A Direct Manipulation Interface for 3D Computer Animation

Scott Sona Snibbe[†]

Brown University
Department of Computer Science
Providence, RI 02912, USA

Abstract

We present a new set of interface techniques for visualizing and editing animation directly in a single three-dimensional scene. Motion is edited using direct-manipulation tools which satisfy high-level goals such as "reach this point at this time" or "go faster at this moment". These tools can be applied over an arbitrary temporal range and maintain arbitrary degrees of spatial and temporal continuity.

We separate spatial and temporal control of position by using two curves for each animated object: the *motion path* which describes the 3D spatial path along which an object travels, and the *motion graph*, a function describing the distance traveled along this curve over time. Our direct-manipulation tools are implemented using *displacement functions*, a straightforward and scalable technique for satisfying motion constraints by composition of the displacement function with the motion graph or motion path. This paper will focus on applying displacement functions to positional change. However, the techniques presented are applicable to the animation of orientation, color, or any other attribute that varies over time.

CR Descriptors: I.3.7 [**Computer Graphics**]: Three Dimensional Graphics and Realism; I.3.6 [**Computer Graphics**]: Methodology and Techniques; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling.
Additional Keywords and Phrases: Animation, Interaction Techniques, Splines.

1 Introduction

Kinematic computer animation is a painstaking process requiring hand adjustment of hundreds of key positions for every object in an animated scene. Most animation systems provide precise control of motion using two-dimensional graphs of individual parameters (e.g. x translation vs. time). Animators must mentally integrate this 2D information with static 3D views and occasional motion previews to maintain a clear sense of the motion which they are creating. This research is an attempt to develop interactive techniques for visualizing 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 on animating the position 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 section 9.

1.1 Problems in Existing Animation Systems

Several problems found in a majority of commercial and research animation systems serve 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 over time is through 2D graph editors. The 3D scene view is used primarily for viewing and editing an object at a single point in time.

[†]The author is currently at Adobe Systems Incorporated, 411 First Avenue South, Seattle, WA 98122 USA. ssnibbe@adobe.com.

- 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 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 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. The animator is forced to cancel out the timing induced by the spline before creating the desired motion.

- 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.

- 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 [2][14]. 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. Adding control points can introduce undesired complexity to the animation and reduces the range over which changes have effect.

- 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). 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—there are no tools for modifying multiple keyframes simultaneously. Many animators find it faster to re-do the animation from scratch in this situation.

1.2 Goals for Animation Control

The following set of goals is an attempt to describe an animation system which addresses the above set of problems:

- Create an system which allows visualization and editing of temporal and spatial information in a single 3D view.
- Express motion goals in terms of distance or velocity vs. time.
- Maintain temporal and spatial continuity while editing animations.
- Allow an arbitrary range over which editing tools are applied.
- Develop motion control techniques which are natually extensible to orientation, scale and any other animated parameters.
- Provide real-time performance for complex scenes.

As an interface to the above goals, we require direct-manipulation tools which correspond to the high-level goals of an animator:

- *Temporal translation*

  Satisfies the goal “Reach this point at this time” while maintaining the shape of the motion path, but changing the speed at which the object travels along the given path.

- *Spatial translation*

  Satisfies the above goal by modifying the spatial curve while maintaining either the duration or velocity of the given segment.

- *Temporal scale*

  Changes the duration of segment of animation. Satisfies the goal "Make this segment of animation longer, shorter, or a specific duration".

- *Velocity modification*

  Satisfies the goals "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.

2 Prior Work

There is a large body of previous published research on kinematic animation and motion control. This section only addresses those prior models which incorporate splines to control spatial interpolation and motion control.

Kochanek and Bartels invented a spline which interpolates a set of spatial control points and allows temporal control via high-level parameters at each control point [6]. Their model allows visualization and editing motion within a single view. Temporal adjustments are made using 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 parametric spacing around each control point. The effects of these parameters is sometimes intuitive, changing the shape and timing in a manner consistent with many naturalistic motions. However, 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. Additionally, modifications to the shape of the curve only has an effect on the two adjacent spline segments—Affecting a larger or smaller range involves moving additional points or adding more control points to the curve. This method of animation control is still extremely popular in computer animation and is present in many of today's commercial systems [2][14].

