1.4

Scripted Bodies and Spline-Driven Animation

Kenny Erleben and Knud Henriksen,
Department of Computer Science,
Copenhagen University

kenny@diku.dk
kaiip@diku.dk

ON THE CD *Additional material or source code for this article is included on the companion CD-ROM.*

In this article, we present some ideas and a theory for extending classical spline driven animation, such that it can be applied in a rigid body simulator. The need for specialized behavior arises because of the scripted bodies in dynamic simulation. In dynamic simulation, a scripted body is used to avoid simulating the physically correct trajectories of the body. This is computationally favorable if one knows that the body is practically unaffected by the physical interactions of the other bodies in the configuration. In this case, the error is negligible. An example of this is a vibratory part singulator, which shakes small parts into recesses for automated assembly. The trajectory of the vibrator is unaffected by the physical interaction with the smaller parts. When dynamic simulation is used in animation, we imagine that scripted bodies would be even more interesting because they provide the animator with the means of constraining a body to move in a physically unexpected or exaggerated way. However, all other bodies will interact in a physically meaningful way with the scripted body.

Scripted bodies move very much like traditional objects known in animation. One specifies a trajectory by defining some key positions at certain points in time. The objects have to move along this trajectory, "hitting" the key positions at the exact point in time where they are defined.

There are mainly two difficulties that arise. First, there is the general problem of reparameterization of the splines into their natural parameters (this is a general problem in spline-driven animation and not specific for dynamic simulation). Second, because scripted bodies can interact with physical bodies in a simulator, we need to know something about their motion, not just about their positions and orientations. The interactions typically involve computation of "time of impact", sweeping volumes, collision impulses, and contact and constraint forces. Thus, we need several different time derivatives, specifying the motion of the object. These are:

$$\vec{v}(t), \vec{a}(t), \vec{\omega}(t) \text{ and } \vec{\alpha}(t) \tag{1.4.1}$$

See Table 1.4.1 at the end of this article for notation explanation.

That is the linear and angular velocity and acceleration. Putting it all together we are looking for a function $Y(t)$, which specifies the motion of a scripted body in a simulator, such that,

$$Y(t) \rightarrow \left\{\vec{r}(t), \vec{v}(t), \vec{a}(t), q(t), \vec{\omega}(t), \vec{\alpha}(t)\right\} \tag{1.4.2}$$

Analogous to rigid bodies, this function can be seen as the state function of a scripted body. We define the scripted motion problem as finding the Y-function. In this article, we have concentrated on handling the linear motion only. However, with small changes, the techniques can be applied to handle the rotational part as well [Erleben02b]. We do not cover how to handle interactions with scripted bodies in a dynamic simulator or dynamic simulation as such [Baraff01].

Motivation

We have been working with dynamic simulation for a little while. When we began to embark upon the task of adding scripted bodies to our simulator, we were surprised to see that most textbooks and papers on the subject either simplified the problem by assuming that the state functions of the scripted bodies were known, or other approaches using implicit functions and such were taken.

In either case, we felt that the theory we had encountered was not very easy for an animator to use compared to systems such as 3ds max™, Rhino, or Maya®. We think that dynamic simulation has a potentially huge application area in animation, and it would be a shame not to pursue this application area. We wanted to extend/twist the traditional animation techniques, which are intuitive to use and well known by most animators, so that they could be used in a dynamic simulator.

Another benefit that arises from our work is a unification of scripted body motion. Typical simulators have specialized algorithms and implementations to take care of each kind of scripted motion type: implicit functions, sinusoidal waves, polynomials, etc. Spline-driven animation makes the specialized algorithms for different types of movement superfluous. When implementing a simulator, it justifies looking at scripted motion as a black box making use of a not needed to know state function $Y(t)$, and make it easier to implement simulators.

