Technical Report DIKU-TR-02/18
Department of Computer Science
University of Copenhagen
Universitetsparken 1
DK-2100 KBH Ø
DENMARK

August 2002

Scripted Bodies and Spline Driven Animation

Kenny Erleben and Knud Henriksen

Abstract: In this paper we will take a close look at the details and technicalities in applying spline driven animation to scripted bodies in the context of dynamic simulation. The main contributions presented in this paper are methods for computing velocities and accelerations in the time domain of the space splines.

Contents

1 Introduction

In this paper we present some ideas and theory for extending classical spline driven animation, such that it can be applied in a rigid body simulator. The need for this rather specialized behavior arises due to what is called scripted bodies in dynamic simulation.

In dynamic simulation a scripted body is used to avoid simulating the physical correct trajectories of the body. This is computational favorable and if one knows that the body is practically unaffected by the physical interactions of the other bodies in the configuration then the error is neglible. Consider for instance 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 exagerated way, but still all other bodies will interact in a physical meaningfull way with the scripted body.

Scripted bodies move very much like traditional objects known in animation. One specifies a trajectory by defining some keypositions at certain points in time, and 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 arises. First, there is the general problem of reparametrization of the splines into their natural parameters (this is a general problem in spline driven animation and not specific for dynamic simulation). Second, scripted bodies can interact with physical bodies in a simulator and therefore there is a need for knowing something about their motion and not just their positions and orientations. The interactions typically invole computation of "time of impact", sweeping volumes, collision impulses and contact- and constraint forces. What we need are several different time derivatives specifing the motion of the object. These are[1]:

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

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

$$Y(t) \rightarrow \{\vec{r}(t), \vec{v}(t), \vec{a}(t), q(t), \vec{\omega}(t), \vec{\alpha}(t)\} \tag{1}$$

In correspondence with rigid bodies this function can be seen as the state function of a scripted body. We define the scripted motion problem as finding a solution for the Y-function. In this paper we have concentrated on handling the linear motion only. However, with small changes the techniques can be applied to handle the rotational part as we discuss in section 7.

This paper is not about how to handle interactions with scripted bodies in a dynamic simulator nor is it about dynamic simulation as such. The reader is not required to know know anything about dynamic simulation, but it would probably give a better picture of why we have stated our problem as we have.

Much of the theory and definitions concerning splines and spline driven animation are presented in a rather compact form to save space. The reader should refer to our references for more details if they are needed.

We have organized the paper in such a way that we start by presenting our motivation for solving the scripted motion problem. Then we will outline the basic idea in spline driven animation and the problems with computing the derivatives of the motion. Hereafter we present solutions for all the technicalities in the scripted motion problem. Finally, we discuss degenerate cases and futrue work before we draw our conclusions.

2 Motivation

We have been working with dynamic simulation for a little while and 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

[1] see appendix A for explanation of notation

on the subject either simplified the problem by assuming that the state functions of the scripted bodies are known or other approaches using implicit functions and such were taken.

In either case we felt that the theory we have encountered was not very easy for an animator to use compared to systems such as 3DS MAX, Rhino, Maya® etc.. We think that dynamic simulation has a potentially huge application area in animation and it would be a shame not to pursuit this application area. So we wanted to extend/twist the traditionally animation techiniques, which are intuitive to use and well known by most animators, such that they can be used in a dynamic simulator.

Another benefit which 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.. Applying spline driven animation means a wide range of motion specification and applicability with the same type of scripted motion. When implementing a simulator it basically justifies looking on scripted motion as a black box making use of a not needed to know state function $Y(t)$ and as such it makes life a little more easy for those people that implement simulators. Let us summarize

- The work we present allows animators to use dynamic simulation with traditional well known animation techniques.
- The solution we present allows a dynamic simulator developer to look upon the motion of scripted bodies as though it were a (not needed to know) state function $Y(t)$.

