Interactive Control of Interpolations for Animation and Modeling

Gabriel Hanotaux, Bernard Peroche

Ecole Nationale Supérieure des Mines de Saint-Etienne
158, cours Fauriel, 42023 Saint-Etienne Cedex 2, France
(33) 77 42 01 69
e-mail: hanotaux@emse.fr

ABSTRACT

Most animation systems based on key-framing require the animator to define a multitude of curves to describe the evolution of parameters such as position, orientation, curvilinear abscissa, color as functions of time. On top of having to input a large number of data, the animator cannot easily estimate the relationships between all these curves. The aim of this paper, is to spare him the separate description of all these curves. In another paper, we presented a scheme making the interactive control of orientation interpolations possible. This scheme is now extended in order to allow the interactive control of both orientation and position curves. Furthermore, we present a new scheme allowing the automatic parametrization of such mixed curves. It provides a parametrization giving acceptable results, without the intervention of the animator. However, the animator has the possibility to modify this parametrization by specifying additional constraint. The parametrization is based on an optimization process. Such an interactive animation system has been used for the interactive design of sweep objects, i.e. objects constructed by moving a contour along a curve.

KEYWORDS: animation, interactive control, interpolation, orientation, quaternion, parametrization, sweep object.

1 INTRODUCTION

For each object of a scene, a key-framed animation system deals with a large set of parameters. For each of them, the animator must supply a curve describing its evolution over time. In this paper, our objective is to show that the orientation and position parameters can be treated on the same display. First, it prevents controlling an excessive number of windows. Besides, the relationships between positions and orientations are clearly controlled. Furthermore, position and orientation tangents provide full interactive control over the trajectories.

Section 2 explains the advantage of using Hermite interpolation in an interactive application. The techniques allowing full interactive control over orientation interpolation thanks to the notion of quaternion logarithms and spherical tangents are presented in section 3. Section 4 applies the developments of section 3 to the interactive control of both position and orientation trajectories. The first four sections only deal with the motion trajectories. In section 5 the motion dynamics are taken into consideration. We present a parametrization scheme based on the minimization of acceleration over the trajectory. These animation techniques are applied in section 6 for the interactive modeling of sweep objects.

2 HERMITE INTERPOLATION

One of the oldest techniques used in computer animation is the automatic generation of inbetweens (intermediate frames) on a set of key-frames supplied by the animator. Each key-frame specifies the position, the orientation and possibly the color, scale, etc ... of each animated body. For the moment, let us concentrate on the interpolation of positions. Each key-frame i is defined by a position P_i and the whole animation is constituted by a sequence of key-positions: $P_1, \ldots, P_N$.

Many techniques (Bézier curves, B-splines, etc...) have been developed for smooth interpolating between positions, some of which focus on the possibility to control the shape of the spline. The user manipulates abstract parameters like *bias, tension* and *continuity* in [11] or β_1, β_2 in [2].

In another approach presented in [6], the possibility of directly interacting on vectors tangent to the curve at each control points gives the user a more intuitive control. The curves are cardinale-splines. Additional control points are provided to help modify tangents vectors. These control points are invisible to the user and express such modifications by their locations.

On second thought, we settled among all cubic splines, for that based on Hermite interpolation. The reason for this is that, although it is a very simple method, Hermite interpolation has many advantages, especially for interaction purposes:

- it directly takes the notion of tangents into account (see below),
- since it is a cubic spline, control over the curve is local. Moving a control point only affects two pieces of curve,
- the curve goes through the control points,
- the computation of the curve is very fast.

The interpolation between P_i and P_{i+1}, requires a right tangent vector R_i and a left tangent vector L_{i+1} (see Fig. 1). In matrix form, the curve can be expressed as:

$$P_i(u) = (u)^T M_{Hermite} \begin{pmatrix} P_i \\ R_i \\ P_{i+1} \\ L_{i+1} \end{pmatrix}, 0 \le u \le 1 \quad (1)$$

where

$$M_{Hermite} = \begin{pmatrix} 2 & 1 & -2 & 1 \\ -3 & -2 & 3 & -1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \text{ and } (u) = \begin{pmatrix} u^3 \\ u^2 \\ u \\ 1 \end{pmatrix}$$

Left and right default tangent values are computed in the following manner:

$$R_i = -L_i = \frac{1}{2}(P_{i+1} - P_{i-1}) \tag{2}$$

These tangents generate what are commonly called Catmull-Rom splines (a subclass of cardinale-splines). The global

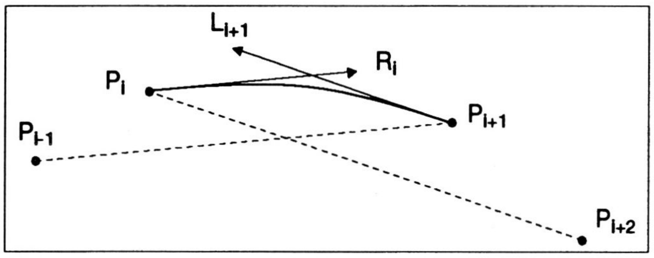

Figure 1: Hermite interpolation in 3-D space.

curve (interpolating between P_1 and P_N) is the concatenation of $N-1$ elementary curves $P_i(u)$ $(1 \le i \le N-1)$ which interpolate between P_i and P_{i+1}. Arbitrary choices for the right (left) tangent in the first (last) frame must be made.