The Basic Idea

Cubic splines are a good choice for specifying the trajectory of the linear motion, because they are twice differentiable. This means that the motion along the spline is "smooth." It also means that the velocity and acceleration can be found by direct differentiation of the cubic spline. Our preferred choice is cubic nonuniform B-splines [Watt92] [Hoschek93]. This class of splines is easily converted into a composition of cubic curve segments (such as cubic Bezier curves [Erleben02a]), and they support nonuniform key positions.

If we have a space spline $C(u)$, then it is parameterized in the global parameter u. We use the space spline to find points in space given a value u, that is:

$$C(u) \to \left(x, y, z\right). \tag{1.4.3}$$

However, the parameter u is an unnatural parameter. It is difficult to predict exactly which point on a spline corresponds to a given value of u. It is better to think in terms of the arc length, s, instead of the spline parameter u. Therefore, we like to use a reparameterization like:

$$U(s) \to u \tag{1.4.4}$$

such that we have

$$C\left(U(s)\right) \to \left(x, y, z\right). \tag{1.4.5}$$

This reparameterization makes sense, because there is a one-to-one mapping between the parameter u and the arc length parameter s. This is due to the fact that if

$$u_1 < u_2 \tag{1.4.6}$$

then

$$S(u_1) < S(u_2). \tag{1.4.7}$$

Here, the S-function is the arc length function, and it is the inverse of the U-function.

Another difficulty occurs if an animator uses a space spline. In this case, he would be interested in specifying how fast an object moves along the space spline. In other words, he might not want the object to move as the parameter u would dictate. Instead, he wants to specify a set of (t,s) pairs, such that when the animation arrives at the time t, then the object would have travelled the distance s along the space spline. These pairs of time and distance are usually represented by a so-called velocity spline [Watt92]. We call this the "travelling distance."

$$V(v) \to (t, s). \tag{1.4.8}$$

Looking at Equation 1.4.4 we see that Equation 1.4.8 is not really useable. We are interested in twisting the equation so that we can find an arc length function, which maps from the time domain to the arc length.

$$S(t) \to s. \tag{1.4.9}$$

The velocity curve has to satisfy the property that there exists exactly one s-value for every possible t-value. Otherwise, the motion that is dictated by the velocity curve would be impossible. It would require an object to be at two positions at the same time. This means that the graph of V must be a function of t. Putting it all together we see that the C-function is reparameterized in the following way:

$$C(U(S(t))) \to (x, y, z). \tag{1.4.10}$$

This final reparameterization allows an animator to work intuitively with the space curve in terms of its arc length. He can specify movements in terms of the animation time parameter, t, totally independent of the global parameter u.

We cannot derive analytical functions for Equations 1.4.4 and 1.4.9, at least not in the framework where C and V are both cubic splines. We have to look for a numerical solution of Equation 1.4.10. Things get more complicated when we look for the first and second derivatives of Equation 1.4.10. By applying the chain rule we get:

$$\frac{dC(U(S(t)))}{dt} = \frac{dC}{du}\frac{dU}{ds}\frac{dS}{dt} \tag{1.4.11}$$

and

$$\frac{d^2C(U(S(t)))}{dt^2} = \frac{d^2C}{du^2}\left(\frac{dU}{ds}\right)^2\left(\frac{dS}{dt}\right)^2 + \frac{dC}{du}\frac{dU}{ds}\frac{d^2S}{dt^2} + \frac{dC}{du}\frac{d^2U}{ds^2}\left(\frac{dS}{dt}\right)^2. \tag{1.4.12}$$

Knowing that we work with cubic nonuniform B-splines (or cubic curve segments), the terms:

$$\frac{d^jC(u)}{du^j} \quad \text{for} \quad j = 0, 1, 2 \tag{1.4.13}$$

are easily computed, but the remaining terms in Equations 1.4.11 and 1.4.12 are somewhat more difficult to compute, because we cannot find analytical representations for the U- and S-functions.