3 The Basic Idea

Cubic splines are a good choice for specifing the trajectory of the linear motion because they are twice differentiable. It 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 prefered choice is cubic nonuniform B-splines. This class of splines is easily converted into a composition of cubic curve segments (such as cubic bezier curves see [6] for details) 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

$$C(u) \rightarrow (x, y, z) \tag{2}$$

However, the parameter u is not a natural parameter for humans to use. It is quite difficult for a human to predict exactly which point on a spline corresponds to a given value u. Humans are much better at thinking in terms of the arc length, s, instead of the spline parameter u. We would therefore like to use a reparameterization like

$$U(s) \rightarrow u \tag{3}$$

Such that we have

$$C(U(s)) \rightarrow (x, y, z)$$

This reparameterization makes sense since 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

$$u_1 < u_2$$

then

$$S(u_1) < S(u_2)$$

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 interrested 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 it. Instead he wants to specify a set of (t, s)-pairs, such that when the animation arives 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 (The term velocity is historical

and perhaps a little confusing in the context of dynamic simulation. A better word would proberly be "travelling distance")

$$V(v) \rightarrow (t, s) \tag{4}$$

Looking at equation (3) we see that equation (4) isn't really usuable. We are interested in twisting the equation such that we can find an arc length function. Which maps from the time domain to the arc length.

$$S(t) \rightarrow s \tag{5}$$

The velocity curve has to fulfill 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 totally 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 have been reparameterized in the following way.

$$C(U(S(t))) \rightarrow (x, y, z) \tag{6}$$

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

Until now things seem pretty easy, however we can not derive analytical functions for equations (3) and (5), at least not in the framework where C and V are both cubic splines. We have to look for a numerical solution of equation (6). It is not hard to imagine that things get even worse when we look for the first and second derivatives of equation (6). By applying the chain rule and product rule we get

$$\frac{dC(U(S(t)))}{dt} = \frac{dC}{du}\frac{dU}{ds}\frac{dS}{dt} \tag{7}$$

and

$$\frac{d^2C(U(S(t)))}{dt^2} = \frac{d^2C}{du^2}\left(\frac{dU}{ds}\right)^2\left(\frac{dS}{dt}\right)^2 + \frac{dC}{du}\frac{dU}{ds}\frac{d^2S}{dt^2} + \frac{dC}{du}\frac{d^2U}{ds^2}\left(\frac{dS}{dt}\right)^2 \tag{8}$$

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

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

are easily computed, but the remaining terms in equations (7) and (8) are somewhat more difficult to compute since we can not find analytical representations for the U- and S-functions.

4 The Linear Scripted Motion Function

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

4.1 The Arc Length Function

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

$$C(u) = \begin{bmatrix} x(u) \\ y(u) \\ z(u) \end{bmatrix} = \begin{bmatrix} a_xu^3 + b_xu^2 + c_xu + d_x \\ a_yu^3 + b_yu^2 + c_yu + d_y \\ a_zu^3 + b_zu^2 + c_zu + d_z \end{bmatrix}$$

Or in matrix notation

$$C(u) = \begin{bmatrix} u^3 \\ u^2 \\ u \\ 1 \end{bmatrix}^T M = U^TM$$

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

$$S(u) = \int_0^u (Au^4 + Bu^3 + Cu^4 + Du + E)^{1/2} du \tag{9}$$

Where (in 3-dimensional space)

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

Unfortunately, we can not 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.

$$s(u) = (Au^4 + Bu^3 + Cu^4 + Du + E)^{1/2} \tag{10}$$

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

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

With the theory we have developed so far, we can pick any numerical integration routine, which fulfills our requirements to performance and accuracy and then apply it in order to compute the arc length.

If one uses cubic bezier curves then there is another approach to evaluate the arc length, which is based on the convex hull property of bezier curves and the de Casteljau algorithm (for details see [4]). The algorithm is quite simple and easily explained. Let the control points of a bezier curve segment be P_0, P_1, P_2 and P_3. Now an estimate for the arc length is computed by

$$S_{est} = \sum_{i=0}^{2} |P_{i+1} - P_i|$$

If the bezier curve is sufficiently flat then this estimate would be very close to the real value of the arc length. In order to determine if the bezier curve is flat enough we measure the orthogonal distance of the two control points, which are not the end points, to the line between the two end control points. see figure 1 for an illustration.