Steketee and Badler [13] published a method for separating temporal and spatial controls 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 object attributes. The composition of the two curves results in the final value for a given attribute. A drawback of their system is that the spatial 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 composite 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 graphs of single parameters vs. time for object attributes [9]. Key values along these curves are connected by spline segments. The shape and type of these splines can be modified in a piecewise manner, choosing the best shape and type for each segment of time. The system has the drawback that each animated parameter (e.g. x position, z rotation, y scale) must be edited individually, and timing curves can only be modified at their control points by using their tangents. The range of operations is also limited to a single spline segment. However, the system was developed in cooperation with traditional animators who find it a convenient and precise way to specify and visualize motion. The Pixar model is implemented in many commercial animation systems today [12].

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 [7]. The filters, applied over an arbitrary range of motion, provide high level control of the overall character of an animation. Animators can modify the gain, decay or oscillation of their digitally sampled motion curve by filtering with an infinite impulse response filter. 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 with an arbitrary range surrounding the modified point. The drawbacks of their system are the lack of a direct manipulation interface to the filtering and blending, the separation of motion into one-dimensional channels and the inability to satisfy precise goals using the high level filters.

A large number of researchers are pursuing techniques for the direct manipulation of spline curves and surfaces over arbitrary ranges. Least-squares techniques [4], constraint-based techniques [1][3][17] and oriented particle systems [15] are the major areas currently being explored. These techniques are all either highly compute-intensive, sensitive to the number and spacing of control points, or do not allow precise control of the effective range of constraints. Other specific drawbacks of applying constraints to satisfy displacement functions are discussed in section 7.

3 Separating Spatial and Temporal Controls

In order to satisfy our independent temporal and spatial goals, we represent an object's changing position over time by two curves. The first curve, Q, is the *motion path* describing a path through space along which an object travels. Q is represented as a parametric function of u:

$$Q(u) = \langle x(u), y(u), z(u) \rangle$$

The second curve, S, is the *motion graph*, a function of distance vs. time which maps from a time value t to a distance traveled along the motion graph, s:

$$S(t) = s$$

We want to use the graph S to determine the parametric position along Q at a given time t. Our direct manipulation tools can then be phrased as geometric constraints on either S or Q.

Since the curve Q is parameterized by its natural parameter, u, and the motion graph gives us distance values in terms of arc-length (s), we must create a mapping from s to u for the curve Q. This problem has no analytic solution, but can be solved numerically. A complete discussion of this problem and several approximate solutions can be found in [5] and [16]. In this paper we will assume for simplicity that we have a function A which maps from u to s: $A(u) = s$, and that the u value for a particular arc-distance s can be determined by $A^{-1}(s)$. Using this equation, we can now express the position of an object as a function of time (figure 1):

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

In practice, a velocity curve $V(t)$ is more commonly used as the motion graph. This enables animators to more easily visualize subtle changes in velocity. The curve $V(t)$ can be integrated without difficulty to determine $S(t)$. For ease of manipulation, the graph S is often represented as a two-dimensional parametric curve, rather than a implicit function of t. Animators find this representation easier to manipulate and capable of finer control of motion with fewer control points [9]. The details of interpreting a two-dimensional spline curve as an implicit function, and assuring that it remains one-to-one are discussed in [11]. For simplicity we will refer to S as a one-dimensional implicit function throughout the rest of this paper.

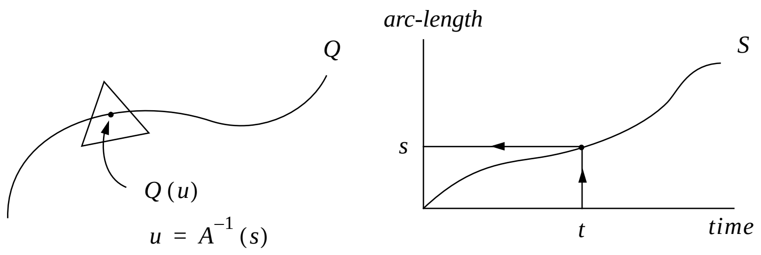

Figure 1: Motion control with separate space and time curves.

4 Tools for Motion Control

Our direct manipulation tools require that constraints such as “reach this point at this time” or “increase velocity at this instant” are satisfied. We can express these constraints as mathematically precise goals for the shape and derivative of the motion graph S and the motion path Q. In addition to the specific constraint at a single point in time, we also wish to blend the change smoothly into an arbitrary range surrounding the current point in time. Previous approaches to this problem proceed by first satisfying the hard constraints, then relaxing (in the case of constraints) or filtering (in the case of digital signal processing) to blend the change into surrounding areas of the curve. We choose a simpler approach that preserves the fine details of the curve and allows precise control over the range. We introduce a *displacement function* F which represents a pre-filtered goal which, when composed with a curve γ, results in the constraints being satisfied *and* smoothly blended over the specified interval. Composing the curves F and γ can be a non-trivial problem when the function F is expressed in terms other than the natural parameterization of the curve γ. Section 7 discusses several techniques for the satisfaction of these goals in such cases.