The Linear Scripted Motion Function

Now let us address the problem of determining the values of the functions $U(s)$ and $S(t)$, given s- and t-values and the difficulties in computing their derivatives.

The Arc Length Function

If we have a cubic curve (for instance a cubic Bezier curve) in 3-dimensional space:

$$C(u) = \begin{bmatrix} x(u) \\ y(u) \\ z(u) \end{bmatrix} = \begin{bmatrix} a_x u^3 + b_x u^2 + c_x u + d_x \\ a_y u^3 + b_y u^2 + c_y u + d_y \\ a_z u^3 + b_z u^2 + c_z u + d_z \end{bmatrix} \tag{1.4.14}$$

or in the matrix notation:

$$C(u) = M\begin{bmatrix} u^3 \\ u^2 \\ u \\ 1 \end{bmatrix} = MU, \tag{1.4.15}$$

then it can easily be derived that the arc length function of the cubic curve is defined by the following function:

$$\begin{aligned} S(u) &= \int_0^u \sqrt{\frac{dC(u)}{du} \cdot \frac{dC(u)}{du}} du \\ &= \int_0^u (Au^4 + Bu^3 + Cu^4 + Du + E)^{1/2} du\,. \end{aligned} \tag{1.4.16}$$

Where (in 3-dimensional space):

$$\begin{aligned} A &= 9(a_x a_x + a_y a_y + a_z a_z) \\ B &= 12(a_x b_x + a_y b_y + a_z b_z) \\ C &= 6(a_x c_x + a_y c_y + a_z c_z) + 4(b_x b_x + b_y b_y + b_z b_z) \\ D &= 4(b_x c_x + b_y c_y + b_z c_z) \\ E &= (c_x c_x + c_y c_y + c_z c_z). \end{aligned} \tag{1.4.17}$$

Unfortunately, we cannot find an analytical expression for this integral so we have to do a numerical integration in order to find the value of $S(u)$ given a u-value. Let us introduce the function:

$$s(u) = (Au^4 + Bu^3 + Cu^4 + Du + E)^{1/2}. \tag{1.4.18}$$

This is the arc length integrand (see Appendix B in [Erleben02b]) used in the definition of $S(u)$. By applying Horner's rule for factoring polynomials we can rewrite the equation into a more computationally friendly form:

$$s(u) = \sqrt{(((Au + B)u + C)u + D)u + E}\,. \tag{1.4.19}$$

We can choose any numerical integration routine, which fulfils our requirements of performance and accuracy, and then apply it in order to compute the arc length, $S(u)$.

Arc Length Reparameterization

In this section, we want to address the problem of determining the reparameterization function U, such that

$$C(U(s)) \rightarrow (x(s), y(s), z(s)). \tag{1.4.20}$$

The function $U(s)$ is obviously the inverse of the arc length function $S(u)$:

$$U(s) = S(u)^{-1}. \tag{1.4.21}$$

In the previous section, we saw that $S(u)$ could only be computed numerically, therefore it is impossible to invert it to obtain the function $U(s)$. Fortunately, we can compute $S(u)$, so we can turn our problem into a root-finding problem. Given a value, s, we search for a value, u, such that

$$| s - S(u) | < \varepsilon . \tag{1.4.22}$$

If we find such a u-value, then we have actually found $U(s)$ (within the numerical tolerance). Recall that we have a one-to-one monotonic relation between the parameter u and the arc length s. This property ensures that exactly one root exists and our problem has a solution. Figure 1.4.1 shows the method for arc length reparameterization.