$$\varepsilon = \max\left(dist(P_1, P_0P_3), dist(P_2, P_0P_3)\right)$$

If this value is greather than some threshold value, then the curve is not flat enough. To handle the problem one uses the de Casteljau algorithm to subdivide the bezier curve into two smaller bezier curves. By the convex hull property of the bezier curves we know that their convex hulls are smaller and tighter fitting, so we simply repeat the algorithm on each of these new bezier segments. Every time we find a bezier curve which is flat enough we halt the recursive subdivision and add the esitmated arc length to the total computed arc length so far.

```
Algorithm recursiveArcLength(B,eps)
  for i=0 to 2
    est += dist(B.P[i+1],B.P[i])
  next i

  L = line(B.P[0],B.P[3])

  err = max(dist(B.P[1],L),dist(B.P[2],L))

  if err > eps then
    (B1,B2) = subdivide(B)
    est = 0
    est += recursiveArcLength(B1,eps)
    est += recursiveArcLength(B2,eps)
  end if

  return est
End algorithm
```

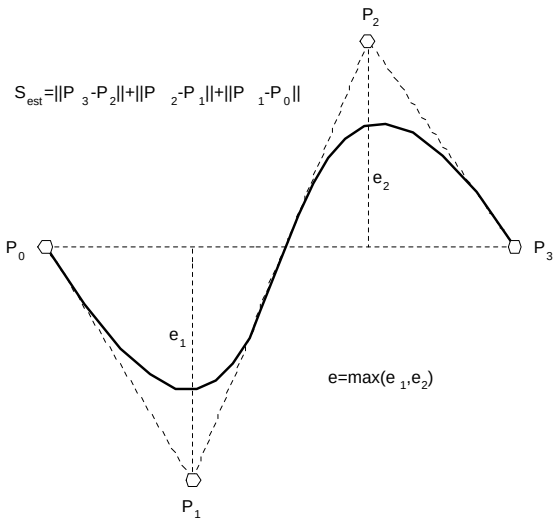


Figure 1: The arc length estimate of a bezier curve and the flatness measure.

The beauty of the algorithm is that it computes the arc length adaptively unlike some numerical integration routines (like the extended simpson), which have a fixed step size and unlike those integration routines which can integrate with adaptive step size, the algorithm is far simpler to implement and quite fast.

There is one subtlety we have overlooked. The algorithm given above computes the arc length of the entire bezier curve segment. We are interrested in finding the arc length of the segment of the bezier curve running from 0 to some value u. So we should do an initial subdivision to get the segment corresponding to the parameter interval we want to compute the arc length of.

```
Algorithm recursiveArcLengthAt(u,B,eps)
  (B1,B2) = subdivideAt(u,B)
  return recursiveArcLength(B1,eps)
End algorithm
```

In our case of cubic bezier curves the de Casteljau algorithm takes the form

$$P_{k,i}(u) = (1-u)P_{k-1,i}(u) + uP_{k-1,i+1}(u)$$

Where

$$\begin{aligned} k &= 1,\ldots,3 \\ i &= 0,\ldots,3-k \end{aligned}$$

And

$$P_{0,i}(u) = P_i$$

Actually de Casteljau can be used to compute a point on the cubic bezier curve because $B(u) = P_{3,0}(u)$. However we are going to use it for computing the new control points for the subdivided bezier curves. The new control points for the bezier curve representing the first half of the subdivision are

$$P_{k,0} \quad \text{for} \quad k = 0,\ldots,3$$

and the control points for the bezier curve representing the last half are

$$P_{3-i,i} \quad \text{for} \quad i = 0,\ldots,3$$

4.2 Arc Length Reparameterization

In this section we want to attack the problem of determing the reparameterization function U, such that

$$C(U(s)) \rightarrow (x(s), y(s), z(s))$$

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

$$U(s) = S(u)^{-1}$$

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

$$|s - S(u)| < \varepsilon$$

If we find such a u-value then we have actually found $U(s)$ (within the numerical tolerance ε). 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.

4.3 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 typical described by using a so called velocity spline. That is

$$V(v) \rightarrow (t(v), s(v))$$

From this we need to find a function

$$S(t) \rightarrow s$$

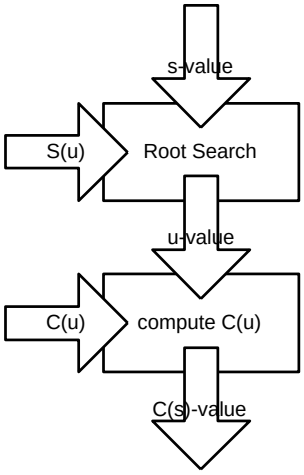


Figure 2: Schematic drawing of the method for arc length reparameterization.

