expressed as:

$$\int_a^b f(x) = \int_{-1}^{1} \frac{(b-a)}{2} f\left(\frac{(b-a)\tau + b + a}{2}\right)$$

This integral is evaluated at n fixed points $x_1 \ldots x_n$ within the interval [-1,1] and weighted by a set of weights $c_1 \ldots c_n$ to determine the best approximation of the integral:

$$\int_a^b f(x) \approx \sum_{i=1}^{n} c_i f(x_i)$$

The points and weights for different orders of Gaussian quadrature can be found in standard mathematical tables or computed as shown in [Press92].

The adaptive Gaussian quadrature scheme is unfortunately not guaranteed to perform correctly in all cases. The error estimation process is flawed and for certain curves may cause the refinement process to halt prematurely. This case can occur when the quadrature estimate for an interval happens to match the sum of the 2 subdivided regions even though the estimates for the sub-intervals are erroneous estimates. This degenerate case seems to be rare, however, and the scheme performs well in practice.

In order to find the u value corresponding to a given s value such that $A(u) = s$ we use Newton-Raphson iteration on the subinterval $[u_i,u_{i+1}]$ corresponding to the interval $[s_i,s_{i+1}]$ such that $s_i \leq s \leq s_{i+1}$. This process becomes unstable where the motion curve's derivative approaches zero, in which case binary subdivision can be used.

In our system, the animator is often in a tight interaction loop modifying a curve's shape, requiring real-time feedback of the new arc-length values along the curve. In this case the adaptive quadrature technique is too time consuming. We use a simple forward differencing scheme for rapidly computing an approximation of the arc-length. A table of (u, s) pairs is built up by sampling at even intervals of u along the entire curve. When the u value of a particular value of s is requested, we linearly interpolate between the bounding values $[u_i,u_{i+1}]$ corresponding to the interval $[s_i,s_{i+1}]$ such that $s_i \leq s \leq s_{i+1}$. The adaptive quadrature technique is only used when the final motion is requested or a more accurate preview is desired. A comparison between normal and arc-length parameterization can be seen in figure 3.3.

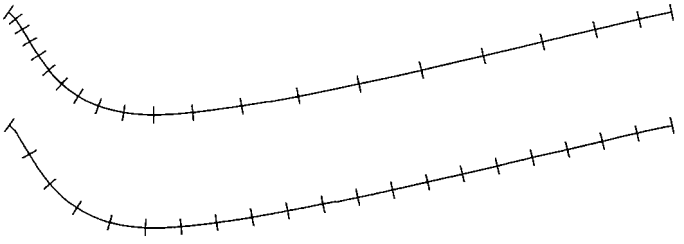


FIGURE 3.3. Comparison of parametric and arc-length subdivisions. The same curve is marked at equal parametric intervals (above) and equal intervals of arc-length (below).

This section was adapted from and inspired by a number of sources. For further numerical approximations to integration see [Burden81] and [Press92]. For a more complete discussion of arc-length computation see [Watt92] and [Guenter90].

3.1.2 Interpreting 2D Splines as Function Graphs

In order to use a two-dimensional spline curve as a function graph we must find a way to map from the parametric representation of the curve

$$S(u) = (t(u), s(u))$$

to a functional representation of the curve

$$S(t) = s$$

Since the parametric curve $S(u)$ is not necessarily one-to-one, this mapping does not make sense for arbitrary curves. However, in the interface for manipulating graphs we can set sufficient constraints to ensure that the resulting parametric curve is one-to-one. Assuming that we do have a parametric curve that is one-to-one, the solution is straightforward. For a spline curve consisting of a single cubic segment, we first find the particular u value u_i corresponding to a given t value t_i:

$$t_i = t(u_i)$$

To put this into a more convenient form for solving:

$$t(u_i) - t_i = 0$$

Since the two-dimensional spline curve is ensured to be one-to-one, the one-dimensional function $t(u)$ is strictly increasing, and therefore there is a single value u_i which satisfies this equation. For a cubic spline curve, this solution can be determined analytically. An efficient implementation of a root finder for cubic polynomials can be found in [Press92]. Once the u_i value has been determined, we determine the graph value s_i:

$$s_i = s(u_i)$$

The entire process is illustrated in figure 3.4.

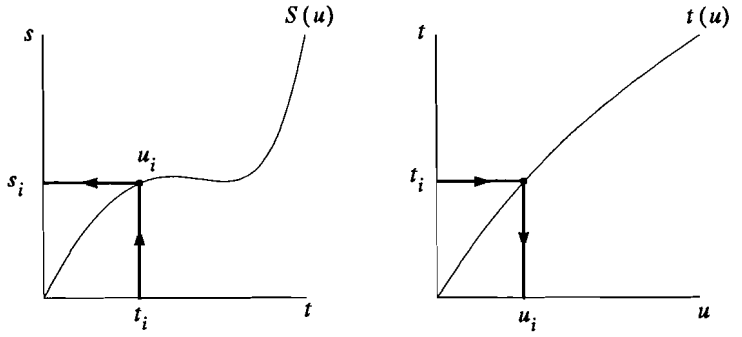


FIGURE 3.4. Interpreting a 2D spline curve as a graph. To determine the graph value s_i at t_i of the curve $Q(u) = (t(u), s(u))$ (left) we first determine the value for which $t(u_i) = t_i$ (right). We then evaluate $s(u_i)$ to determine s_i. This assumes that $t(u)$ is strictly increasing so that there is a unique u_i for each t_i.

A function curve usually consists of more than one spline segment. In this case we must first determine the cubic segment corresponding to the inquired t_i. Once again, we exploit the fact that the function $t(u)$ is strictly increasing and compare the desired t_i with the computed values of $t(u)$ at the start and end of each segment (t_j and t_{j+1}). When the segment for which $t_j \le t_i \le t_{j+1}$ is found, we solve for u_i and s_i as shown above. For splines that interpolate the first and last control points of each segment, the control points themselves can be used for comparison.

3.2 Constraint Based Solutions

All of our tools for motion control can be expressed as precise goals for the shape and derivative of the motion curve Q and the motion graph S. The most common solution to this type of problem is to express the goals as a series of mathematical constraints and solve the set of simultaneous equations. There is generally no unique solution to these problems. They are normally either under-constrained or over-constrained. For problems which are under-constrained, there exist an infinite number of solutions to the problem. For these problems, we must add additional qualifications to the constraints, for example, we find the solution that minimizes change in the shape of the curve, relative to its last

representation. For over-constrained problems, there is no solution to the set of simultaneous equations. In this case we try to find a solution that comes "closest" to satisfying our constraints. For our research, all of these problems can be expressed as under-constrained systems. This section describes analytic solutions to point constraints of position and derivative. Techniques for phrasing and solving more complicated sets of constraints vary greatly depending on the nature of the problem and are the subject of ongoing research in computer graphics and computer aided design[Gleicher91][Welch92][Moreton92].

3.2.1 Single Segment Constraints

Point Constraint

As a first case, consider a single point constraint on the graph S (figure 3.5). The initial value of S at time t is s_1, which indicates a distance along the spatial curve Q. The animator decides that she would like to modify this motion, so that the object instead travels the distance s_2 at time t. We must modify the shape of the curve S so that it passes through the point (t, s_2). This type of problem can be solved by expressing the constraint mathematically. In this case, the constraint is:

$$S(t) = s_2$$

We then find a new formulation of S so that this, and any additional constraints, are satisfied. In our particular case, this is an under-constrained problem whose result will dependent on the representation and shape of the curve S. To find a solution, we must express this problem more formally.