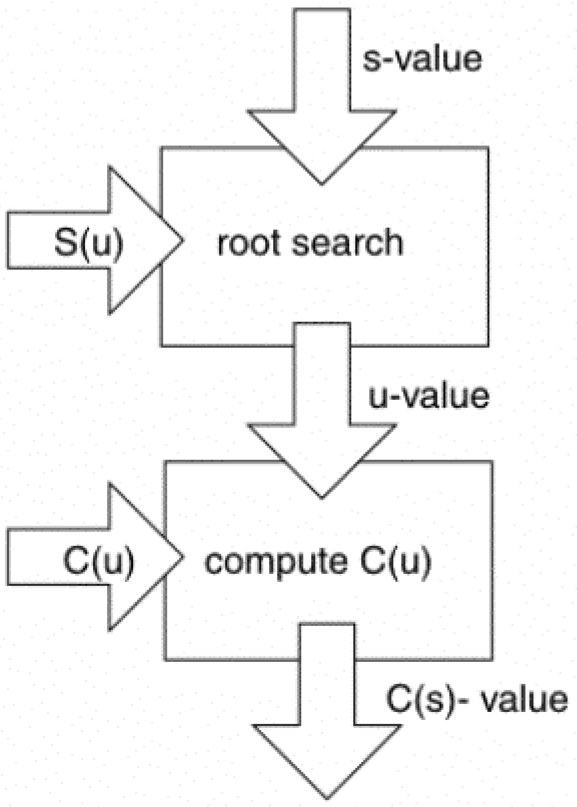

FIGURE 1.4.1 *Schematic drawing of the method for arc length reparameterization.*

Time Reparameterization

In an animation, one usually knows a *t*-value and wants to find the corresponding *s*-value. As we have explained previously, this sort of information is typically described by using a so-called velocity spline such as:

$$V(v) \to (t(v), s(v))\,. \tag{1.4.23}$$

From this, we need to find a function:

$$S(t) \to s\,. \tag{1.4.24}$$

The problem is very similar to the arc length parameterization problem. The main difference is that this time we are looking for a corresponding coordinate and not a "measure" of the spline. However, the problem can still be solved in a similar fashion. That is, as a root-finding problem. Given a *t*-value, we search for a *v*-value such that:

$$|\, t - t(v) \,| < \varepsilon \;\;. \tag{1.4.25}$$

When we find such a *v*-value, we can compute the *s*-value directly as *s*(*v*). The time of the animation is strictly an increasing function as the animation evolves, so our root-finding problem is guaranteed to have one unique solution. Notice that the velocity spline has to be physically meaningful, as we have explained earlier. This guarantees that we can always find a unique value for *s* (given a *t*-value). Figure 1.4.2 shows method for time reparameterization.

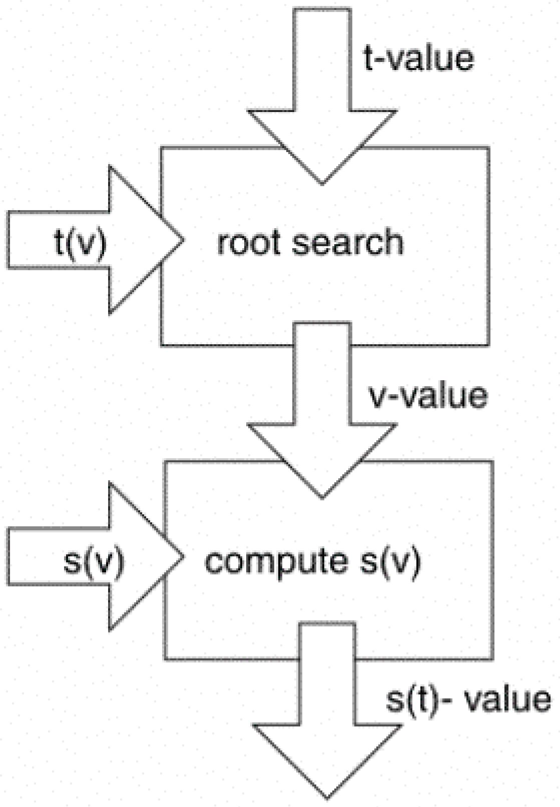

FIGURE 1.4.2 *Schematic drawing of the method for time reparameterization.*

Computing the Derivatives

From Equations 1.4.11 and 1.4.12, we see that we need to compute the following derivatives

$$\frac{d^2C}{du^2}, \frac{dC}{du}, \frac{d^2U}{ds^2}, \frac{dU}{ds}, \frac{d^2S}{dt^2} \text{ and } \frac{dS}{dt}. \tag{1.4.26}$$