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 search problem. Given a t-value we search for a v-value such that

$$|t - t(v)| < \varepsilon$$

When we find such a v-value then we have actually found the value of $v(t)$ and we can compute the s-value directly as $s(v)$. The time of the animation is a strictly increasing function as the animation evolves, so our root problem is guaranteed to have exactly one unique solution. This means our problem is solvable. Note, the velocity spline has to be physically meaningfull as we have explained earlier. This guarantees that we can always find an unique value for s (given a t-value).

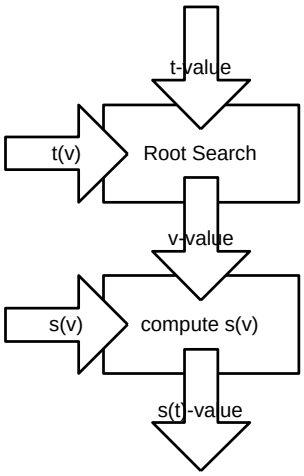


Figure 3: Schematic drawing of the method for time reparameterization.

4.4 Computing the Derivatives

From equations (7) and (8) we see that we need to compute the following values

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

We allready know how to handle the derivatives of C with repect to u. This leaves us with the remaining four derivatives

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

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

Let us start by taking a look at equation (19). With the help of equation (21) we can rewrite the integrand as the dot product of the derivative of the space spline, that is

$$S(u) = \int_0^u \sqrt{\frac{dC(u)}{du} \cdot \frac{dC(u)}{du}} du \tag{11}$$

This is a bit more convient to work with. Now, let us differentiate the arc length function. That is we need to differentiate the integral as a function of its limits. By doing this we get the following expression.

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

If we look at the inverse mapping then we will get

$$\frac{dU(s)}{ds} = \frac{1}{\sqrt{\frac{dC(U(s))}{du} \cdot \frac{dC(U(s))}{du}}} \tag{12}$$

Recall, that we allready know the value of $U(s)$. This was computed during the arc length reparamerization phase of the space spline and 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.

$$\begin{aligned}
\frac{d^2U(s)}{ds^2} &= \frac{d}{ds}\left(\frac{1}{\sqrt{\frac{dC(U(s))}{du} \cdot \frac{dC(U(s))}{du}}}\right) \\
&= -\frac{\frac{d}{ds}\left(\frac{dC(U(s))}{du} \cdot \frac{dC(U(s))}{du}\right)}{2\left(\frac{dC(U(s))}{du} \cdot \frac{dC(U(s))}{du}\right)^{3/2}} \\
&= -\frac{\left(\frac{d^2C(U(s))}{du^2} \cdot \frac{dC(U(s))}{du} + \frac{dC(U(s))}{du} \cdot \frac{d^2C(U(s))}{du^2}\right)\frac{dU(s)}{ds}}{2\left(\frac{dC(U(s))}{du} \cdot \frac{dC(U(s))}{du}\right)^{3/2}}
\end{aligned}$$

Cleaning up a bit we finally get

$$\frac{d^2U(s)}{ds^2} = -\frac{\frac{dC(U(s))}{du} \cdot \frac{d^2C(U(s))}{du^2}}{\left(\frac{dC(U(s))}{du} \cdot \frac{dC(U(s))}{du}\right)^2} \tag{13}$$

Again we notice that this is an analytical expression since we allready know the value of $U(s)$. Now let us turn our attention towards the computation of the derivatives of the arc length function with respect to t. From equation (4) we allready know that.

$$V(v) \rightarrow (t(v), s(v))$$

Since $V(t)$ is a cubic spline so is each of its coordinate functions. This means that we can find an analytic expression for

$$\frac{dt(v)}{dv} \tag{14}$$

By straightforward computations. In fact this would be a second order polynomial. Likewise we can compute

$$\frac{ds(v)}{dv} \tag{15}$$

We also know that

$$\frac{ds(v(t))}{dt} = \frac{ds(v(t))}{dv}\frac{dv(t)}{dt} \tag{16}$$

Looking at the last term we see that this is in fact the inverse mapping of equation (14), so we end up having

$$\frac{ds(v(t))}{dt} = \frac{ds(v(t))}{dv}\frac{1}{\frac{dt(v(t))}{dv}} \tag{17}$$