The next two sections show how Hermite interpolation may be used for interactive control of position and orientation interpolations.

3 ORIENTATION INTERPOLATION

3.1 Previous Work

The introduction of quaternions in computer animation by Shoemake [15] provided a new and efficient way to describe orientations. Quaternions lie on the hypersphere of $\mathcal{R}^4$. They allow a concise, uniform and rotation-independent representation of orientations and have been widely used to interpolate orientations. We distinguish three different approaches:

- **Geometrical construction in quaternion space.** Spherical linear interpolation (*slerp* for short) between two quaternions q_1 and q_2 is well defined [15]. This spherical linear interpolation is the central part of techniques recently suggested by various authors to generate smooth curves on the hypersphere [15, 13, 7]. They use a geometric decomposition or subdivision scheme solely based on linear interpolation in the Cartesian space for different families of parametric cubic splines. Then, they construct curves confined to the surface of the hypersphere, substituting the spherical linear interpolation to the 3-D linear interpolation.
- **Parametrization of quaternion space.** This approach uses a parametrization of quaternion space by 3-D vectors. In [18], the parametrization is based on the exponentials of antisymmetric matrices and allows an interactive edition of orientation splines, thanks to 3-D spline manipulation. However, it seems hard to connect $\mathcal{R}^3$ and quaternion space.

  Our solution, built on a parametrization using quaternion logarithms, is presented below but belongs to this category.
- **Optimization process.** A recent approach presented in [1] applies the mathematical foundations of Cartesian space spline curves in quaternion space. It minimizes the tangential acceleration along the quaternion path. Unfortunately, the time required for numerical resolution of the optimization problem forbids any interactive application.

We now give an overview of the work presented in [9].

3.2 Log and Exp of Quaternions

As any complex number in $\mathcal{R}^2$, a quaternion is composed of a real part and a complex part. We shall denote it as $Q = [w, v]$, where w – the real part – is a scalar and v – the imaginary part – is a 3-D vector. The quaternion which represents a rotation about a unit axis v over angle θ is:

$$Q = \left[\cos\frac{\theta}{2}, \sin\frac{\theta}{2}v\right] \tag{3}$$

Quaternion exponentiation is defined in the standard way as:

$$\exp(Q) = 1 + \frac{Q}{1!} + \frac{Q^2}{2!} + \ldots + \frac{Q^n}{n!} + \ldots$$

An interesting case occurs when the real part of Q is zero [5]:

$$\exp(Q) = \exp([0, v]) = \left[\cos\|v\|, \frac{v}{\|v\|}\sin\|v\|\right] \tag{4}$$

and v is just the logarithm of $\exp(Q)$. By comparing Eq. 3 and Eq. 4, we see that it is possible to represent a rotation about a unit axis v over an angle θ by the unit quaternion $Q = \exp(\frac{\theta}{2}v)$ which makes a parametrization of this orientation by the 3-D vector $\log Q = \frac{\theta}{2}v$ possible.

3.3 Interpolation in Orientation Space

This parametrization is now applied to interpolation in quaternion space. We just have to interpolate quaternion logarithms and exponentiate, so as to obtain the interpolated quaternion.

As an illustration, spherical linear interpolation (denoted as *slerp*[1]) may be expressed in terms of quaternion logarithms as:

$$slerp(Q_1, Q_2, u) = \exp(lerp(\log Q_1, \log Q_2, u)) \tag{5}$$

It is important to notice that this *slerp* generally differs from the correct one. They are only equal when both axes of rotation are the same. It is a direct consequence of the non-commutativity of the quaternion product, i.e. $\exp(\log Q_1 + \log Q_2) \neq Q_1 Q_2$. However, we showed in [9] that the approximation is not far from the optimal.

Using this parametrization, it is possible to smoothly interpolate in orientation space. It suffices to replace *lerp* from Eq. 5 by another interpolation scheme such as Hermite interpolation. The spherical piece of curve which interpolates

[1]This notation was introduced by Shoemake in [15] in reference to *lerp.*, the linear interpolation in Cartesian space