Given a displacement function $F(t)$ which represents the desired change in the curve $\gamma(t)$ over the time interval $[t_\alpha, t_\beta]$, we can express the resulting curve $\bar{\gamma}(t)$ as the addition of the two functions:

$$\bar{\gamma}(t) = \gamma(t) + F\left(\frac{t - t_\alpha}{t_\beta - t_\alpha}\right), \forall t : t_\alpha \leq t \leq t_\beta \tag{1}$$

The above function assumes that F is defined over the domain $[0, 1]$ so that F can be easily applied to different time intervals. No matter which method we use to satisfy this goal, the resulting curve $\bar{\gamma}$ will preserve the continuity of the original curve γ if the displacement function F is also continuous to the desired degree.

The following sub-sections show how we construct displacement functions to satisfy a spatial or temporal goal. In the illustrations of these techniques, we show the graph S along with the 3D scene. However, keep in mind that the animator does not need to see or edit the graph in order to visualize and modify the animation. In the next section, we show how these tools are applied using the principles of direct-manipulation in a 3D view.

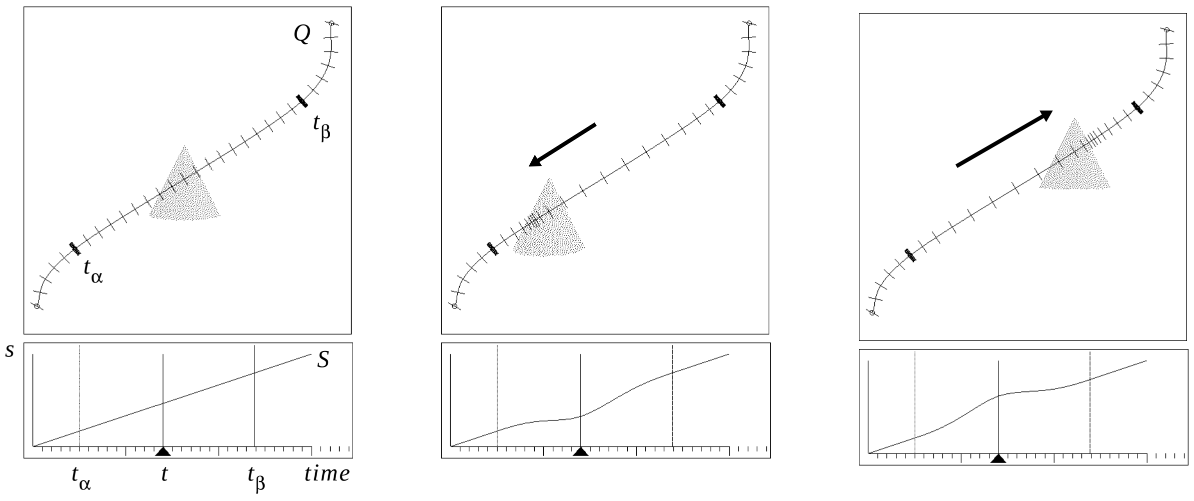

Figure 2: Temporal translation. Within the specified interval $[t_\alpha, t_\beta]$, the animator drags the cone object along the length of the motion path Q. The timing around the object changes to maintain the desired position at the current time (middle, right). The motion graph S, below each image, shows the application of the positional displacement function F at the current time (indicated by the black triangle).

4.1 Temporal Translation

The *temporal translation* technique for motion control allows the animator to change the position of an object at time t to a new point along the motion path Q. This positional goal is satisfied by 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 (figure 2). To implement this operation, we construct a displacement function F which will maintain continuity over the specified range and give a maximum displacement value of 1 at time t. These goals can be expressed mathematically as:

$$\begin{aligned} F\left(\frac{t-t_\alpha}{t_\beta - t_\alpha}\right) &= 1 \\ F(0) &= 0 \\ F(1) &= 0 \\ F'(0) &= 0 \\ F'(1) &= 0 \end{aligned}$$

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

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

We can 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 qualities 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 (figure 3).