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 (17) we get

$$\begin{aligned}
\frac{d^2s(v(t))}{dt^2} &= \frac{d^2s(v(t))}{dv^2}\frac{dv(t)}{dt}\frac{1}{\frac{dt(v(t))}{dv}} + \frac{ds(v(t))}{dv}\frac{-1}{\left(\frac{dt(v(t))}{dv}\right)^2}\frac{d}{dt}\left(\frac{dt(v(t))}{dv}\right) \\
&= \frac{\frac{d^2s(v(t))}{dv^2}}{\left(\frac{dt(v(t))}{dv}\right)^2} - \frac{\frac{ds(v(t))}{dv}}{\left(\frac{dt(v(t))}{dv}\right)^2}\frac{d^2t(v(t))}{dv^2}\frac{dv(t)}{dt} \\
&= \frac{\frac{d^2s(v(t))}{dv^2}}{\left(\frac{dt(v(t))}{dv}\right)^2} - \frac{\frac{ds(v(t))}{dv}\frac{d^2t(v(t))}{dv^2}}{\left(\frac{dt(v(t))}{dv}\right)^3}
\end{aligned} \tag{18}$$

Again we see that we have an analytic expression. Looking at all our equations for the derivatives we see that we can get into trouble if we ever have

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

As it turnx out these degenerate cases do not cause any difficulties in practice and we have devoted the next section to talk about it.

5 Degenerate Cases

There are two ways to handle the degenerate cases, the first is by repairing and the second is by prevention.

5.1 Repairing

If the first derivative of a spline vanishes it will occur at a single parameter value (see [6]). That is, in any small neighborhood around this parameter value the first derivative would be non-zero. This sugests that we can remove the discontinuity by interpolating the velocities and accelerations of the scripted motion in a small neighborhood around the time value that caused the first derivative of the space spline (or the velocity spline for that matter) to become zero. Alternatively, one could also use a taylor expansion around a nearby point in time to extrapolate the values of the scripted motion function.

This method will work fine as long as we are not having a cusp. Fortunately in most cases cusps are easy to see with the naked eye and are therefore easily removed by remodelling the spline.

5.2 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 kind of splines are called regular splines (see [6] for details).

This is a rather tedius approach. Luckily we do have some pointers from the spline theory, which can help us to avoid most of the nasty cases. For instance, we could make sure that we do not have any knot values with a multiplicity greather than 1, or any succeding control points that coincide or any sequence of three or more control points lying on the same line. This would definitely guarantee that the first derivative is always non-zero at any knot value. However, it does not eliminate problems inbetween the knot values, here cusps could occur. These cusps could be removed by either adjusting the control points or 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.

6 Testing

We have implemented a small framework for testing our theory. In this section we will outline our strategy and present our first results.

6.1 Strategy

Four open nonuniform cubic B-splines splines were used in our test. Two three dimensional space splines and two two-dimensional velocity splines. We did construct a simple and an advanced space spline, the two velocity splines were a constant-velocity spline and an exagerated ease-in-ease-out velocity spline.