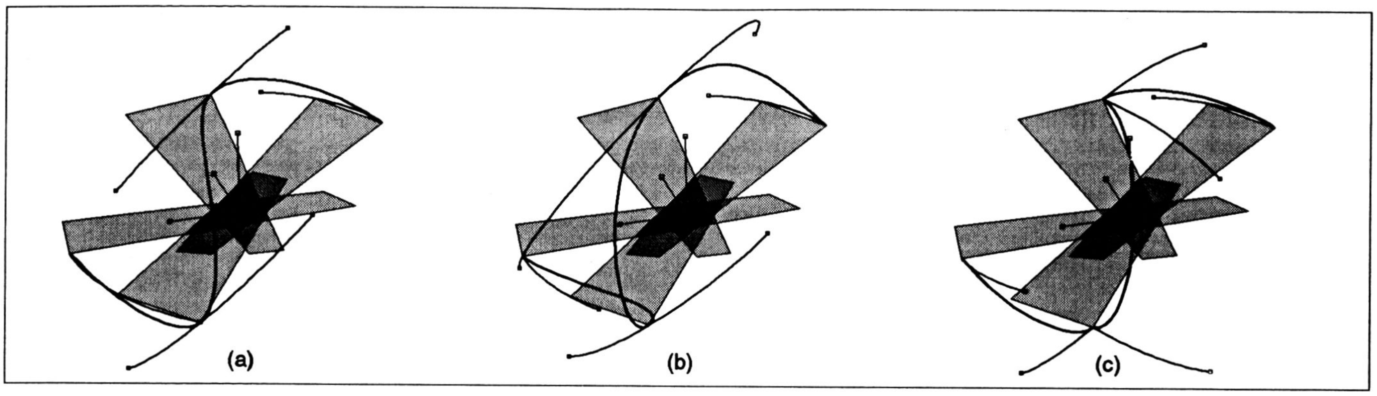

Figure 2: The curve represents the path followed by one vertex of the body (a single planar polygon). It interpolates four orientation-keys. (a) inital path using Catmull-Rom default tangents. (b) one tangent direction has been rotated, creating a loop, and one tangent amplitude has been increased. (c) some discontinuity effects generated by treating left and right tangents apart.

between two orientations Q_i and Q_{i+1} is then given by:

$$Q_i(u) = \exp\left((u)^T M_{Hermite} \begin{pmatrix} \log Q_i \\ \log Qr_i \\ \log Q_{i+1} \\ \log Ql_{i+1} \end{pmatrix} \right) \quad (6)$$

By analogy with the 3-D case, Qr_i and Ql_{i+1} represent what we shall call half spherical tangents. Qr_i represents half the rotation from Q_{i-1} to Q_{i+1}, and Ql_{i+1} half the rotation from Q_{i+2} to Q_i. Again (see Eq. 2), their default values are computed in terms of Catmull-Rom splines tangents. From group properties, we have:

$$Ql_{i+1} = (Q_{i+2}^{-1} Q_i)^{\frac{1}{2}}, Qr_i = (Q_{i-1}^{-1} Q_{i+1})^{\frac{1}{2}}$$

These spherical tangents can be interactively controlled in order to generate any desired effect (Fig. 2). An interesting analogy can be made with the figures from [1], obtained with another method. The difference is that our curves can be edited in real time.

4 MIXING POSITION AND ORIENTATION CONTROL

4.1 Generalized Key-Frames

Considering Eq. 1 and Eq. 6, it is possible to integrate position and orientation in the same interpolation process. It is enough to define key-frame i as the six-dimensional vector[2] containing both position P_i and logarithm of quaternion Q_i:

$$K_i = (\ P_i\ \log Q_i\ )$$

The interpolated frames $K_i(u)$ are then calculated as follow:

$$K_i(u) = (u)^T M_{Hermite} \begin{pmatrix} P_i & \log Q_i \\ R_i & \log Qr_i \\ P_{i+1} & \log Q_{i+1} \\ L_{i+1} & \log Ql_{i+1} \end{pmatrix} \quad (7)$$

4.2 Mixing Position and Orientation Curves

In most current computer animation systems, position and orientation interpolations are performed separately. The animator interactively edits two curves in separate windows, but has no way to see the relationships between positions and orientations. In the reality, positions and orientation are intrinsically linked. Exploiting this will undoubtedly make the trial-and-error process of finding the desired interpolation much less painful. Our aim is to enable the animator to interpolate orientations and positions in a parallel way – in other terms, on the same display.

[2]It is obvious that other quantities such as the one-dimensional vector S_i (the scale of the object) or the three-dimensional vector C_i (color of the object) could be added in K_i.

In what preceded, a solution allowing the interactive control of orientation interpolations was presented. The curves represent the path followed by one selected point of the animated body subjected to interpolated orientations. While we restricted the study to orientation (i.e. we assumed that all key-positions were located at the origin), the curve lay on a sphere.

To extend this approach for position inbetweening, we just have to allow the interactive manipulation of position of successive local coordinate systems. For this, a new curve – the position curve – is added, which interpolates the P_i's. As for orientations, Hermite interpolation is used. More details are given later on, regarding the interactive control of this curve.

In addition, the orientation curve must now depend upon the position curve. The former, attached to one point of the object, now represents the path followed by this point subjected to both position and orientation interpolations.

An example is shown on Fig. 3. Position and orientation curves are displayed. Each of them has its own set of tangents. The resulting animation is shown on the right side of the figure. The key-frames are shaded in dark gray, the inbetweens in light gray.