We already know how to handle the derivatives of *C*, with respect to *u*. This leaves us with the remaining four derivatives:

$$\frac{d^2U}{ds^2}, \frac{dU}{ds}, \frac{d^2S}{dt^2} \text{ and } \frac{dS}{dt}. \tag{1.4.27}$$

We already know that we cannot solve the *S* and *U* functions analytically. Fortunately, it turns out that we can solve their derivatives analytically.

Recall the equation for the general arc length function:

$$S(u) = \int_0^u s(u)du \tag{1.4.28}$$

where

$$S(u) = \int_0^u \sqrt{\frac{dC(u)}{du} \cdot \frac{dC(u)}{du}}du. \tag{1.4.29}$$

Now, let us differentiate the arc length function, $S(u)$. That is, we need to differentiate the integral as a function of its limits. By doing this we get the following expression:

$$\frac{dS(u)}{du} = \sqrt{\frac{dC(u)}{du} \cdot \frac{dC(u)}{du}}. \tag{1.4.30}$$

If we look at the inverse mapping, we get:

$$\frac{dS(u)}{du} = \frac{1}{\sqrt{\frac{dC(U(s))}{du} \cdot \frac{dC(U(s))}{du}}} \tag{1.4.31}$$

where we already know the value of $U(s)$. This was computed during the arc length reparamerization phase of the space spline. Thus, the gradient of the space spline is easily computed. All in all we have an analytical expression for the first derivative of *U*, with respect to *s*. Now, let us differentiate this derivative with respect to *s*.

$$\frac{d^2U(s)}{ds^2} = \frac{d}{ds}\left(\frac{1}{\sqrt{\frac{dC(U(s))}{du}\cdot\frac{dC(U(s))}{du}}}\right)$$

$$= -\frac{\frac{d}{ds}\left(\frac{dC(U(s))}{du}\cdot\frac{dC(U(s))}{du}\right)}{2\left(\frac{dC(U(s))}{du}\cdot\frac{dC(U(s))}{du}\right)^{3/2}}$$

$$= -\frac{\left(\frac{d^2C(U(s))}{du^2}\cdot\frac{dC(U(s))}{du} + \frac{dC(U(s))}{du}\cdot\frac{d^2C(U(s))}{du^2}\right)\frac{dU(s)}{ds}}{2\left(\frac{dC(U(s))}{du}\cdot\frac{dC(U(s))}{du}\right)^{3/2}} \tag{1.4.32}$$

Collecting terms and substituting, Equation 1.4.31, for yields $\frac{dU(s)}{ds}$

$$\frac{d^2U(s)}{ds^2} = -\frac{\frac{dC(U(s))}{du}\cdot\frac{d^2C(U(s))}{du^2}}{\left(\frac{dC(U(s))}{du}\cdot\frac{dC(U(s))}{du}\right)^2}. \tag{1.4.33}$$

Again we notice that this is an analytical expression, because we already know the value of $U(s)$.

Now let us turn our attention toward the computation of the derivatives of the arc length function with respect to t. From Equation 1.4.33, we already know that

$$V(v) \rightarrow (t(v), s(v)). \tag{1.4.34}$$

Because $V(v)$ is a cubic spline, so is each of its coordinate functions. This means that we can find analytic expressions for the coordinate functions:

$$\frac{dt(v)}{dv} \text{ and } \frac{ds(v)}{dv} \tag{1.4.35}$$

by straightforward computations. In fact, these would be second-order polynomials. We also know that:

$$\frac{ds(v(t))}{dt} = \frac{ds(v(t))}{dv}\frac{dv(t)}{dt}. \tag{1.4.36}$$

Looking at the last term, we see that this is in fact the inverse mapping of Equation 1.4.35, so we end up having:

$$\frac{ds(v(t))}{dt} = \frac{ds(v(t))}{dv} \frac{1}{\frac{dt(v(t))}{dv}} . \qquad (1.4.37)$$