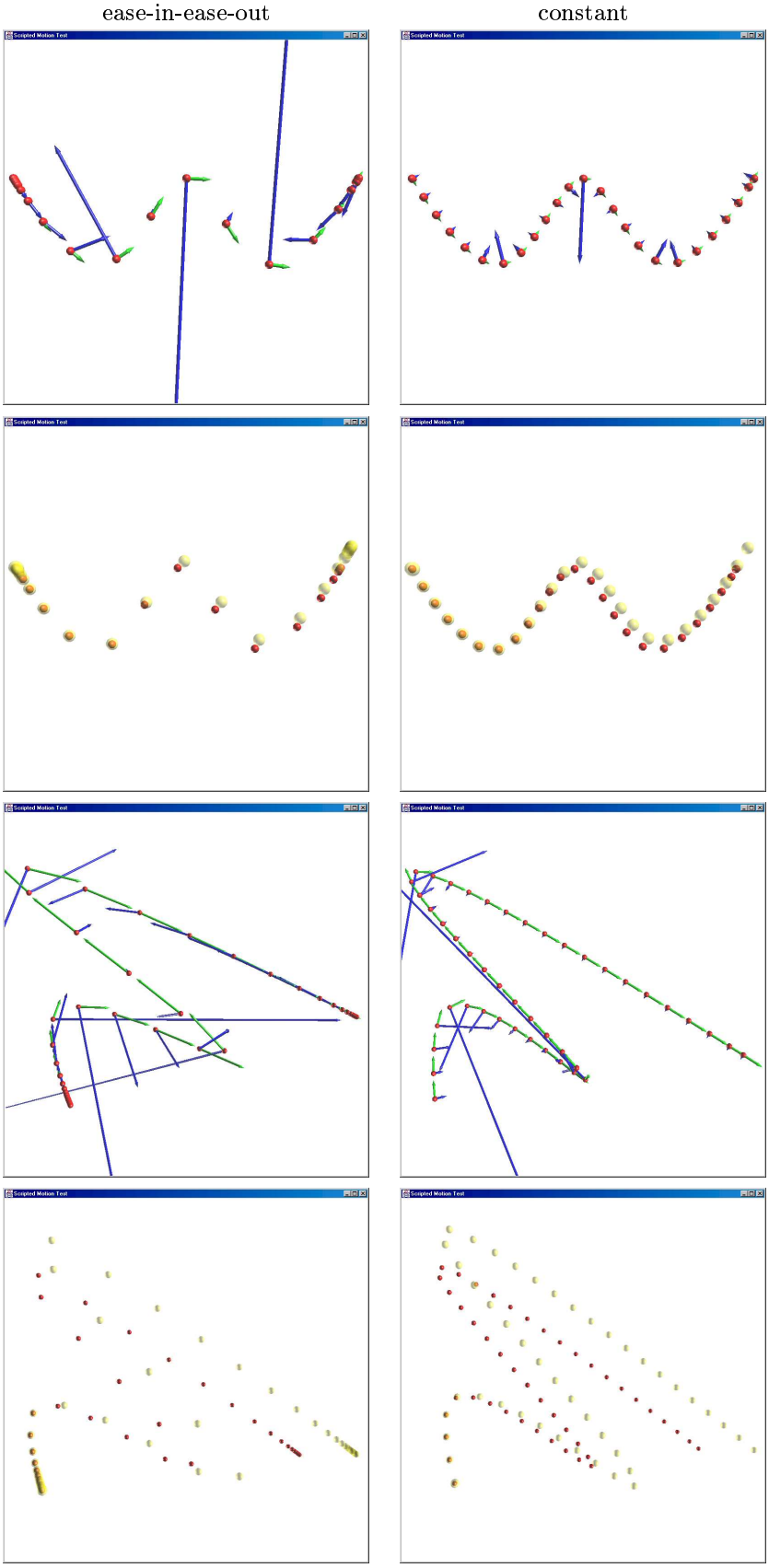
We have combined both space splines with both velocity splines and ploted the scripted motion position (red balls), velocity (green arrows) and acceleration (blue arrows) at equivdistant time intervals. In the plots we have scaled the velocity and acceleration vectors to 1/12'th of their true magnitude to make it more easy for the viewer to verify the correctness of our visualizations.

It can be hard to see if the accelerations really are correct, because the blue arrows do tend to grow very rapidly. So in order to prove that our accelerations really are physically correct, i.e. they "produce" the motion we want, we came up with the following test strategy. We used the scripted motion to construct a coupled first order differential equation like the one below

$$\frac{d}{dt}\begin{bmatrix} \vec{r}(t) \\ \vec{v}(t) \end{bmatrix} = \begin{bmatrix} \vec{v}(t) \\ \vec{a}(t) \end{bmatrix}$$

We then used a fourth order Runge Kutta ODE solver with adaptive step size to solve for the positions (yellow balls). The idea were to compare the positions computed by the ODE solver with the positions computed by the scripted motion function (red balls). If these positions are close to each other then in our opinion it would indicate that the accelerations are computed physically correct.

6.2 Results



6.3 Discussion

In our opinion these results show two problems. The first problem is best seen on those plots having constant velocity. Positions appear to be equidistant everywhere except at the endpoints of the space splines. Our investigations revealed that the problem is dependent on the choice of the numerical integrator, which is used to solve the arc length integrand. The problem also depends on the required numerical accuracy in the "root search" reparameterizations. We are therefore lead to believe that this problem is caused by numerical imprecision and numerical instability of the arc length integrand integration.