4.3 Interactive Control

All these curve lie in 3-D space and the input device is restricted to move in two dimensions. In order to perform interaction, we use an extension of the notion of 3-D cursor presented in [12]. The principle is to restrain the movement in two dimensions. For position, only translation along one axis (1-D translation) or in a plane perpendicular to one axis (2-D translation) are possible. For orientations, only

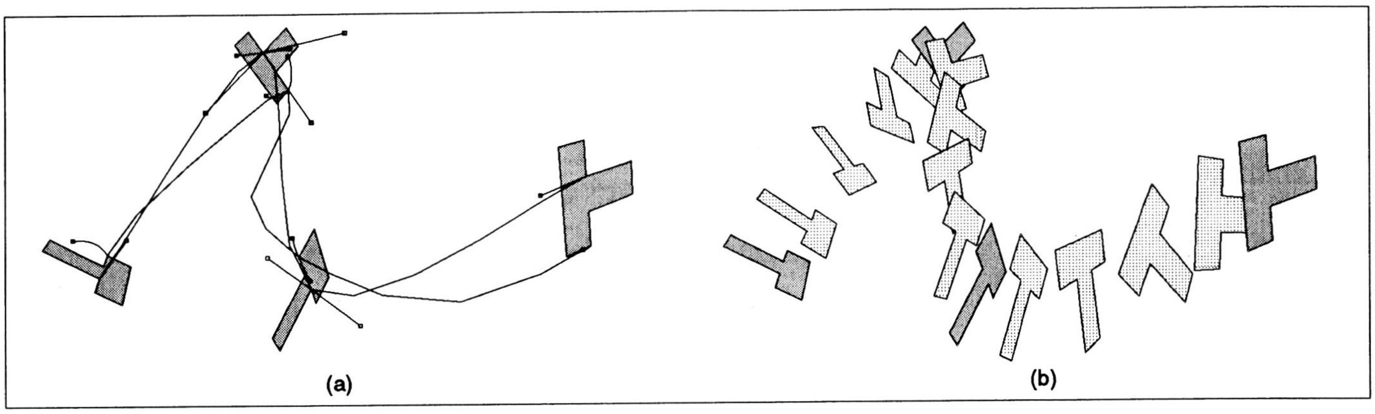

Figure 3: Position and orientation interpolation.

rotations about one axis are allowed (1-D rotation). Rotation about two axes (with two degrees of freedom) would be possible, but it doesn't seem intuitive to users.

In the two following subsections, we detail the interactive control of key-frames – which consists in modifying the P_i's or the Q_i's – and the interactive control of tangents to the curves. The former allows to modify the basic shape of the path while the latter allows fine tuning of the curves' appearance.

4.3.1 Key-Frames. We use the local coordinate system of each key-frame as reference axis. Positions are modified with 1D- or 2D-translations. Orientation-keys are modified with 1D-rotations. The user grasps one object and then translates or rotates it according to the cursor constraints. At each mouse displacement, the two curve pieces involving the key-frame are computed over again and refreshed.

4.3.2 Tangents. The interactive control of both flat[3] and spherical tangents is divided into two distinct operations. At any time, the user may either modify the amplitude or the direction of the tangents. This is not without interest since modifying direction and amplitude have different effects.

There is a similarity between the effects generated by modifying flat tangents and spherical tangents:

- **direction**. Changing the tangent direction results in the trajectory anticipating or overshooting a position-key or an orientation-key. The effect is equivalent to the *tension* effect described in [11]. Flat tangents are modified by rotating the tangent vector about the selected reference axis. Spherical tangents are modified by rotating an arc as described in [9].
- **amplitude**. Increasing the amplitude yields a more exaggerated curve. On the other hand, decreasing the amplitude generates a much sharper trajectory. Modifying the amplitude induces something similar to the *bias* effect. As for the position curve, the amplitude is modified by moving the tangent vector along a line, while in the case of orientations the amplitude is modified by moving the tangent arc along a circle.

[3]We call flat tangents, the tangents to the position curve by opposition to spherical tangents.

- **continuity**. Another possibility is to manipulate both left and right tangents at the same time or to manipulate only one half tangent at a time. The first case ensures C_1 continuity while the second allows to create discontinuities.

On the following example, the animation consists in four key-frames. We concentrate on showing the effect of spherical tangents, the case for position being well known. Fig. 4 shows three different animations based on the same key-frames, resulting from merely modifying spherical tangents. On Fig. 4.a, the motion uses default tangents. On Fig. 4.b, the direction of the spherical tangents has been interactively modified in the second and third key-frames. In the second key-frame (from left to right) a twist effect is created around the vertical bar of the T. On the third key-frame a twist effect is created around the horizontal bar of the T. On Fig. 4.c, the amplitudes of the spherical tangents have been increased to exaggerate the effects.

5 AUTOMATIC REPARAMETRIZATION

5.1 Introduction

In the previous sections, all the tools required for the fine description of position and orientation paths were presented. We did not worry about the motion dynamics, i.e. the velocity of bodies along trajectories. Indeed, tangents were designed for finely tuning the curve's appearance, not for specifying velocity at key-frames. First, the use of piecewise parametric polynomials may generate unpleasant effects. If key-frames are equally spaced in time, the number of samples is the same between two control points, be they very close or very distant from one another (see for example Fig. 5.a). Secondly, allowing the manipulation of tangents may result in the loss of velocity continuity.