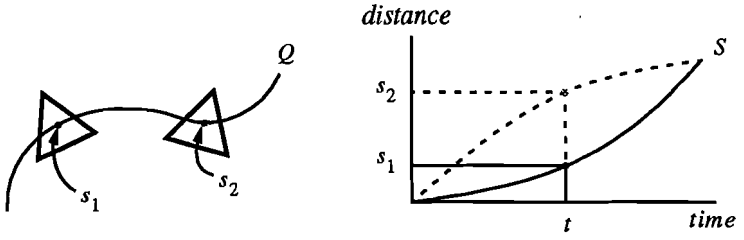


FIGURE 3.5. Single point position constraint. The triangular object originally reaches point s_1 at time t. The animator decides that point s_2 should now be reached at time t (left). In order to achieve this goal, the graph must be modified so that its curve passes through the point (t, s_2) while maintaining the values at the graph's endpoints (right).

To solve this point constraint, consider a curve S consisting of a single two-dimensional Bézier segment. The curve can be represented in matrix form as

$$S(u) = BP$$

where B is the Bézier basis matrix evaluated at u:

$$B = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The rows of the matrix P represent the control vertices:

$$P = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}$$

We know the parametric value u corresponding to the point on the curve and the amount by which we want to move the point (ΔS). Using techniques presented in [Bartels89] and [Fowler91] we can express our constraint as follows:

$$S(u) + \Delta S = B(P + \Delta P)$$

This equation reduces to

$$\Delta S = B\Delta P$$

which we must solve for ΔP, the change we need to make to the control points. Since we have an underdetermined system, we are actually trying to find a solution which minimizes the length of ΔP. The vector ΔP can be expressed in terms of mutually orthogonal components, one in the *row space* of B and the other in the *null space* of B:

$$\Delta P = B^T\lambda + z$$

We now seek to minimize this function, and observe that the system does not depend on z, and thus we should limit the search to those values of ΔP for which $z = 0$. By substituting back into the expression for ΔS, we now have

$$\Delta S = BB^T\lambda$$

solving for λ gives us

$$\lambda = (BB^T)^{-1}\Delta S$$

which we then substitute back into the expression for ΔP to yield our solution:

$$\Delta P = B^T(BB^T)^{-1}\Delta S$$

For interactive manipulation of a curve, the expression $B^T(BB^T)^{-1}$ can be thought of as a set of weights *W*. *W* need only be evaluated once for a given parameter value *u*. Direct manipulation of the curve then simply involves computing $W\Delta S$ with each incremental change to ΔS.

If we want to fix one of the control vertices, we need to force ΔP_i to be zero, where *i* corresponds to the *i*th control vertex. To achieve this we replace the *i*th column of *B* with zeros before evaluating *W*. Fixing the first and last control vertices of a graph segment will achieve a result similar to our initial example in figure 3.5. This is often the desired behavior, as we would like to modify the character of the motion between two points in time while maintaining the values at the endpoints.

Velocity Constraint

A specific goal for the derivative of the curve *S* can be described as a *velocity constraint*. For a given time *t* along the motion graph there is a corresponding parameter u_s along the motion graph *S* (represented as a two-dimensional parametric curve). The velocity at u_s is determined by the slope of *S* at that point. An animator provides a specific goal for the velocity of *S* at the given point. From their goal velocity, we determine the desired change in slope, ΔS_1, at the given point. We must then modify *S* to satisfy this constraint. Unfortunately, if we use a single constraint to modify the velocity, then the position at the given point is free to wander. In this case the position of *S* at the parameter value u_s may no longer correspond to the original value of *t* and our constraint will not be met at time *t*. So, instead we must construct a double constraint system which holds the position constant at u_s while modifying the slope (figure 3.6). To implement this velocity constraint

we have to solve the double constraint system:

$$\begin{bmatrix} 0 \\ \Delta S_1 \end{bmatrix} = \begin{bmatrix} B(u_s) \\ B'(u_s) \end{bmatrix} \Delta P$$

The top constraint holds the position at the parameter value u_s constant while the bottom constraint modifies the derivative at u_s by ΔS_1. Other combinations of constraints can be simultaneously maintained including higher derivative values, curvature and other properties of the curve. Analytic solutions to such double and triple constrained systems are described in [Fowler91].

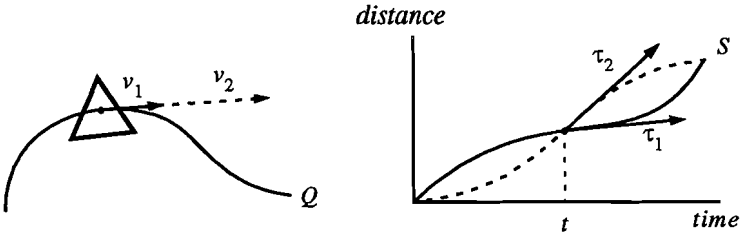


FIGURE 3.6. Constraint controlling velocity while maintaining position. Initially, the velocity at time t is v_1 : $S'(t) = v_1$. The animator decides that she wants the velocity at time t to be increased to v_2 (left, velocity is indicated by the length of the tangent vector). In order to achieve this goal, the graph must be modified so that its tangent at point t is altered from τ_1 to τ_2 so that $S'(t) = v_2$.

The most serious drawback of these single point constraints is that they only affect a fixed range of the curve. These formulations only allow modification of a single segment of the spline curve. If the motion graph has only a few segments, the animator's changes may be too broad. If the graph has many segments, then the changes made might be too fine and not blend well with the surrounding motion. In addition, the single point constraint technique has widely varying results depending upon how close the selected point is to the endpoints of the segment. Changes in the middle of the segment are well behaved, with the effect of the change falling off with distance from the selected point. If the selected point is close to one of the ends of the segment, however, large changes far from the point of influence must be generated so that the cubic segment passes through the new point (figure 3.7). We will later describe a method which allows arbitrary control of the range over which the constraint's changes are distributed. Additionally, this method

should result in displacements which fall off smoothly with respect to the distance from the selected point.

FIGURE 3.7. Unequal distribution of change with a single point constraint. Using least-squares spline manipulation over a cubic curve with its endpoints fixed, the magnitude of a change seems to fall off evenly with distance when the manipulated point is near the center of the curve (left). However, when the point is closer to one of the endpoints (right) the cubic segment can only pass through the goal position by making changes larger than the points's change on portions of the curve far from the point's position.

3.2.2 Multi-segment Constraints

There are many approaches to phrasing and solving constraints over multiple segments of spline curves and surfaces. Here, we will briefly examine two common approaches and judge their applicability to the satisfaction of our shape and derivative constraints for motion paths and motion graphs.

The first technique solves point constraints like those phrased in the last section, except allowing an arbitrary range of segments along the curve to be affected. This approach is presented in [Gortler94]. The process begins by solving for the single segment along which the point constraint lies. The technique then iteratively tries to minimize curvature (or some other measure of "goodness") over a broader range across the curve, while maintaining the point constraint (figure 3.8). The minimization is accomplished with any number of solution techniques, the simplest being a gradient search. The drawbacks of this approach become apparent when one attempts to shape a curve using successive applica-

tions of this technique. Since we minimize curvature, all fine details along the curve are lost with each application.

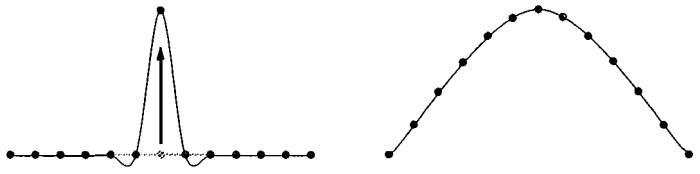


FIGURE 3.8. Constraint satisfaction using curvature property. The first step in satisfying this multi-segment constraint involves satisfying the point constraint over a single segment (left). The process then iteratively modifies the surrounding control points, minimizing curvature over the entire curve to achieve the final result (right).