4.2 Spatial Translation

The *spatial translation* method achieves the effect of directly manipulating the position of an object by modifying the underlying motion path. Given a change in position for the object, $\Delta\mathbf{q}$, a displacement function is constructed which will modify Q so that its position at time t passes through $Q(u) + \Delta\mathbf{q}$, where u is the parameter value along Q at time t. The displacement function used is the same as that used for temporal translation, 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 with the natural parameterization of the curve. In this case, the application of the function is slightly more complicated. The displacement

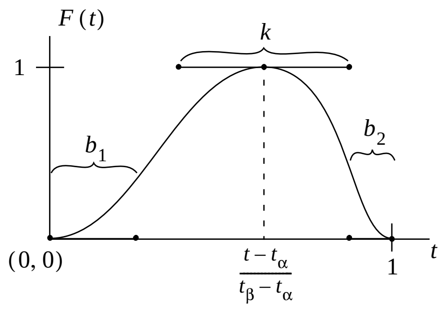

Figure 3: Construction of positional displacement function

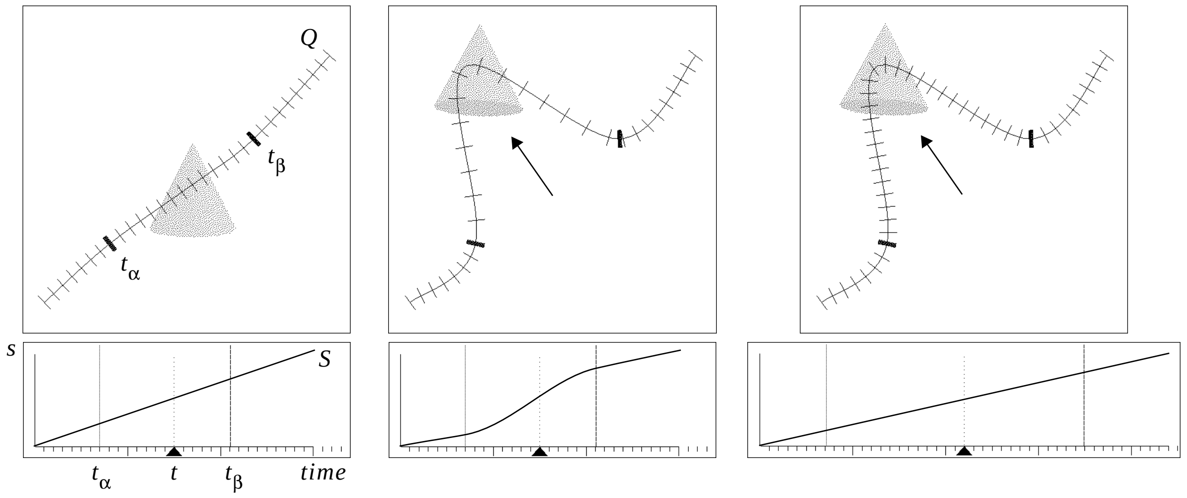

Figure 4: Spatial translation. By applying the point displacement function along the length of the curve Q, we can directly manipulate the animated object. At left is the original spatial path and graph. In the next two images, we have dragged the object to a new position by applying the point displacement function to the mouse delta. Applying a scaling filter allows us to either maintain the duration of the segment (middle) or maintain the velocity of the segment (right).

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 $[t_\alpha, t_\beta]$, F must be converted to a function of distance vs. u (the natural parameter to the curve Q) by transforming via the arc-length function A, and $S(t)$:

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

This is a transformation from time to distance to parametric space. In practice, this function cannot be efficiently implemented, since there are no analytic expressions for either S or A. A fast approximation of F can be determined 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 can then be applied in a manner similar to the temporal translation method:

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

Where u_α and u_β are the parameter values along Q corresponding to times t_α and t_β. The constant k is a scaling constant which ensures that $\tilde{F}(u_i) = 1$, so that the position of the curve Q at u_i will exactly equal $Q(u_i) + \Delta\mathbf{q}$:

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

Since the addition of F and Q changes the total 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 edited segment along Q, then the graph must be adjusted so that the distance traveled within the interval $[t_\alpha, t_\beta]$ of S is modified to equal the new distance from t_α to t_β along Q. This involves a scaling along the vertical axis of S. If instead, we want to maintain velocity, then the graph S must be scaled horizontally in t, changing the duration of the segment within the interval $[t_\alpha, t_\beta]$. These scaling operations are described in the following section. Figure 4 shows the results of these two operations.

4.3 Scale Operations

We present three scale operations in this section. The first two methods (*time/arc-length scale* and *arc-length scale*) are only used in conjunction with the spatial translation method to maintain velocity or duration. The third method (*time scale*) is applied directly by the animator to change the length of an animation segment.