A standard solution to solve these problems is based on a reparametrization of the trajectory. Principally, it is based on a reparametrization of parameter u in function of a more realistic and intuitive parameter, say t, the time. This reparametrization is made thanks to another curve expressing the relationships between the time and the parameter u (see [16]). In fact, this parametrization is more likely to be expressed as a function giving the velocity [3] or the curvilinear abscissa of the path in function of time.

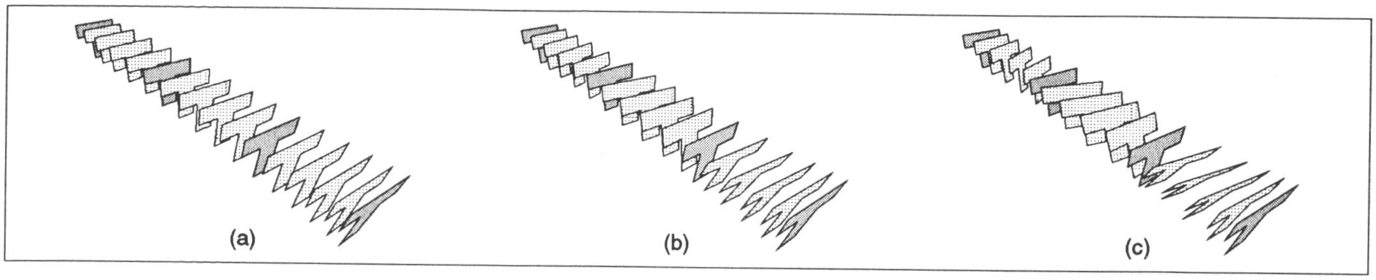

Figure 4: Influence of spherical tangents

However, such a reparametrization presents, from our point of view, the drawback to require the specification of still another curve, hence more and more interaction. Our aim is to develop a technique allowing the automatic reparametrization of interpolation curves. However, we also wish to let the user have the possibility to modify the proposed solution with the help of additional constraints. Let us first consider the method for position interpolation before turning to orientation interpolation.

The initial trajectory, as described previously, is made up of a sequence of cubic splines. Each of them is parametrized by u ranging from 0 to 1. We may consider that the global trajectory depends upon a more general parameter, say $\overline{u}$, ranging from 1 to N (recall that N represents the number of key-frames). The position on the curve is easily evaluated from a fixed value of $\overline{u}$. It suffices to determine the relevant piece of curve i and parameter u inside this curve. An animation sequence requires the computation of a fixed number of frames between time t_1 and time t_N. It motivates the discretization of the continuous $\overline{u}$ parameter in a set of equally spaced samples $\overline{u_i}$. Our objective is to find another set of samples, denoted t_i, such that the sampling of the $P(t_i)$'s (i.e. the motion dynamics) over the path appears more realistic than the initial sampling of the $P(\overline{u_i})$'s. The initial t_i values are initialized with the $\overline{u_i}$ values and iteratively improved according to a criterion described below. In the following, P_i denotes $P(t_i)$.

As we want to generate realistic motion, we choose to minimize the sum of the forces required to realize this motion. This principle is suggested by the fact that realistic motions tend to minimize the energy expanded to perform them [17]. According to Newton's law, it comes down in fact to minimizing the sum of the accelerations occurring during the motion. The scheme we have chosen is based on an optimization technique.

As usual, the discretization step is performed with the help of the finite differences method, frequently used in most animation techniques:

$$v_i = \frac{P_{i+1} - P_i}{h}, \quad a_i = \frac{P_{i-1} - 2P_i + P_{i+1}}{h^2} \tag{8}$$

In fact, we want to minimize the amplitude of the forces. So, we choose to minimize the sum of the squared accelerations. The criterion to be minimized is expressed as the finite sum:

$$C = \sum_{1 \leq i \leq N} a_i^2$$

Using Eq. 8 and discarding constant term h, we have:

$$\begin{aligned} a_i^2 &= (P_{i+1} - 2P_i + P_{i-1})^2 \\ &= 4P_i^2 + P_{i+1}^2 + P_{i-1}^2 + 2P_{i+1}P_{i-1} \\ &\quad -4P_i(P_{i+1} + P_{i-1}) \end{aligned} \tag{9}$$

5.2 The Optimization Process

The previous problem boils down to finding a vector t such that C(t) is minimized. We use an iterative process which consists in finding improved estimates $t^* = t + p$ such that $C(t^*) < C(t)$. In order to improve convergence rates, we use a second order Taylor expansion of C:

$$C(t+p) = C(p) + g^T(t)p + \frac{1}{2}p^T H(t)p$$

where g denotes the gradient of C and H the Hessian matrix of C. It can be shown (see [8]) that decreasing C implies solving:

$$g + Hp = 0 \tag{10}$$

We now give further details about the evaluation of the gradient and Hessian matrix of C and present a fast resolution algorithm for solving the linear system (10).