The second technique is similar, but minimizes the change between the previous curve and the new curve. This method maintains the small details in the curve with each successive modification (figure 3.9).

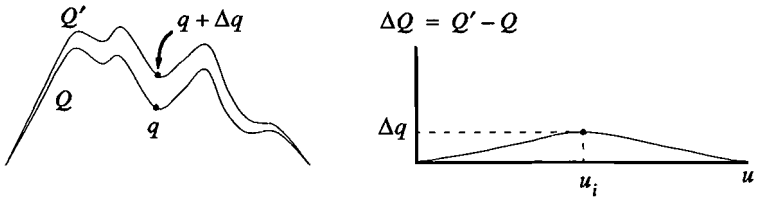


FIGURE 3.9. Constraint satisfaction using shape property. The point q on curve Q is moved, minimizing the change in shape of the curve (left). To minimize the change in shape we must simultaneously satisfy $\Delta Q\,(u_i) = \Delta q$ while maintaining the values at the edges equal to 0 and minimizing curvature over the change function ΔQ (right).

Both of these constraint techniques are extremely sensitive to the number and spacing of control points in the curve. To successfully use these techniques, we must ensure that the control points are evenly spaced over the length of the curve. In addition, the iterative solutions are time-consuming, vary in their accuracy, and are not guaranteed to converge.

3.3 Displacement Functions Applied to Space and Time Curves

As a more controllable method with guaranteed time complexity for real-time interaction, we consider applying displacement functions to motion curves. This technique eschews the complicated mathematics of constraints for a simple composition of func-

tions. In general, there is a displacement function F which when added to the motion path Q or the motion graph S to achieve our goals for position, shape and derivative.

Our initial approach was to represent F as a point change which we add to a curve γ. Consider the simple geometric goal that γ pass through point q at time t_0. We can define a new curve

$$\bar{\gamma}(t) = \gamma(t) + F(t)$$

Where F is defined to be 0 except at t_0 and

$$F(t_0) = q - \gamma(t_0)$$

This satisfies our goal, but destroys the continuity of $\bar{\gamma}$. We can smooth $\bar{\gamma}$ by repeatedly filtering with a box filter to increase continuity. However, this will also gradually remove any fine details from $\bar{\gamma}$. Instead of post-filtering $\bar{\gamma}$, we can *pre*-filter the displacement function F. If we smooth the point-change filter, the resulting function F now has a "total displacement" of 1, although the exact displacement at t_0 is now less than one. We have a smooth displacement function which no longer satisfies our constraint.

Using these ideas as a starting point, we attempt to construct F without any iterative convolution by representing F as a curve with the continuity properties of the filtered curve and the desired values at the constraint points. Although F will have a simpler shape than the convolved function, it satisfies our geometric goals.

3.3.1 Principles of Displacement Functions

The application of a displacement function is simple. The function $S(t)$ (here considered a one-dimensional function) representing distance vs. time is added to a displacement function $F(t)$ over the time interval $[t_\alpha, t_\beta]$, resulting in a modified version of S, S_f (figure 3.10):

$$S_f(t) = S(t) + F\left(\frac{t - t_\alpha}{t_\beta - t_\alpha}\right), \forall t : t_\alpha \le t \le t_\beta$$

The above function assumes that F is defined over the domain $[0, 1]$, so that the expres-

sion $\left(\frac{t-t_\alpha}{t_\beta - t_\alpha}\right)$ maps from the function's domain to $[t_\alpha, t_\beta]$. If the function S is represented as a digitally sampled curve with N samples within the interval $[t_a, t_{a+N}]$, such that $t_a = t_\alpha$ and $t_{a+N} = t_\beta$, we can sample F at the same frequency and rewrite the above equation as:

$$S_f(t) = \sum_{i=0}^{N} S(t_{a+i}) + F(t_i)$$

The drawback of this representation is that the high level controls afforded by splines are lost when we discretize the curve. This drawback can be remedied by promoting the discrete representation back to a spline representation. This can be achieved using two different methods. The first simply involves fitting a spline through every discrete point in the graph, but this is likely to introduce redundant control points along many sections of the curve. The second method involves using a curve fitting algorithm which finds the best spline curve interpolating the set of points within a given tolerance [Schneider90].

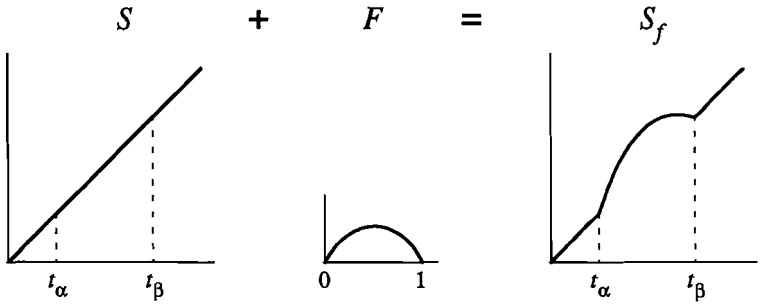


FIGURE 3.10. Using a displacement function to modify a curve. The displacement function F (middle) is used to precisely modify the shape of S (left) by scaling F over the range $[0, t_\beta - t_\alpha]$ adding F to S over the range $[t_\alpha, t_\beta]$. The resulting curve S_f is shown at right.

An alternate method of applying the displacement function does not involve any discretization of the original curve. This method is only applicable when both the original curve S and the displacement function F are represented by Bézier curves. Two properties of these curves allow us to directly add the control points of the splines to produce the sum of the two functions. These properties are:

1. The sum of two Bézier curves, Q_1 and Q_2, with equal number of control points is exactly equal to the curve obtained by adding their control points.

2. Subdivision of Bézier curves can be accomplished without affecting shape or continuity of the original curve.

Using these properties, and assuming that S and F both begin and end on interpolated points, we can simply subdivide F so that it has the same number of control points as the region of S being displaced, then add the control points of the two splines. The t values of the subdivision points must also correspond to the relative t values of the displaced region (figure 3.11).

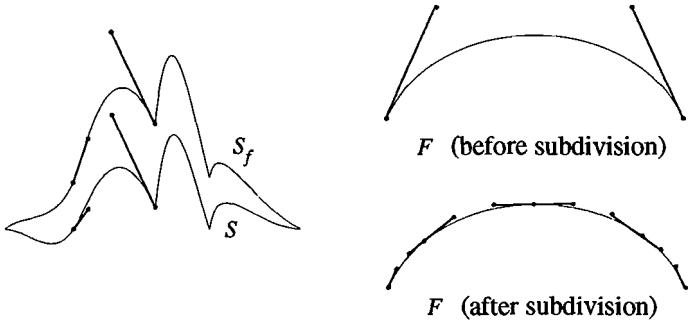


FIGURE 3.11. Applying a displacement function using Bézier segments. The Bézier curve at left consists of thirteen control points, four of which are shown. We want to convolve S with the displacement function F consisting of a four point Bézier segment (top right), resulting in $S_f = S + F$. This result can be achieved by subdividing F so that it has the same number of control points as S (lower right), then adding the control points of each spline together.

All of the methods for applying the displacement function preserve the continuity of the original curve S if the displacement function F is continuous to the desired order. The following sub-sections will describe the functions we use to implement our set of gestural tools, which maintaining arbitrary degrees of continuity.