4.3.1 Time/Arc-length Scale

The time/arc-length scale function is applied in combination with the spatial translation method. Its purpose is to maintain the velocity along a segment of the motion path given a change in arc-length Δs by varying the duration of the segment. In simpler terms, an animator drags the animated object through space and wants the velocity at the given time to remain the same, and the surrounding motion to maintain the same character, while allowing the duration of the entire segment to vary. We assume that the original segment lies within the range $[t_\alpha, t_\beta]$ and that we 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. Since the ratios of Δs to total arc-length (s_{end}) and Δt to total duration (t_{end}) are equal, we can easily compute Δt and then calculate the new version of S, S by uniformly scaling the interior segment:

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

Where

$$\tau = 1 + \frac{\Delta t}{t_\beta - t_\alpha} \qquad \widehat{t_\beta} = t_\beta + \Delta t \qquad \widehat{t}_{end} = t_{end} + \Delta t$$

4.3.2 Arc-length Scale

An animator may wish to scale the arc-distance traveled over a specified segment while maintaining the shape of the motion path and the total length of the animation. This situation occurs while directly-manipulating the object's spatial position. Since the arc-distance travelled changes as the animator drags the object, and the animator wishes to maintain the duration of the segment, the velocity of motion along the motion path must be changed. In this case, maintaining continuity becomes a more difficult problem. We must scale the selected segment $[t_\alpha, t_\beta]$ in s and simultaneously compress the surrounding segments in s, then displace using a function F which restores continuity over the selected region. The new curve can be determined by the following equation:

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

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

$$\begin{aligned} F(0) &= 0 \\ F(1) &= 0 \\ F'(0) &= \bar{S}'(t_\alpha) - S'(t_\alpha) \\ F'(1) &= \bar{S}'(\widehat{t_\beta}) - S'(t_\beta) \end{aligned}$$

One choice for the representation of this function, constructed from four Bézier segments, is shown in figure 5. We provide the high-level parameters k and m which correspond intuitively to the range of influence and the magnitude of change for the displacement function.

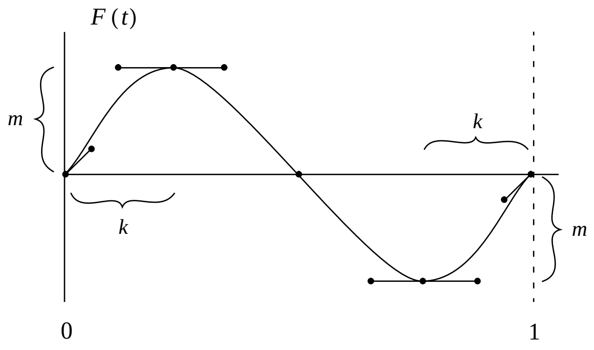

Figure 5: Construction of time scale displacement function

4.3.3 Time Scale

The most common scale operation is simply to change the duration of a segment without modifying the motion path, so that the total arc-length remains fixed. The chosen segment is scaled in time by τ and displaced in s by a function F which restores continuity over the region $[t_\alpha, t_\beta]$:

$$\bar{S}(t) = \begin{cases} S(t) & 0 \leq t \leq t_\alpha \\ S\left(t_\alpha + \frac{t-t_\alpha}{\tau}\right) + F\left(\frac{t-t_\alpha}{\widehat{t_\beta}-t_\alpha}\right) & t_\alpha \leq t \leq \widehat{t_\beta} \\ S(t-\Delta t) & \widehat{t_\beta} \leq t \leq \widehat{t}_{end} \end{cases}$$