5.3 Computing First and Second Order Derivatives

The term in C where t_i occurs (denoted as C_i) is computed from a_i^2 (see Eq. 9), a_{i-1}^2 and a_{i+1}^2:

$$C(t_i) = C_i = 6P_i^2 - 8P_i(P_{i-1} + P_{i+1}) + 2P_i(P_{i-2} + P_{i+2})$$

The first derivative of C with respect to t_i, treating all other parameters as constants, is given by[4]:

$$g_i = \frac{\partial C_i}{\partial t_i} = 12P_iP_i' - 8P_i'(P_{i-1} + P_{i+1}) + 2P_i'(P_{i-2} + P_{i+2})$$

The Hessian of C containing the second derivatives of C is a matrix defined by:

$$\frac{\partial^2 C_i}{\partial t_i \partial t_j} = \begin{cases} 2P_i'P_{i-2}' & j = i-2 \\ -8P_i'P_{i-1}' & j = i-1 \\ 12P_i'^2 + P_iP_i'' - 8P_i''(P_{i-1} & \\ \quad +P_{i+1}) + 2P_i''(P_{i-2} + P_{i+2}) & j = i \\ -8P_i'P_{i+1}' & j = i+1 \\ 2P_i'P_{i+2}' & j = i+2 \\ 0 & \text{elsewhere} \end{cases} \tag{11}$$

[4] P_i' (resp. P_i'') denotes the first (resp. second) derivative of $P(t_i)$ with respect to parameter t_i. P_i' (resp. P_i'') is computed from Eq. 1, replacing $\begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix}$ by its first derivative $(\, 3u^2 \;\; 2u \;\; 1 \;\; 0 \,)$ (resp. second derivative $(\, 6u \;\; 2 \;\; 0 \;\; 0 \,)$).

Given the special structure of Eq. 11, the Hessian matrix is a non symmetrical penta-diagonal band matrix.

5.4 Solving the Linear System

In order to improve resolution time as well as storage requirements, we take advantage of the special structure of the Hessian matrix. This is particularly useful in our case where the number of time samples – and then the sparsity of this matrix – is high.

As for the organization of the Hessian matrix, only non-zero elements are of interest. They are stored in a $N \times 5$ array, each line containing significant elements of one line of the Hessian matrix.

The algorithm we use is based on a LU decomposition of the initial matrix. The system $Hp = -g$ is transformed into $LUp = g$ and then solved in two steps. A forward substitution first solves $Ly = g$, a backward substitution then solves $Up = y$, leading to the final solution. The LU decomposition and the forward substitution may be performed in the same step with an $O(N)$ algorithm, avoiding the storage of the lower triangular matrix. However, the upper matrix must to be stored in a $N \times 3$ array for later use in the backward substitution, which also takes $O(N)$ time.

The algorithm we implemented is very efficient and only requires minimum storage. Combined with the optimization algorithm whose convergence rate – thanks to second derivatives approximation – is very good, this solver algorithm allows to use the automatic parametrization scheme in an interactive environment. Also note that, due to the special structure of the Hessian matrix, no pivoting is required.

5.5 Results

On the following examples, the big squares represent the key-frames while the small ones represents the samples.

5.5.1 Equirepartition of Time Steps. Fig. 5.a shows an initial curve composed of the concatenation of three Hermite curve pieces. The curve is mainly designed for its small curvature. Compared with the other two, the central curve has a small curvilinear length. So, the global curve behaves as if the animated body's velocity suddenly slows down while crossing the second control point and then increases while crossing the third control point. Using our reparametrization, the samples are almost equally spaced, assuming constant speed along the specified path (Fig. 5.b).

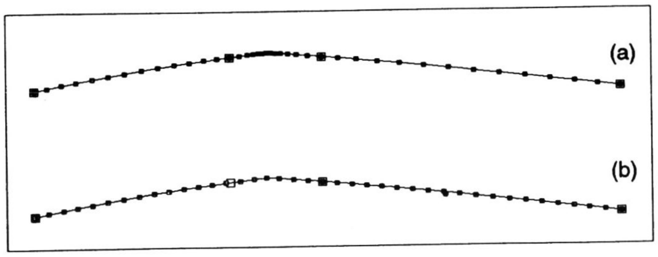

Figure 5: Trajectory with low curvature. (a) initial samples. (b) after reparametrization.

5.5.2 Curvature Influence on the Solution. Of course, the aim of reparametrization is not only to space interpolated points equally. This was indeed the case in 5.5.1 because the curve was very close to being a straight line. Just the same as a car slows down while reaching a bend and speeds up while leaving it, the velocity of our animated body decreases near a bended piece of curve and increases at the end of the curved stretch. Fig. 6.a shows the initial point samples. The same curve after the reparametrization producing a smooth motion is shown on Fig. 6.b. The higher the curvature, the higher the number of samples.

5.5.3 Additional Constraints. It is possible to impose a value for the velocity at various time samples. It is enough to substitute in Eq. 8 the specified values. Another choice is to prescribe the animated object to stand in a fixed position on the curve at a fixed instant of time. In this case, the optimization problem is divided into as many sub-problems as there are constraints. In both cases, the optimization problem remains unconstrained. On Fig 6.c, velocity constraints have been added to the end points of the path. The first constraint imposes a high initial speed while the second imposes a low final velocity (the motion goes from right to left). The resulting motion involves less sample points at the beginning and more at the end of the path.