3.3.2 Temporal Translation

Temporal translation is our term for the operation which satisfies the goal "Reach this point on the motion graph at this time". This technique for motion control maintains the shape of the motion path while altering the motion graph to satisfy a point goal. The positional goal is satisfied by compressing or expanding surrounding temporal information.

This amounts to translating $S(t)$ up or down at the point corresponding to the current time, while maintaining the continuity of S over the range being modified. An example of this method in action is shown in figure 3.12. To implement this operation, we construct a

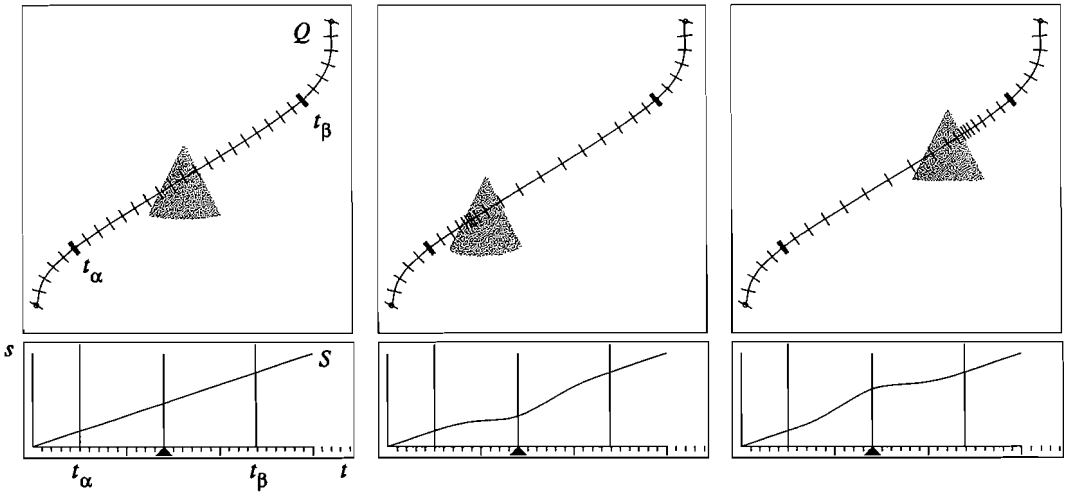


FIGURE 3.12. Temporal translation. Within the specified interval $[t_\alpha, t_\beta]$, the animator drags the cone object along the length of the motion path Q. The timing around the object changes to maintain the desired position at the current time. The motion graph S (below) shows the application of the displacement function F at the current time (indicated by the black triangle). Note that continuity with the surrounding motion is maintained.

displacement function F which will maintain continuity over the specified range and give a maximum displacement value of 1 at time t_i, relative to the start of the function F. These goals can be expressed mathematically as:

$$F(0) = 0$$

$$F(t_i) = 1$$

$$F(1) = 0$$

$$F'(0) = 0$$

$$F'(1) = 0$$

If higher degrees of continuity are desired, then the higher order derivatives at the end-points must also equal zero. The function is applied over the specified range of the motion graph, scaled by the desired displacement, Δs:

$$S_f(t) = S(t) + \Delta s F\left(\frac{t - t_\alpha}{t_\beta - t_\alpha}\right), \forall t : t_\alpha \le t \le t_\beta$$

We represent the displacement function F as a two segment Bézier curve. By adjusting the

tangents of the curve F, the displacement function's shape can be changed to achieve different types of interaction. The parameters k, b_1, and b_2 control the width and amount of blending at the two endpoints of the function. Each parameter can vary from 0 to 1, representing the minimum and maximum values the tangents can have for a given value of t_i (figure 3.13).

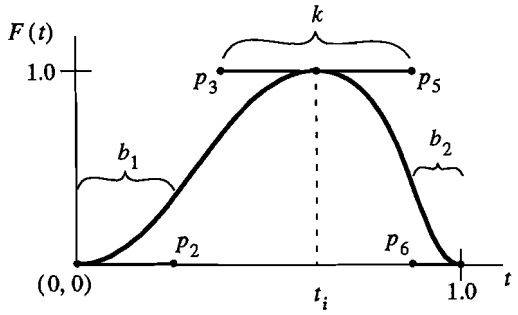


FIGURE 3.13. Construction of displacement function for position. This function is constructed with a two segment cubic Bézier curve. The maximum values for the parameters k, b_1 and b_2 are determined by the value of t_i and the values of the other parameters. In order to keep the curve one-to-one, the tangents values must remain in the quadrant whose corners are $(t_i,1)$ and either $(0, 0)$ or $(1, 0)$ for the first and second segments respectively. This cubic function guarantees C_1 continuity. For higher degrees of continuity, a higher order Bézier curve is required. However, the construction is identical.

3.3.3 Spatial Translation

The spatial translation method modifies the motion path to achieve the effect of directly manipulating the position of the animated object. Given a change in position for the object, Δq, a displacement function is constructed which will modify Q so that its position at time t passes through $Q(u) + \Delta q$, where u is the parameter value along Q at time t. The displacement function used is the same as that of figure 3.13, but applied to the motion path rather than the motion graph. We wish to have the displacement function fall-off with respect to time, rather than according to the normal parameterization of the curve. In this case the application of the function is slightly more complicated. The displacement function $F_t(t)$ is constructed as a function of distance vs. time. In order to apply this function to the motion path Q over the time interval $[t_\alpha, t_\beta]$ the function must be converted to a function of distance vs. u (the natural parameter to the curve Q) by transform-

ing first via the arc-length function A and the function of arc-length vs. time $S(t)$:

$$F(u) = F_t\left(\frac{S^{-1}(A(u)) - t_\alpha}{t_\beta - t_\alpha}\right)$$

This is a transformation from time to distance to parametric space. We cannot efficiently evaluate this function, so we must approximate F. We accomplish this by sampling F_t at even intervals of t, $t \in [t_0 \ldots t_n]$ where n is selected based on the number of frames in the interval. For each value of t we calculate the parameter value u and construct a one-dimensional spline $\tilde{F}$ from the set of points $F(u_0) \ldots F(u_n)$. Additional control points may be required at the ends of the spline so that the curve's derivatives equal 0 at the endpoints. The displacement function is applied in a manner similar to the temporal translation method:

$$Q_f(u) = Q(u) + \Delta \boldsymbol{q} k \tilde{F}\left(\frac{u - u_\alpha}{u_\beta - u_\alpha}\right), \forall u : u_\alpha \le u \le u_\beta$$

The constant k in this equation is a scaling constant which ensures that $\tilde{F}(u_i) = 1$, so that the position of the curve at u_i will exactly equal $Q(u_i) + \Delta \boldsymbol{q}$:

$$k = \frac{1}{\tilde{F}(u_i)}$$

Since this function changes the arc-length of the curve Q, the graph S must be scaled to maintain the character of the animation. If we wish to maintain the duration of the segment along Q being edited, then the graph must be adjusted so that the distance traveled within the interval $[t_\alpha, t_\beta]$ of S is modified to exactly equal the new distance from t_α to t_β along Q. This involves a scaling along the vertical axis of S. If we want to maintain velocity, then the graph S must be scaled horizontally in t, changing the duration of the

segment within the interval $[t_\alpha, t_\beta]$. Figure 3.14 shows the results of these two operations. The scaling operations are defined more precisely in the next section.