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

4.4 Velocity Control

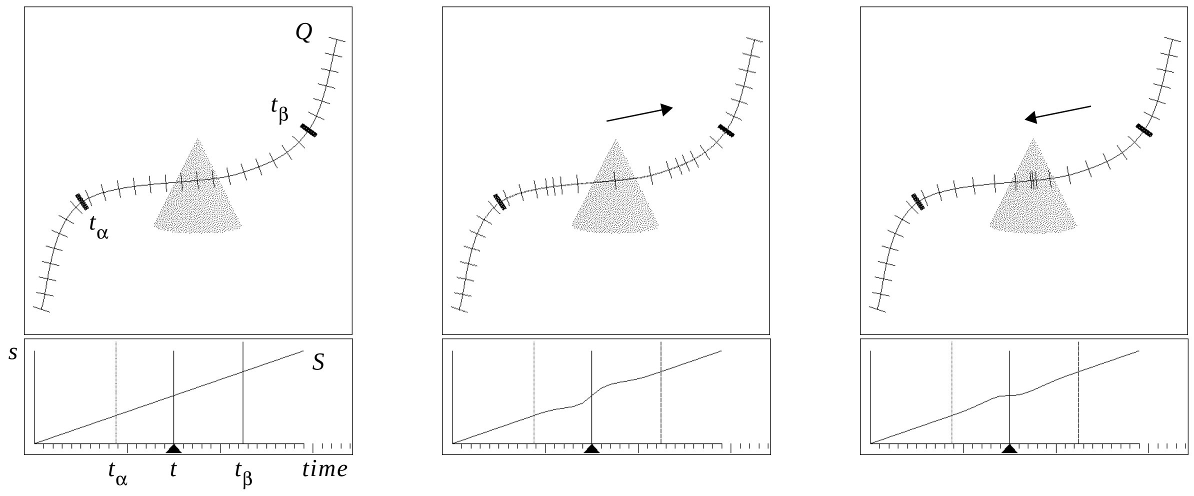

Figure 6: Velocity Control. An object moving at a constant velocity is shown at left. By applying the velocity displacement function to the time graph (middle, right), we can change the velocity at time t without changing the duration of the segment or the position of the object at time t.

The *velocity control* method modifies the motion graph to achieve a specific velocity goal at a given time. A displacement function is applied to the curve S at the 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 6. This function is applied only to the graph S, maintaining the shape of Q. The displacement function F must have the following properties:

$$\begin{aligned} F\left(\frac{t - t_\alpha}{t_\beta - t_\alpha}\right) &= \Delta v \\ F(0) &= 0 \\ F(1) &= 0 \\ F'(0) &= 0 \\ F'(1) &= 0 \end{aligned}$$

Where Δv is the desired change in velocity of the graph S at point t. We show our own construction of the velocity control displacement function from Bézier curves in figure 7.

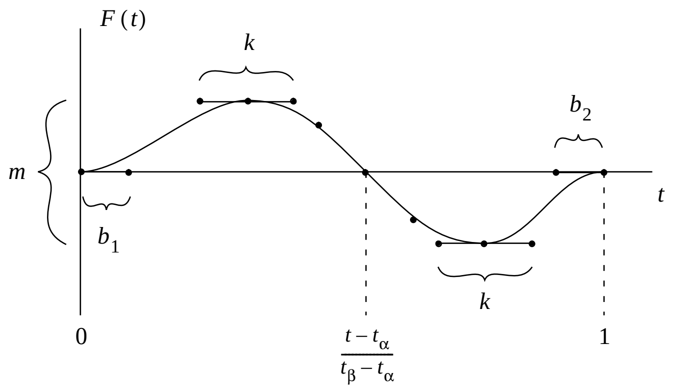

Figure 7: Construction of velocity displacement function

5 Applying Motion Tools Using 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 [8]. Suprisingly, many current modeling and animation systems still do not maintain such correspondence for their editing tools. In practice, the direct-manipulation process amounts to a bit of trigonometry which maps from camera-space mouse-deltas to the 3D space of the objects in the scene. In our case, we must determine a specific change in position, time or velocity from a user's mouse motion, which we then apply using the previous algorithms.

5.1 Direct Manipulation for Spatial Translation

Direct manipulation for the spatial translation tool is straightforward. We use a method which allows manipulation of the object in a plane parallel to the film plane of the viewing camera. Suppose the initial position of an object is at $\mathbf{q}$ (figure 8). We project subsequent mouse deltas onto a plane parallel to the film-plane which passes through the point $\mathbf{q}$, by calculating the intersection of the ray through the new screen-space position with the film-plane. This gives us the new point in space $\mathbf{q}'$. By subtracting $\mathbf{q}$ from $\mathbf{q}'$, we can determine the vector, $\Delta\mathbf{q}$, which we use in the application of the displacement function.

Other methods for direct manipulation are possible, which are only briefly described 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.

5.2 Direct Manipulation for Temporal Translation

For the temporal translation tool, we must determine Δs, the relative change in arc-distance for the object at time t. We assume that a user has selected the object to manipulate, and wishes to drag the object along the motion path by applying the temporal translation function to the motion graph. We know the parametric position, u at which the object initially lies along Q. As the user drags in camera-space, we compute the closest point on the path Q to the ray formed by extending the

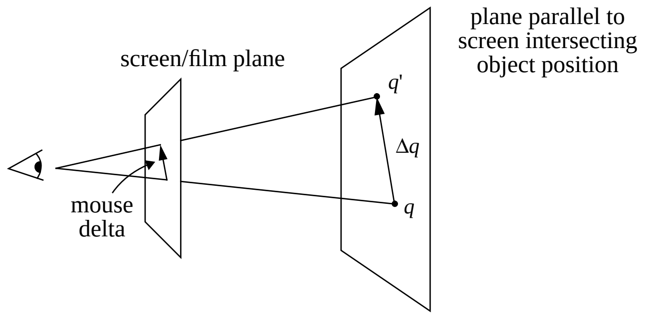

Figure 8: Mapping mouse position to object position

eyepoint through the point on the film plane (this is the same ray we used to determine q' in the previous technique). Using the parameter value u_{new}, corresponding to the new point on the path, we can now calculate the change in arc-distance with the arc-length function A:

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

5.3 Direct Manipulation for Velocity Control

For our direct manipulation interface to velocity control, we would like the tick-marks surrounding the object position to track the mouse position in the same way that the object tracks the mouse position during temporal translation. We can accomplish this by using the value Δs, as computed in the method for temporal translation. We assume that the user clicks on the object, viewed at time t_i and drags 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, the discrete change in velocity from this information (figure 9):

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

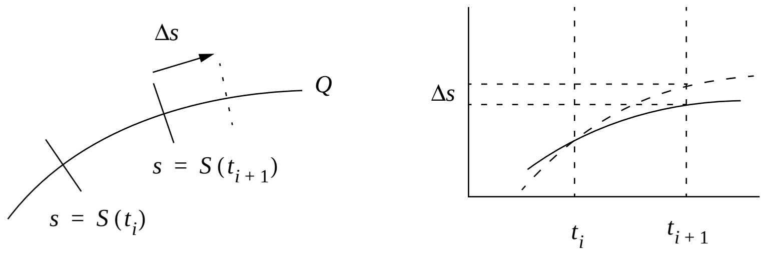

Figure 9: Mouse tracking for direct manipulation of velocity

5.4 Specification of Range

To specify the range $[t_\alpha, t_\beta]$ over which the displacement functions 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, we determine the closest point on the curve Q to the mouse position. 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 of Q at that point. We can then calculate the time-value t at the endpoints to specify the range of the motion graph S to be modified. To indicate the start and end of the range, we color the two bars green and red, respectively.

This interface can become tedious when making multiple 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 current time or object. For example, the animator might decide that she always wants a range of five frames about the manipulated temporal point to be modified.

6 Visualizing Temporal Change

We employ several techniques for visualizing temporal change in our system. The simplest technique involves drawing a point or line at equal intervals of time along the motion path and is found in many commercial systems. Points are sometimes difficult to distinguish in a complicated scene. We chose to use short lines or "tick-marks", which provide stronger 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 certain 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, a disk or a 3D axis. If we want to visualize changing orientation along the curve, an axis is particularly useful.

As a more accurate visualization, we can draw 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 10). The animator can easily edit the object at different points in time, by simply clicking on the version of the object to modify. This representation can become 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.

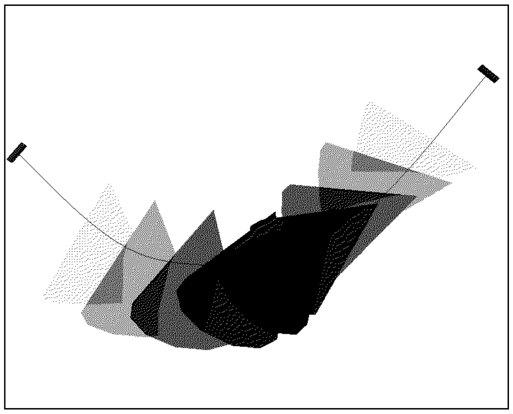

Figure 10: Ghosting used to visualize change over time

7 Implementation of Displacement Functions

A popular and natural approach to satisfying geometric goals for spline curves is constrained optimization. However, we encountered several major problems when we attempted to use this method to implement our displacement functions. The first drawback is that we cannot always represent the constraint as an analytic function, since the constraints are phrased in terms of arc-length or time rather than the natural parameterization of the splines Q and S. We instead have to approximate our constraints with a series of point goals. Our second problem is the range over which the constraints are applied. The boundaries of this region are limited to the control points along the curve. Finally, the time to arrive at a solution using constrained optimization is too great for interactive manipulation and the final solution is only approximate. For a more detailed discussion of these problems see [11].

We instead use two more predictable and mathematically less complex methods. The first method works when the spline to be displaced is represented as a digitally sampled curve with N samples within the interval $[t_\alpha, t_{\alpha+N}]$ such that $t_{\alpha+N} = t_\beta$. We can sample F at the same frequency and rewrite equation 1 as

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

This is the method we currently use for modifying the motion graph S. 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 [10]. Finally, we can apply this technique directly to the control points of an interpolating spline to achieve an approximate solution to our goal, which is the solution we choose when we modify Q.