5.6 Reparametrization of Orientation Curves

Of course, we also want to reparametrize orientation curves. The objectives are equivalent to those regarding positions, except we now have to minimize the sum of the torques applied to the animated body.

We use Newton's law applied to orientations

$$T = I\,\dot{\omega}$$

where

- T denotes the sum of torques applied to the body, no matter what these torques are due to.
- I represents the inertial matrix of the body. This matrix is computed in the local coordinate system of the body. Unfortunately, this coordinate axis does not necessarily coincide with the center of mass of the body, leading to a non diagonal inertial matrix.
- $\dot{\omega}$ represents the angular acceleration.

As previously, we want to minimize a finite sum of all the applied torques, i.e.:

$$C = \sum_{1 \le i \le N} (I\,\dot{\omega}_i)^2$$

where $\dot{\omega}_i$ denotes the angular acceleration at time t_i. This quantity is easily computed using finite differences and quaternion logarithms. First, the angular velocity may be represented by the angle θ_i traveled from orientation q_i to orientation q_{i+1} around a unit axis v_i in time h, i.e. by $\omega_i = \frac{\theta_i v_i}{h}$. $\frac{\theta_i v_i}{2}$ is the logarithm of the quaternion for rotation from q_i to q_{i+1}; we then have:

$$\omega_i = \frac{\log q_{i+1} - \log q_i}{2h}$$

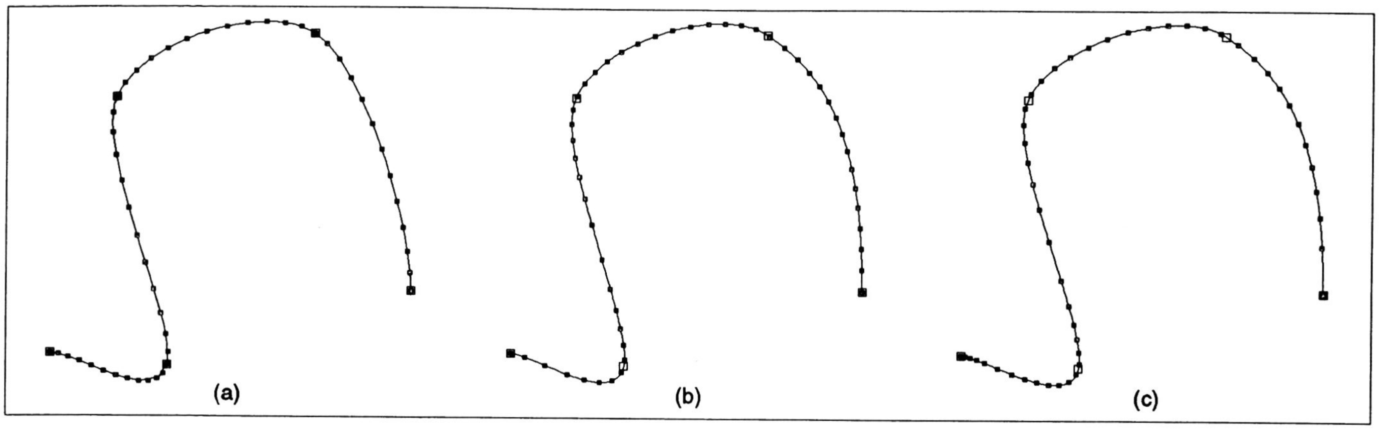

Figure 6: High curvature trajectory. (a) initial samples. (b) reparametrization. (c) other extremal constraints.

and the angular acceleration at time i is given by:

$$\dot{\omega}_i = \frac{\omega_i - \omega_{i-1}}{h} = \frac{\log q_{i+1} - 2\log q_i + \log q_{i-1}}{2h^2}$$

The gradient and Hessian matrix are computed as previously, even if the final expressions are more complicated, especially due to the fact that the inertial matrix is rarely diagonal. Positions $P(t_i)$ are replaced with the quaternion logarithms of interpolated orientation ($\log Q(t_i)$). The important fact is that the Hessian matrix remains a penta-diagonal matrix, ensuring good performance during the numerical resolution.

5.7 Reparametrization of Mixed Curves

In fact, we do not parametrize positions independently from orientations. Indeed, it would lead in most cases to two differents time parameter sets. In consequence, we choose to minimize both translational and angular accelerations. The new criterion is the sum of the previous criterions for position and orientation. Using both preceding subsections, computation of the gradient vector and of the Hessian matrix is straightforward. The Hessian matrix always remains a penta-diagonal matrix.

Also note that it is possible to favor the minimization of forces versus torques, by merely modifying the mass or inertial matrix of the animated body.

6 INTERACTIVE DESIGN OF SWEEP OBJECTS

This part covers the application to modeling of mixed position and orientation interpolations, which have initially been imagined for key-framed animation. Few authors [13] apply such animation techniques for the creation of sweep objects, also called generalized cones (GC for short) [10].

An extruded object is defined by a contour, a trajectory, and an orientation curve. The extruded object is obtained by moving the contour along the trajectory. The orientation curve specifies how the contour is rotated about a vector tangent to the trajectory (only one degree of freedom). If need be, other curves, giving the scale, the shape, etc.. of the contour in function of time may be added.