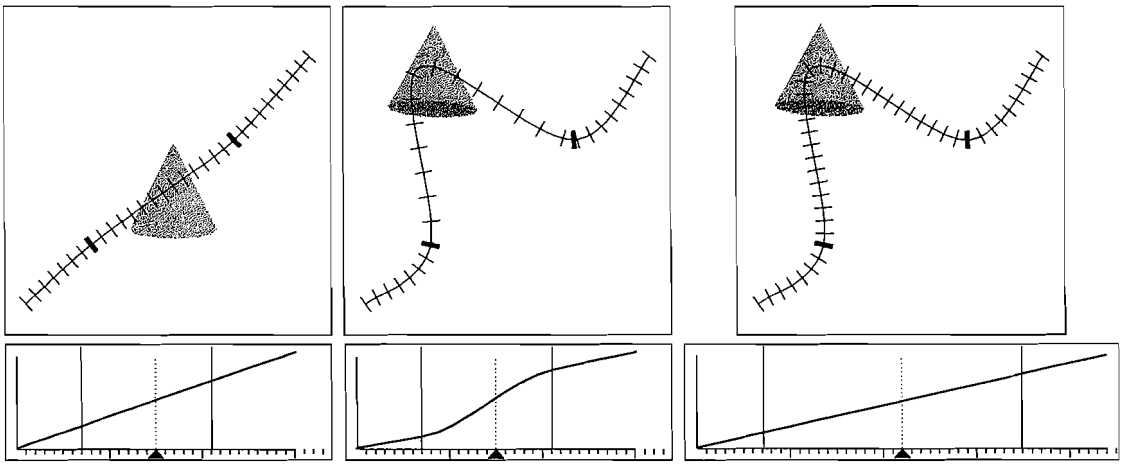


FIGURE 3.14. Spatial translation. By applying the point displacement function along the length of the curve Q, we can achieve direct manipulation of the animated object. At left is the original spatial path and graph. In the center, we have dragged the object to a new position by applying the point displacement function to the mouse delta. To maintain the duration of the segment being modified, we must scale the corresponding region of the graph (below middle). The graph segment must also be modified to restore continuity at its edges. To maintain the velocity of the segment, we must scale the interior segment by an amount proportional to the change in arc-length (right).

3.3.4 Scale Operations

The scale operations are more easily understood if we do not normalize the s-axis of the graph, as we have done in the previous section. In this case, the motion graph S may be seen to be growing in both axes as points are added in space and time, instead of the compression and expansion of the graph seen in figure 3.14.

Time/Arc-length Scale

The time/arc-length scale function is only applied in combination with the spatial translation method. Its purpose is to maintain the velocity along a segment of the curve given a change in arc-length Δs by varying the duration of the segment. We assume that the original segment lies within the range $[t_\alpha, t_\beta]$ and that we also wish to maintain continuity at the boundaries of this region. We need to change the duration by an amount proportional to the change in arc-length. The ratios of Δs to total arc-length (s_{end}) and Δt to

total duration (t_{end}) are thus equal:

$$\frac{\Delta t}{t_{end}} = \frac{\Delta s}{s_{end}}$$

So, given Δs we can easily compute Δt:

$$\Delta t = \frac{\Delta s \times t_{end}}{s_{end}}$$

We can then define the transformation to the new graph S_f from S:

$$S_f(t) = \begin{cases} S(t) & 0 \le t \le t_\alpha \\ S(t_\alpha) + \tau\left(S\left(t_\alpha + \frac{t - t_\alpha}{\tau}\right) - S(t_\alpha)\right) & t_\alpha \le t \le t_\beta' \\ S(t - \Delta t) + \Delta t & t_\beta' \le t \le t_{end}' \end{cases}$$

Where

$$\tau = 1 + \frac{\Delta t}{t_\beta - t_\alpha}$$

The diagram in figure 3.15 illustrates this transformation.

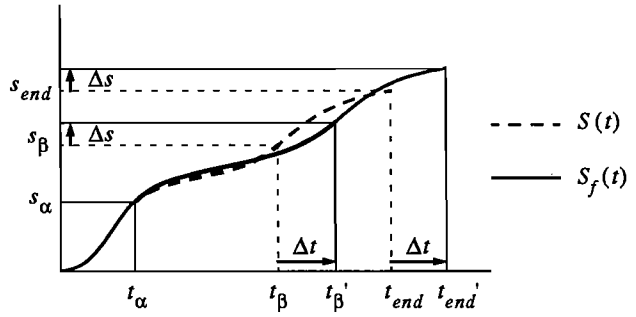


FIGURE 3.15. Time/arc-length scale. The curve S is transformed so that the segment in the region $[t_\alpha, t_\beta]$ is uniformly scaled, maintaining the velocity of the motion while expanding its duration. The segments at either end retain the same duration and velocity and connect smoothly with the scaled segment, since the derivatives at the ends of the scaled segment remain the same.

Time Scale

If we want to scale a segment temporally while its total arc-length remains fixed, then maintaining continuity becomes a more difficult problem. The chosen segment is scaled in time by τ and displaced in s by a function F which restores continuity over the region $[t_\alpha, t_\beta']$:

$$S_f(t) = \begin{cases} S(t) & 0 \le t \le t_\alpha \\ S\left(t_\alpha + \frac{t - t_\alpha}{\tau}\right) + F\left(\frac{t - t_\alpha}{t_\beta' - t_\alpha}\right) & t_\alpha \le t \le t_\beta' \\ S(t - \Delta t) & t_\beta' \le t \le t_{end}' \end{cases}$$

The displacement function F must be continuous to the desired degree and have the following properties:

$$F(0) = 0$$

$$F(1) = 0$$

$$F'(0) = S_f'(t_\alpha) - S'(t_\alpha)$$

$$F'(1) = S_f'(t_\beta') - S'(t_\beta')$$

We would also like some range along the interior of F to equal 0 so that a portion of the original motion is maintained. One choice for the representation of this function is shown in figure 3.16. The function has many degrees of freedom which determine the magnitude of the change and how quickly the motion blends into the surrounding regions. An additional constraint on the function might require that the resulting curve S_f is nondecreasing, which is discussed in section 3.3.6. We leave the details of constructing such functions from Bézier curves to the reader.

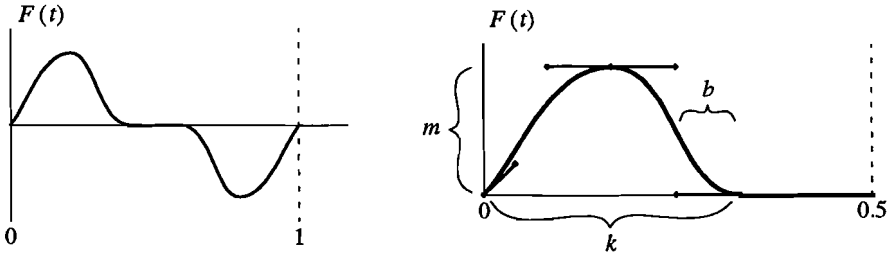


FIGURE 3.16. Time scale displacement function. The curve on the left shows one solution for the displacement function F which restores continuity to a scaled segment of the motion graph S. On the right we show the construction of the first half of this filter from Bézier segments. The parameters k, m and b are convenient abstractions to the animator, corresponding intuitively to the range of influence, the magnitude of change, and the amount of blend of the displacement function. The second half of the function is constructed similarly.

Arc-length Scale

An animator may wish to scale the amount of distance travelled over a specified segment, while maintaining the shape of the motion path and the total length of the animation.