The second problem is much more obvious. If we compare the yellow balls to the red balls then it appears as though we keep the shape off the space spline, but we get somewhat of track. It is also seen that we get most out of track when the space spline bends rapidly. If we look at the corresponding acceleration vectors at the places where we get the most out off track, we see that at these places the vectors change rapidly both in direction and magnitude. This gives us the clue that the off-track problem is caused due to the fact that the accelerations are forming a stiff differential equation. This actually makes perfect sense since any physical real world force field which could cause the acceleration changes we see must be similar to a very stiff spring.

6.4 Conclusion

Our solution works but is influenced by numerical imprecision and instability (at least our implementation is). The first problem we encountered is unpleasent, but can be dealt with by extending the space spline beyond the ending position one really wants. This ensures that the velocity spline will not overshoot the maximum travelling distance of the space spline.

The second problem is actual an advantage to us. If scripted bodies are used in a simulator and their scripted motion actually corresponds to stiff differential equations then it is much more tracktable to have an analytical function computing the state of the scripted body instead of using an ODE solver.

7 Future Work

The degenerate cases did not cause great problems to us in practice, but it sure would be nice to have a spline interpolation method, which could guarantee that the first derivative never becomes zero. To our knowledge no such interpolation method exists today.

The work we have presented in this paper focuses on the linear motion part of the scritped motion problem. It appears to us that with very little effort the work could be extended to handle the rotational motion part as well, by for instance 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 later cases do however pose some minor difficulties in the computation of the derivatives (i.e. anuglar velocity and acceleration, see appendix C for hints on how to do it).

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

8 Conclusion

In this paper we have presented a method for computing the first and second derivatives of traditional spline driven motion in the time domain of the animation/simulation. We have clearly 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.

Besides that we have sugested two directions for future work, spline interpolation with nonzero first order derivative and quaternion interpolation with C^2 continuity.

References

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

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

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

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

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

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

A Notation

$\vec{r}(t)$	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

B The Arc Integrand Function

We will now derive the expression for the integrand function (see equation (10)). In the general n-dimensional case we have

$$S(u) = \int_0^u s(u)du \tag{19}$$

Where

$$s(u) = \sqrt{\sum_{i=0}^{n-1} \left(\frac{d}{du}C_i(u)\right)^2} \tag{20}$$

The subscript on C designates the i'th coordinate. This equation can be seen to be true by looking at the length, ds, of the arc length element between $C(u)$ and $C(u+du)$, That is.

$$ds = ||C(u+du) - C(u)||$$

Rewriting this we have

$$\begin{aligned} ds &= \left(dC_0(u)^2 + ... + dC_{n-1}(u)^2\right)^{1/2} \\ &= \left(\frac{dC_0(u)}{du}^2 + \cdots + \frac{dC_{n-1}(u)}{du}^2\right)^{1/2} du \end{aligned} \tag{21}$$

By comparision with equation (20) it is not hard to see that equation (19) is in fact true. Figure 4 shows how ds is defined in three dimensions.

Now let us try to resolve the equation from the inside out.

$$\frac{d}{du}C(u) = \left[3u^2, 2u, 1, 0\right] M$$

Looking at the i'th coordinate we get

$$\frac{d}{du}C_i(u) = 3M_{0,i}u^2 + 2M_{1,i}u + M_{2,i}$$

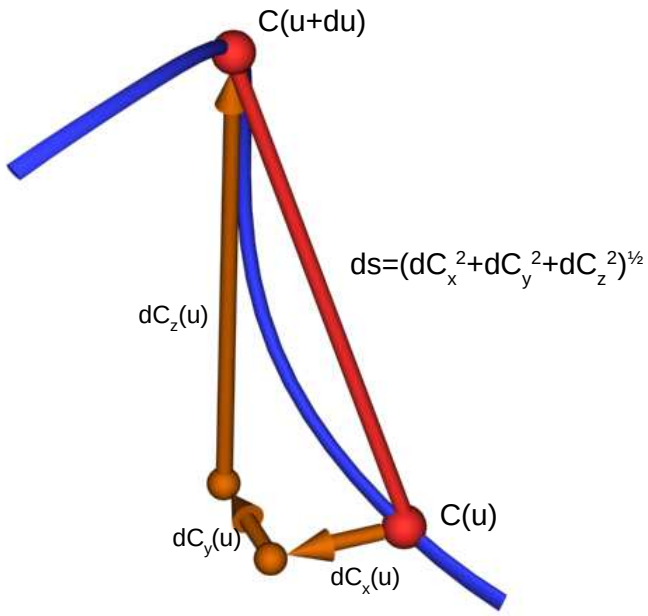


Figure 4: Illustration of the definition of ds.