The value of $v(t)$ is already known. It was found by the root search that we performed in the time reparamerization phase of the space spline. Now let us try to look at the second derivative with respect to t. From Equation 1.4.37 we get:

$$\begin{aligned}
\frac{d^2 s(v(t))}{dt^2} &= \frac{d^2 s(v(t))}{dv^2} \frac{dv(t)}{dt} \frac{1}{\frac{dt(v(t))}{dv}} + \frac{ds(v(t))}{dv} \frac{-1}{\left(\frac{dt(v(t))}{dv}\right)^2} \frac{d}{dt}\left(\frac{dt(v(t))}{dv}\right) \\
&= \frac{\frac{d^2 s(v(t))}{dv^2}}{\left(\frac{dt(v(t))}{dv}\right)^2} - \frac{\frac{ds(v(t))}{dv}}{\left(\frac{dt(v(t))}{dv}\right)^2} \frac{d^2 t(v(t))}{dv^2} \frac{dv(t)}{dt} \\
&= \frac{\frac{d^2 s(v(t))}{dv^2}}{\left(\frac{dt(v(t))}{dv}\right)^2} - \frac{\frac{ds(v(t))}{dv} \frac{d^2 t(v(t))}{dv^2}}{\left(\frac{dt(v(t))}{dv}\right)^3} . \qquad (1.4.38)
\end{aligned}$$

Again we see that we have an analytic expression.

Looking at all our equations for the derivatives, we see that we may have trouble if we ever have:

$$\frac{dC(U(s))}{du} = 0 \text{ or } \frac{dt(v(t))}{dv} = 0 \qquad (1.4.39)$$

These degenerate cases do not cause any difficulties in practice, because there are two ways to handle the degenerate cases: repairing and prevention.

Degenerate Cases

Repairing

If the first derivative of a spline vanishes, it will occur at a single parameter value [Erleben02a]. That is, in any small neighborhood around this parameter value, the first derivative would be nonzero. This means we can remove the discontinuity by interpolating the spline in a small neighbourhood around the parameter value that caused the discontinuity.

Alternatively, we can also use a Taylor expansion around a nearby point in time to extrapolate the values of the scripted motion function. This method will work well as long as we do not have a cusp. Fortunately, in most cases, cusps are easy to see with the naked eye and are therefore easily removed by remodeling the spline.

Prevention

The second way of handling the degenerate cases implies that the splines are constructed in such a way that their first derivative never becomes zero. These kinds of splines are called regular splines [Erleben02a].

This is a rather tedious approach. Luckily, we do have some pointers from the spline theory, which can help us to avoid most of these cases. For instance, we can make sure that we do not have any knot values with a multiplicity greater than 1, or any succeeding control points that coincide, or any sequence of three or more collinear control points. This would guarantee that the first derivative is always nonzero at any knot value. However, it does not eliminate problems in between the knot values where cusps can occur. These cusps can be removed by adjusting the control points or by changing the interval size of the two-knot values in question.

The drawback of this method is that it requires some assistance on behalf of an animator to iteratively manipulate the splines. An automated spline interpolation, which guarantees that the order derivative does not become zero, would be preferable.

Examples

In this section, we show eight examples of scripted motion trajectories. We use a simple and an advanced space spline as well as two velocity splines (a constant-velocity spline and an exaggerated ease-in-ease-out velocity spline).

We have combined each space spline with each velocity spline and plotted the scripted motion position (small dark balls), velocity (bright arrows), and acceleration (dark arrows) at equidistant time intervals, see Figure 1.4.3. In the plots, we have scaled the velocity and acceleration vectors to 1/12th of their true magnitude in order to make it easier to verify the correctness of the visualizations.