This is accomplished by scaling the selected segment $[t_\alpha, t_\beta]$ in s and simultaneously compressing the surrounding segments in s. This is a simple variation on the previous methods. The new curve can be determined by the following equation:

$$S_f(t) = \begin{cases} \frac{S(t)}{\tau} & 0 \le t \le t_\alpha \\ S(t_\alpha) + \tau(S(t) - S(t_\alpha)) + F\left(\frac{t - t_\alpha}{t_\beta - t_\alpha}\right) & t_\alpha \le t \le t_\beta \\ S(t_\beta) + \frac{S(t) - S(t_\beta)}{\tau} & t_\beta \le t \le t_{end} \end{cases}$$

where the function F is identical to the time scale displacement function, applied over the range $[t_\alpha, t_\beta]$.

3.3.5 Velocity Control

The velocity function implements the intuitive controls "go faster", "go slower", or "reach a specific velocity". A displacement function is applied to the curve S at a specified point t in time, maintaining the shape of the underlying spatial curve and the duration of the temporal segment. An illustration of the velocity function in use can be seen in figure 3.17. This function is applied in a manner identical to the temporal translation func-

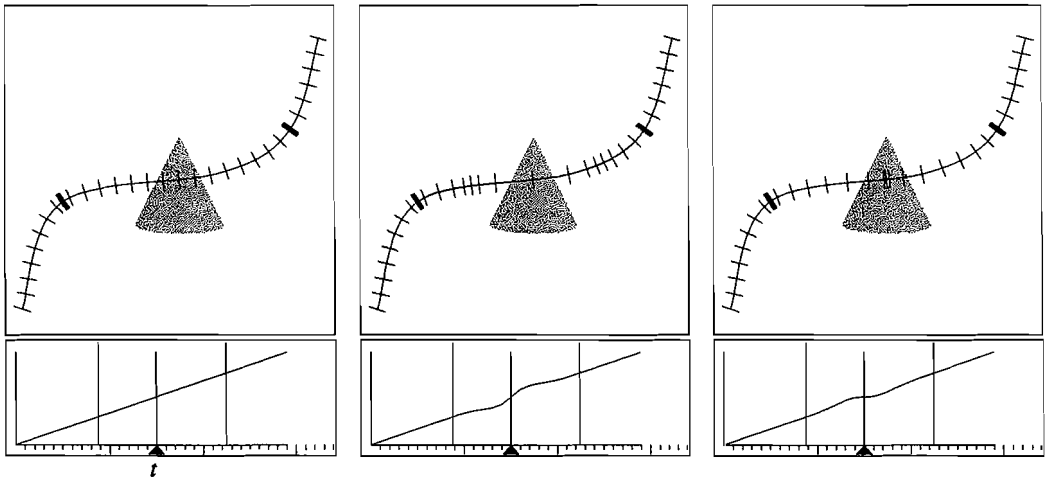


FIGURE 3.17. Velocity control. An object moving at a constant velocity is shown at left. By applying the velocity displacement function to the time graph, we can increase the speed at time t without changing the duration of the segment or the point reached by the object at time t. At right is an application of the function to decrease the velocity at the chosen point.

tion, modifying the graph only. The displacement function F must have the following properties:

$$F(0) = 0$$

$$F\left(\frac{t-t_\alpha}{t_\beta - t_\alpha}\right) = \Delta v$$

$$F(0) = 0$$

$$F(1) = 0$$

$$F'(0) = 0$$

$$F'(1) = 0$$

where Δv is the desired change in velocity of the graph S at point t. If higher degrees of continuity are desired at the boundaries of the modified region, then the higher derivatives at the endpoints of F must also be constrained to 0. Construction of this function is once again an under-constrained problem and we show our own construction of such function from Bézier curves in figure 3.18.

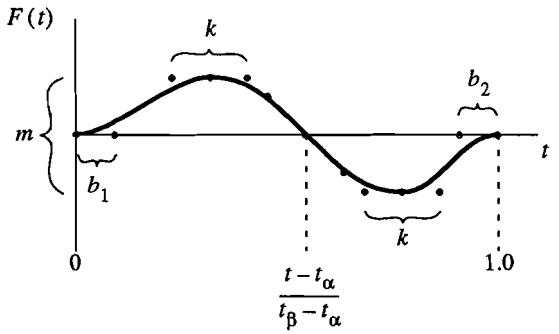


FIGURE 3.18. Construction of velocity displacement function. Four cubic Bézier segments are used to construct a velocity displacement function. The parameters k, m, b_1 and b_2 control the amount of influence, magnitude of change and amount of blending respectively. The actual range of these parameters are dependent on t and the values of all other parameters. The user is given a range between 0 and 1 for each parameter which is mapped to the actual minimum and maximum values.

3.3.6 Ensuring that Displaced Graphs are Increasing Functions

Ensuring that the displaced graph S_f remains strictly increasing is a simple problem. If the graph is represented as a series of discrete samples or as a Bézier curve, then we have to insure that successive samples or control points are strictly increasing:

$$\forall i,\ 1 \le i \le n-1\ :\ S_{f_x}(t_{i-1}) \le S_{f_x}(t_i) \le S_{f_x}(t_{i+1}) \wedge S_{f_y}(t_{i-1}) \le S_{f_y}(t_i) \le S_{f_y}(t_{i+1})$$

We compare the values along the curve as the displacement function is applied to S. If the function violates the strictly increasing condition we have several choices:

1. Clamp the values violating such condition. This has the drawback of destroying the continuity properties of the graph.
2. Stop the application of the displacement function and revert to the previous version of S.
3. Stop the application of the displacement function, adjust the function so that smaller changes in magnitude are applied at the regions of violation, and re-apply.

For our implementation we chose to stop applying the function when it violates the strictly increasing property and allow the animator to change the parameters affecting the shape of the displacement function F.

3.4 Direct Manipulation

To achieve direct manipulation we must maintain a one-to-one correspondence between a user's mouse motions and the changes made to objects in the visible scene. In practice, this amounts to some trigonometry which maps from raw mouse-deltas onto the objects in the scene to determine the underlying parameters which we are changing. These techniques are generically applicable to any of our motion control methods, using either the constraint-based solution or the displacement function solution. All of our methods require a specific change in position, time or velocity. This section discusses how to determine these specific values so as to maintain a direct correspondence between mouse motions and temporal changes.

3.4.1 Spatial Translation

Direct manipulation for spatial translation is straightforward. We use a method which allows manipulation of the object in a plane parallel to the film plane of the viewing camera. We project the mouse delta onto a plane parallel to the film-plane which passes

through the object being manipulated (figure 3.19). This gives us a vector in world-space, Δp, which can be applied using the techniques described in section 3.3.3.

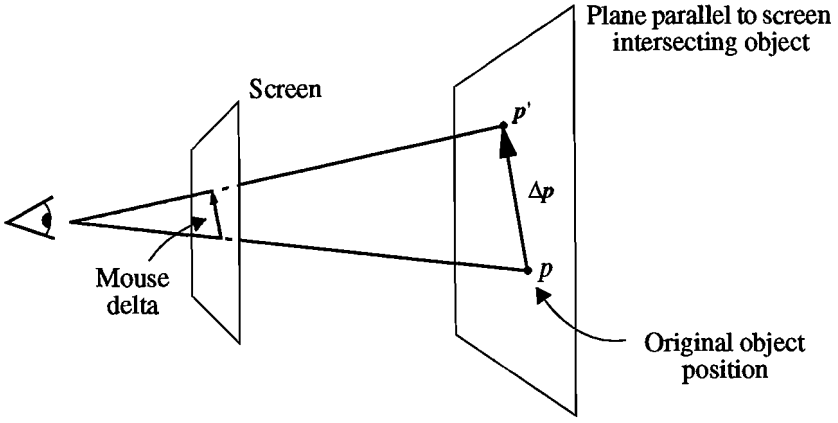


FIGURE 3.19. Mapping mouse delta to object delta.