The second 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 the 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 and spacing of control points as the region of S being displaced, then add the control points of the two splines.

8 Results

Our research presents a complete set of tools for visualizing and editing motion in a three-dimensional scene. Every user-interface 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-parameterizing the motion curve by arc-length, animators can think and see in the natural terms of distance vs. time. We believe our set of direct manipulation tools is simple to learn and easy use, although we require furthur user studies to determine this with more confidence.

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, a single stereo view is sufficient for visualizing and editing motion at the same time. Direct manipulation is the most natural means of interacting in VR and our techniques are easily extended to accommodate a freely moving three-dimensional point of manipulation, rather than a two-dimensional cursor.

Motion capture and performance animation are steadily growing areas of computer animation. Our editing techniques are especially well suited to modifying sampled motion of this type, since the motion data 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 traveled 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 a major computational bottleneck. If we approximate arc-length too coarsely in order to speed interaction, then the results may not precisely match the final animation. 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.

Our biggest 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. 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 changes may also distort the original motion curve in a manner unintended by the animator, destroying the small details of motion.

9 Future Work

We would like first and foremost to improve the representation of the motion path and motion graph. For the motion path, we would like a spline parameterized 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 [15]. 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 [3]. The most serious drawback is the difficulty of precisely specifying the range over which a modification has effect.

Although our techniques are easily applicable to other one-dimensional parameters such as color or single channels of scale and rotation, there are currently no methods to visualize and edit rotation or scale over time through direct manipulation. For rotation we are hopeful that a quaternion representation might be a fruitful visualization scheme. 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 by using handles extending from the points along the motion curve.

Instead of drawing entire "ghost" copies of the animated object to visualize temporal change, we are considering methods to visualize the leading or trailing edge of the object as it fades temporally. This technique is inspired by traditional animators' techniques for motion blur. 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.

Acknowledgments

I would like to gratefully acknowledge the support of Professor Andy van Dam, who allowed me the opportunity and time to pursue this research. Professor John Hughes provided many stimulating discussions and helped me develop the underlying mathematical techniques. Cindy Grimm, Brook Conner and many other members of the Brown Graphics Group provided excellent feedback on several drafts of this paper. This research was sponsored in part by NASA, the NSF/ARPA Science and Technology Center for Computer Graphics and Scientific Visualization, Sun, Autodesk, Taco Inc., NCR, HP, IBM, DEC, Apple, Microsoft and Adobe Systems.

REFERENCES

[1] Richard H. Bartels and John C. Beatty. A technique for the direct manipulation of spline curves. In *Proceedings of Graphics Interface '89*, pages 33–39, June 1989.

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

[3] Adam Finkelstein and David Salesin. Multiresolution curves. In *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 261–268, July 1994.

[4] Barry Fowler and Richard Bartels. Constraint-based curve manipulation. *IEEE Computer Graphics and Applications*, 13(5):43–49, September 1993.

[5] Brian Guenter and Richard Parent. Computing the arc length of parametric curves. *IEEE Computer Graphics and Applications*, 10(3):72–78, May 1990.

[6] D. H. Kochanek and R. H. Bartels. Interpolating splines for keyframe animation. In *Graphics Interface '84 Proceedings*, pages 41–42, 1984.

[7] Peter C. Litwinowicz. Inkwell: A $2\frac{1}{2}$-D animation system. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 113–122, July 1991.

[8] Gregory M. Nielson and Dan R. Olsen, Jr. Direct manipulation techniques for 3D objects using 2D locator devices. In *Proceedings of 1986 Workshop on Interactive 3D Graphics*, pages 175–182, 1986.

[9] Samuel J. Leffler William T. Reeves Eben F. Ostby. The Menv modelling and animation environment. *Journal of Visualization and Computer Animation*, 1(1):33–40, August 1990.

[10] Philip J. Schneider. An algorithm for automatically fitting digitized curves. In Andrew Glassner, editor, *Graphics Gems*, pages 612–626. Academic Press, San Diego, CA, 1990.

[11] Scott S. Snibbe. Gestural controls for computer animation. Master's thesis, Brown University, Department of Computer Science, 1994.

[12] Softimage version 2.66. Montreal, Canada, 1994.

[13] Scott N. Steketee and Norman I. Badler. Parametric keyframe interpolation incorporating kinetic adjustment and phasing control. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 255–262, July 1985.

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

[15] 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.

[16] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques: Theory and Practice*. Addison-Wesley Publishing Company, 1992.

[17] William Welch and Andrew Witkin. Variational surface modeling. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 157–166, July 1992.