We have also computed the motion by numerical integration of two coupled differential equations. The positions computed this way are shown as larger brighter balls. If we compare the larger brighter balls to the small dark balls, we see the same shape, displaced a little from the trajectory of small balls, especially at those places where the space spline bends rapidly. This is a typical behaviour of stiff differential equations, which is indicated by the growth of the acceleration vectors' magnitudes and changing directions. If scripted bodies are used in a simulator and their trajectories correspond to stiff differential equations, then it is better to compute their motion using our scripted motion function.

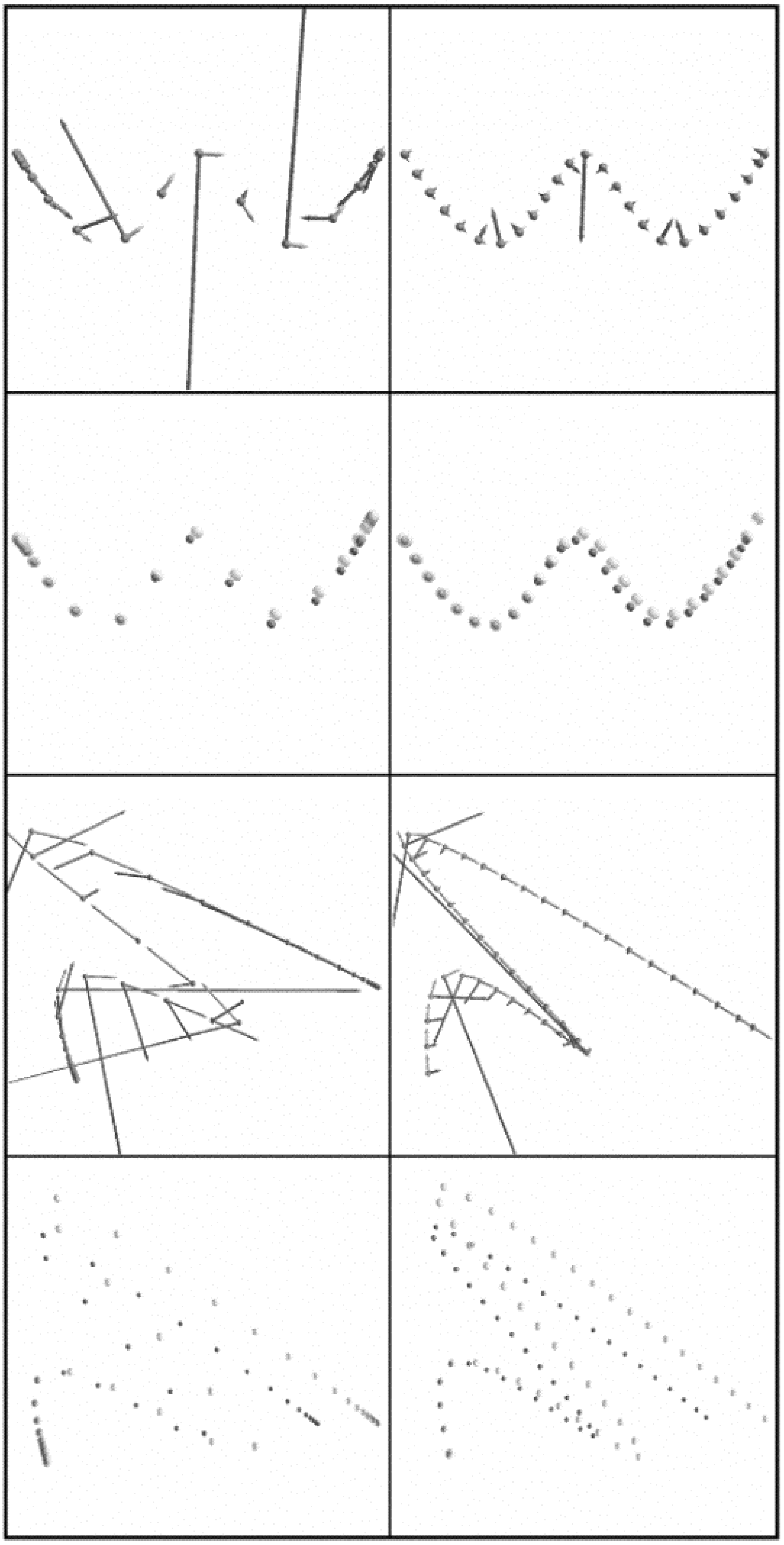

FIGURE 1.4.3 *Examples of dynamic trajectories.*

Future Work

The degenerate cases did not cause great problems for us in practice, but a spline interpolation method, which could guarantee that the first derivative never becomes zero, would be preferred and is left for future research.

The work we have presented in this article focuses on the linear motion part of the scripted motion problem. It appears to us that with very little effort, the work could be extended to handle the rotational motion part as well. For instance, by letting each of the Euler angles be described by a one-dimensional space spline, or setting up a "center of interest" space spline and/or a "view-up" space spline. The latter cases do, however, pose some minor difficulties in the computation of the derivatives (i.e., angular velocity and acceleration; see Appendix C in [Erleben02b]).

However, we feel that it would be much more interesting to look at an interpolation method that uses quaternions, like a squad spline [Watt92]. Unfortunately, the squad spline cannot be used. It only has C^1 continuity at its break points, and we require C^2 continuity in order to be able to work with angular acceleration. To our knowledge, a quaternion interpolation method that guarantees C^2 continuity everywhere does not exist.