Other methods for direct manipulation are possible, which we will only briefly describe here. The first projects the mouse deltas onto the three axes corresponding to the object-space basis of the object being manipulated. This allows manipulation in three spatial dimensions, in contrast to the parallel-plane method. However, it is very sensitive to the camera view and orientation of the object, sometimes making it impossible to move along a particular axis, or vacillating between choices of axis. A third method works similarly, but projects onto the world-space basis.

3.4.2 Temporal Translation

For the temporal translation function (section 3.3.2) we must determine Δs, the change in arc-distance for the point at time t. In this case, we determine the world-space vector Δp as before, then project the new object position p' back onto the curve Q, which gives us the point on the curve Q closest to the new mouse position. We find the parameter value u_{new}, corresponding to the parameter value of the projected point, then map this to an arc-distance along the curve. Subtracting the new arc-distance from the old, we arrive at Δs:

$$\Delta s = A(u_{new}) - A(u)$$

3.4.3 Velocity Modification

For the velocity controls we must determine Δv, the change in velocity at time t_i. In our interface, we would like the tick-marks surrounding the object position to track the mouse position, in the same way that the object tracked the mouse position during temporal translation. We can accomplish this by using the value of Δs, as computed in the previous method. We assume that the user clicks on the object, dragging in either direction along the curve to increase or decrease velocity. The tick at the next time step t_{i+1} should be moved by Δs. We can determine Δv from this information:

$$\Delta v = \frac{\Delta s}{t_{i+1} - t_i}$$

We can then use Δv with one of the methods for modifying the graph $S(t)$. Figure 3.20 shows the direct manipulation process.

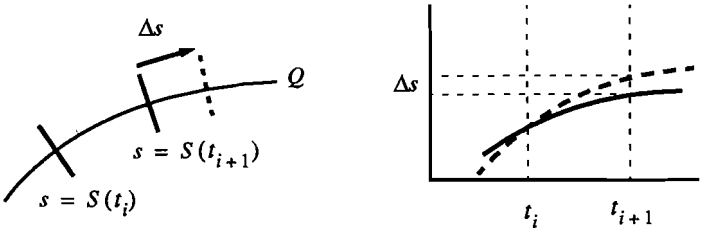


FIGURE 3.20. Mouse tracking for direct manipulation of velocity change.

3.4.4 Specification of Range

To specify the range $[t_\alpha, t_\beta]$ over which the displacement functions or constraints are applied, we allow the animator to place two bars along the length of the curve Q which indicate the start and end of the modified region. To manipulate these bars, we use a method similar to the direct manipulation technique for temporal translation. As the user drags on the bar, the mouse deltas are projected onto the point on the curve Q. This new point is then chosen as the new position for the bar, and the orientation of the bar is determined by the Frenet frame at this point. We also calculate the time-value t at the endpoints to specify the range of the motion graph S to be modified. To specify the start and end of the range, we color the two bars green and red, respectively.

This interface can become tedious when making many modifications to different objects along different curves. A simpler, but less flexible interface for range specification might allow the animator to choose a fixed range, which is automatically calculated whenever the animator changes the currently viewed time or the current object. For example, the animator might decide that she always wants a range of five frames about the manipulated temporal point to be modified.

3.5 Choice of Representation

Both the constraint and the displacement methods for motion control can be applied to a wide range of representations for the curves. For our research we chose to represent the spatial path of an object's motion with a Kochanek-Bartels spline with tension, continuity and bias all fixed at 0.0. This choice conveniently gives us a spline that interpolates its control points while maintaining first and second order continuity throughout the length of the spline. The disadvantage of this representation is that continuity is not easily broken when needed, and adding control points changes the shape of the curve. A more appropriate representation might be Bézier curves whose tangents are automatically adjusted to maintain continuity except where a user specifically request lower order continuity. Adding control points to a Bézier curve does not effect shape or continuity.

For the graph describing distance traveled along the curve vs. time we use a one-dimensional function of position vs. time. This function is discretely sampled at a resolution corresponding to the frame rate of the animation. The discrete sampling is required for the application of displacement functions over specific ranges within the function S. At times, the one-dimensional representation is promoted to a two-dimensional parametric spline curve fitting through the points defined by (time, position) pairs. This is required when scaling the motion, or when a higher-level hand adjustment of the curve is required. This two-dimensional representation is temporary, however. As soon as another application of a displacement function is required, the curve is again re-sampled and stored as discrete points.

3.6 Visualizing Temporal Change

We employ several simple techniques for visualizing temporal change in our system. The technique of drawing a point or line at equal intervals of time along the curve is the most basic, and is found in many commercial systems. Points are sometimes difficult to distinguish, and spacing between the points can be difficult to ascertain. Lines provide strong visual cues, but present a problem in three dimensions. If the lines are drawn at a fixed orientation relative to the curve, they are difficult to see from many viewing angles. A simple solution is to always draw the lines perpendicular to the film plane of the camera. Other solutions involve drawing three-dimensional objects at each point, such as a vector or a 3-dimensional axis. If we want to visualize changing orientation along the curve, an axis is particularly useful.

A more sophisticated visualization draws actual copies of the animated object at equal intervals of time along the curve. As time recedes in either direction, the transparency of the copies increases. This allows the most complete visualization by the animator, as all the information is visually present (figure 3.21). This representation can be visually cluttered and slow for complicated objects. One solution to this problem is to draw the animated object at wider intervals of time, for example every 1/4 second. We can also use a simpler version of the animated object for the copies, even resorting to bounding boxes.

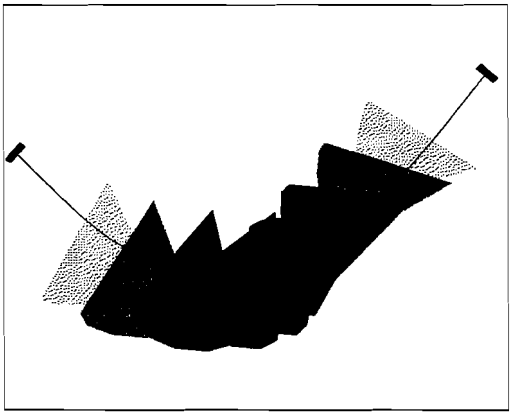


FIGURE 3.21. Ghosting used to visualize change over time. Instead of tick-marks, copies of the actual object are distributed at equal intervals of time along the curve. This gives a more accurate and complete picture of all the object's properties varying over time.

Other possible techniques could involve a hybrid of these processes. Instead of seeing entire copies of the animated object, we might only see only the leading or trailing edge of the object as it fades temporally. This technique is inspired by traditional animators' techniques (figure 3.22). We do not yet have a method for this type of visualization, but are looking at both image processing and polygonal approximations for possible solutions.

FIGURE 3.22. Hand-drawn visualization of temporal change. This drawing of Wile E. Coyote by Ken Harris [Schneider88] shows a sophisticated method for visualizing time-varying information. Suggestive line work indicates the essential features and boundaries of the hands at distinct moments of time. It is not necessary to draw the entire figure at each time step.

4.0 Evaluation of Gestural Tools

Our research presents a complete set of tools for visualizing and editing motion in a single two or three-dimensional view. By carefully applying these techniques using the principles of direct manipulation, the interface is made intuitive to the user. Every action has a distinct and reversible visual reaction directly proportional to the user's movements. The animator has precise control of the range over which an operation can be blended into the surrounding motion. This allows animators to refine motion at a high level, rather than constantly resorting to individual frame-by-frame adjustment. By re-parametrizing the motion curve by arc-length, animators can think and see in the natural terms of distance vs. time.