Squaring the above equation gives us the following equation

$$\begin{aligned}\left(\frac{d}{du}C_i(u)\right)^2 &= 9\left(M_{0,i}\right)^2 u^4 + 12M_{0,i}M_{1,i}u^3 \\ &+ \left(4\left(M_{1,i}\right)^2 + 3M_{0,i}M_{2,i}\right)u^2 + 2M_{1,i}M_{2,i}u + \left(M_{2,i}\right)^2\end{aligned}$$

Writing this in matrix from gives us

$$\left(\frac{d}{du}C_i(u)\right)^2 = \left[u^4, u^3, u^2, u, 1\right]\begin{bmatrix} 9\left(M_{0,i}\right)^2 \\ 12M_{0,i}M_{1,i} \\ 4\left(M_{1,i}\right)^2 + 3M_{0.i}M_{2,i} \\ 2M_{1,i}M_{2,i} \\ \left(M_{2,i}\right)^2 \end{bmatrix}$$

Now let us take the summation into account

$$\sum_{i=0}^{n-1}\left(\frac{d}{du}C_i(u)\right)^2 = \left[u^4, u^3, u^2, u, 1\right]\sum_{i=0}^{n-1}\begin{bmatrix} 9\left(M_{0,i}\right)^2 \\ 12M_{0,i}M_{1,i} \\ 4\left(M_{1,i}\right)^2 + 3M_{0.i}M_{2,i} \\ 2M_{1,i}M_{2,i} \\ \left(M_{2,i}\right)^2 \end{bmatrix}$$

Let us now introduce the following notation

$$\begin{aligned} A &= 9\sum_{i=0}^{n-1}(M_{0,i})^2 \\ B &= 12\sum_{i=0}^{n-1}M_{0,i}M_{1,i} \\ C &= 4\sum_{i=0}^{n-1}(M_{1,i})^2 + 3\sum_{i=0}^{n-1}M_{0,i} + M_{2,i} \\ D &= 2\sum_{i=0}^{n-1}M_{1,i}M_{2,i} \\ E &= \sum_{i=0}^{n-1}(M_{2,i})^2 \end{aligned}$$

With these terms we can easily see that now we can write $s(u)$ in the following form:

$$s(u) = \sqrt{Au^4 + Bu^3 + Cu^2 + Du + E}$$

C The Rotational Motion

Let us assume that we can compute $q' \equiv \frac{dR}{dt}$ and $q'' \equiv \frac{d^2R}{dt^2}$ From physics we know that if a body rotates with angular velocity $\vec{\omega}$ then its change of orientation would be

$$\frac{dq}{dt} = \frac{1}{2}[0, \vec{\omega}]\, q$$

From this we derive

$$\frac{d^2q}{dt^2} = \frac{1}{2}[0, \vec{\alpha}]\, q + \frac{1}{2}[0, \vec{\omega}]\left(\frac{1}{2}[0, \vec{\omega}]\, q\right)$$

That is

$$\begin{aligned} q' &= \frac{1}{2}[0, \vec{\omega}]\, q \\ q'' &= \frac{1}{2}[0, \vec{\alpha}]\, q + \frac{1}{4}[0, \vec{\omega}]^2\, q \end{aligned}$$

Since we know q, q', and q'' we can easily isolate $\vec{\omega}$ and $\vec{\alpha}$ as follows

$$\begin{aligned} [0, \vec{\omega}] &= 2q'q^* \\ [0, \vec{\alpha}] &= 2\left(q'' - \frac{1}{4}[0, \vec{\omega}]\, q'\right) q^* \end{aligned}$$

In other words if we can compute upto the second derivative of the rotational "spline"/motion then we can compute the physical quantities we are looking for.