Conclusion

In this article, we have presented a method for computing the first and second derivatives of traditional spline driven motion in the time domain of an animation/simulation. We have shown how the method is applicable to handle the linear motion part of the scripted motion problem and we have hinted at how the method could be used to handle the rotational motion part as well.

We have also suggested two directions for future work, spline interpolation with nonzero first order derivative and quaternion interpolation with C^2 continuity.

Table 1.4.1 Notations

$\vec{r}(t)$	The position of center of mass
$\vec{v}(t)$	The linear velocity of the center of mass
$\vec{a}(t)$	The linear acceleration of the center of mass
$q(t)$	The orientation of the body frame, represented by a quaternion
$R(t)$	The orientation of the body frame, represented by a rotation matrix
$\vec{\omega}(t)$	The angular velocity around the center of mass
$\vec{\alpha}(t)$	The angular acceleration around the center of mass

References

[Baraff01] Baraff, David, "Physical based modeling: Rigid body simulation," Online SIGGRAPH 2001 Course Notes, Pixar Animation Studios.

[Erleben02a] Erleben, Kenny and Knud Henriksen: *B-splines*, DIKU Technical Report 02/17, August 2002.

[Erleben02b] Erleben, Kenny and Knud Henriksen: *Scripted Bodies and Spline Driven Animation*, DIKU Technical Report 02/18, August 2002.

[Farin93] Farin, Gerald: *Curves and Surfaces for Computer Aided Geometric Design. A Practical Guide*, Third Edition, Academic Press, INC., Computer Science and Scientific Computing, 1993.

[Gravesen95] Gravesen, Jens: *The Length of Bezier Curves*, Graphics Gems V, pp. 199–205, 1995.

[Hoschek93] Hoschek, J. and D. Lasser: *Fundamentals of Computer Aided Geometric Design*, English translation 1993 by A K Peters, Ltd.

[Piegl95] Piegl, L. and W. Tiller: *The NURBS Book*, Springer-Verlag Berlin Heidelberg New York. 1995.

[Watt92] Watt, A. and M. Watt: *Advanced Animation and Rendering Techniques*, Theory and Practice, Addison-Wesley 1992.