This research has applications outside of desktop animation systems. In the emerging world of virtual reality, the user does not have access to a keyboard or windowing system which are currently essential components of today's multiple-view animation systems. Using our techniques, the single stereo view is sufficient for visualizing *and* editing

motion at the same time. Direct manipulation is the only available tool in VR and our techniques are easily extended to accommodate a freely moving three-dimensional point of manipulation.

Motion capture and performance animation are steadily growing areas of computer animation. Our editing techniques are especially well suited to modifying this type of sampled motion, since it is naturally sampled at the frame-rate of the animation and is difficult to edit. We have experimented briefly with motion capture using our system. A segment of animation is captured, then converted to our space/time curve representation. Obtaining a spatial curve is accomplished by fitting a spline through the sampled spatial points to create the spatial path. The temporal curve is created by forward differencing the sampled points to determine distance travelled vs. time. The motion can then be edited using our tools.

Several problems are currently inherent to our system. The arc-length evaluation process is still a computational bottleneck. The accuracy of our current methods is not guaranteed, although they perform well in practice. Users must be made aware of the approximate nature of the motion in the same way that they now understand how a wireframe or low-resolution preview of their animation relates to the final rendered version. Choosing the correct displacement functions is currently more of an art than a science. We would like to provide simpler parameterization to the user or automatically determine the "optimal" displacement function, if there is such a thing.

Another problem is the spatial direct manipulation of the animated object. The application of a displacement function to the motion curve only has the desired effect when there are sufficient control points within the edited region. If the point of direct manipulation is far from any control point, the curve can behave unintuitively, overshooting the point of manipulation in a manner similar to that produced by constraint-based curve manipulation (figure 3.7). If a control point is added at the point of direct manipulation, this problem disappears; however we then face two other problems — adding control points changes the shape of some spline types, and the number of control points can

quickly become unmanageable. Large deltas may also distort the original motion curve in a manner unintended by the animator, destroying the small details of motion.

5.0 Future Work

We would like first and foremost to improve the representation of the spline and graph. For the spatial curve, we would like a spline parametrized directly by arc-length, or a faster and more robust method for approximating arc-length. We would like to find a spline type that allows the application of displacement functions to be expressed analytically so that the results of displacement are precise and continuous. As possible solutions to these problems we are considering curves represented as NURBS and other schemes using Bézier curves which automatically add and remove control points. A more radical solution might involve representing the curves as a one-dimensional oriented particle system with continuity properties automatically maintained at each "particle" along the curve. Geometric modeling using these techniques has already been explored by Richard Szeliski in [Szeliski93]. A wavelet representation for the curves is also possible, although preliminary results of wavelet spline manipulation show many of the same problems of other constraint-based techniques [Finkelstein94]. The most serious drawback is the difficulty of precisely specifying the range over which a modification has effect.

Although these techniques are easily applicable to other one-dimensional parameters such as color or single channels of scale/rotation, there are currently no methods to visualize and edit rotation or scale through direct manipulation. For rotation we are hopeful that a quaternion representation might be a fruitful means to visualize rotation. The rotation of the object could be represented as a path along the surface of a sphere surrounding the object, and tick-marks adjusted along this path. Scale might be visualized using additional paths showing the extent of the object, or some sort of handles extending from the points along the motion curve.

6.0 Acknowledgments

I would like to gratefully acknowledge the support of Andy van Dam, who allowed me the opportunity and time to pursue this research. My thesis advisor, John Hughes, pro-

vided many stimulating discussions and helped me develop the underlying mathematical techniques. My fellow student Cindy Grimm provided excellent feedback on several drafts of this thesis. I could not have completed this work without the support from all of them.

7.0 References

[Bartels89] R. Bartels and J. Beatty. A Technique for the Direct Manipulation of Spline Curves. In *Graphics Interface '89*, pp. 33-39, London, Ontario, 1989. Morgan Kaufmann Publishers, Palo Alto, California.

[Burden81] R. Burden, J. Faire, and A. Reynolds, *Numerical Analysis*, Prindle, Webster Schmidt, Boston, 1981.

[Chenoweth93] Chenoweth, Amelia. Personal Communication while at XAOS and ILM, California, 1993-1994.

[CoSA93] CoSA After Effects version 2.0. CoSA, a division of Aldus Corporation, Seattle, WA. 1993.

[ElectricImage93] Electric Image version 2.0. Pasadena, CA 1993.

[Finkelstein94] A. Finkelstein and D. Salesin. Multiresolution Curves. *Proceedings of SIGGRAPH '94*, In *Computer Graphics* Proceedings, Annual Conference Series, 1994, pp. 261-8.

[Fowler91] B. Fowler and R. Bartels. Constraint Based Curve Manipulation. *SIGGRAPH '91 course 25 notes, Topics in the Construction, Manipulation and Assessment of Spline Surfaces*, pp. 4.0-4.16, July 1991

[Gleicher91] Michael Gleicher and Andrew Witkin. Differential Manipulation. In *Graphics Interface '91*, pp. 61-67, Calgary, Alberta, 1991.

[Gortler94] Steven Gortler and Michael Cohen. Variational Modeling with Wavelets. *Princeton University Technical Reports*, 1994.

[Guenter90] B. Guenter and R. Parent. Computing the Arclength of Parametric Curves. *IEEE Computer Graphics and Applications*, **10**(3), pp. 72-8, May 1990.

[Kochanek84] D. Kochanek and R. Bartels. Interpolating Splines with Local Tension, Continuity, and Bias Control. *Proceedings of SIGGRAPH '84*, In *Computer Graphics*, **18**(3), pp. 33-41.

[Litwinowicz91] Peter C. Litwinowicz. Inkwell: A 2 1/2-D Animation System. *Proceedings of SIGGRAPH '91*, In *Computer Graphics*, **25**(4), pp. 113-122.

[Moreton92] Henry P. Moreton and Carlo H. Séquin. Functional Optimization for Fair Surface Design. *Proceedings of SIGGRAPH '92*, In *Computer Graphics*, **26**(2), pp. 167-176.

[Pasquale93] Pasquale, Joseph. Personal Communication at ILM, California. 1993.

[Press92] W. Press, S. Teukolsky, W. Vetterling and B. Flannery. *Numerical Recipes in C* Second Edition. Cambridge University Press, New York, 1992.

[Reeves90] W. Reeves, E. Ostby and S. Leffler. The Menv Modelling and Animation Environment. *Journal of Visualization and Computer Animation* Volume 1, pp. 33-40, 1990.

[Schneider88] Steve Schneider. *That's all Folks! The Art of Warner Bros. Animation.* Henry Holt and Co. New York, 1988.

[Schneider90] Philip J. Schneider. An Algorithm for Automatically Fitting Digitized Curves. In *Graphics Gems* edited by Andrew Glassner. pp. 612-626. Academic Press, San Diego, CA 1990.

[Steketee85] S. Steketee and N. Badler. Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control. *Proceedings of SIGGRAPH '85*, In *Computer Graphics*, **19**(3), pp. 255-62.

[Szeliski93] R. Szeliski, D. Tonnesen and D. Terzopoulos. Curvature and Continuity Control in Particle-Based Surface Models. In *SPIE Geometric Methods in Computer Vision II*, 1993. Society of Photo-Optical Instrumentation Engineers.

[TDI93] TDI Explorer version 3.02. Culver City, CA 1993.

[Watt92] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques.* ACM Press. New York, 1992.

[Welch92] William Welch and Andrew Witkin. Variational Surface Modeling. *Proceedings of SIGGRAPH '92*, In *Computer Graphics*, **26**(2), pp. 157-166.