It is easy to apply the work presented above for the interactive design of GC's. We merely replace the animated three-dimensional object by a two-dimensional contour. Besides, a menu option enables to initialize the orientation-keys using the Frenet frame (see [10] for details). Then, the user may take advantage of all our interactive techniques to finely adjust the shape of the sweep object. In our application, we give the possibility to rotate the object about any axis (Fig. 7).

Our aim was not to extend theoretical work about GC, but rather to exhibit a method for providing fine interactive control on such objects. It may happen, when the diameter of the contour exceeds the curvature of the trajectory or when the trajectory describes loops, that parts of the GC self-intersect. In order to avoid this problem, the initial GC is split into smaller ones. Each of them corresponds to the object resulting from moving the contour along the trajectory from one time sample to the next. Then, we construct a CSG object made up of the union of all the small cones. The boundary evaluation [4] of the CSG object leads to a coherent object, the final GC (Fig. 8).

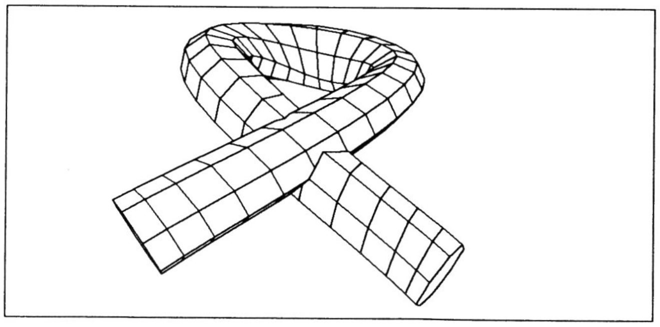

Figure 8: Self-intersecting object.

7 CONCLUSION

We have shown a new technique, based on an approximated parametrization of quaternion space, allowing the creation and visualization of spherical splines. The parametrization used seems easier to understand and to implement than those previously presented.

Hermite interpolation provides an efficient way to implement spherical tangents. Moreover, we provide full interactive control over this spherical splines. The use of spherical tan-

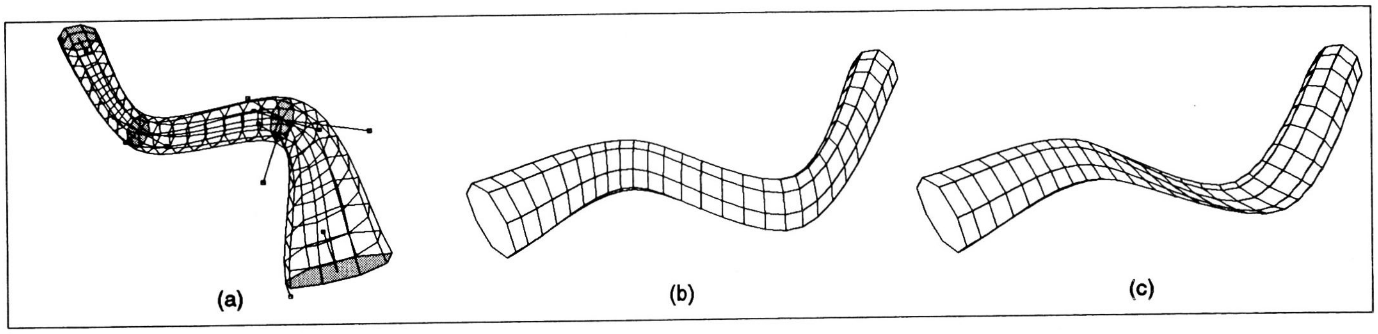

Figure 7: (a) Initial object using a Frenet frame. (b) Rotation about a vector tangent to the trajectory (the position of the eye has been changed). (c) Rotation about an arbitrary axis.

gents gives an intuitive means to control orientations. The effects of any manipulation may be visualized in real time.

The possibility to conjointly edit both position and orientation interpolations makes the job of the animator easier. One does not have to worry about two (or more) different curves, the relationships between positions and orientations being self-explanatory.

Furthermore, we gave a solution allowing automatic reparametrization of the curve, sparing the animator the trouble to worry about with still another curve. The parametrization is performed at request. The results are obtained immediately. Interactive specification of constraints is not still enabled. Work is in progress for this to be performed. Tangents are used to manipulate the curve and can not be used for specifying the motion dynamics. Our aim is to allow the user to directly manipulate the repartition of samples along the curves.

All these techniques have been conceived from the point of view of animation, but they have also been applied to the interactive design of generalized cones.

REFERENCES

1. BARR, A., CURRIN, B., GABRIEL, S., AND HUGHES, J. Smooth interpolation of orientations with angular velocity constraints using quaternions. *Computer Graphics 26*, 2 (1992), 313–320. Proc. SIGGRAPH'92.

2. BARTELS, R., BEATTY, J., AND BARSKY, B. *Splines for Use in Computer Graphics & Geometric Modeling*. Morgan Kaufmann Publishers, Inc., Los Altos, California, 1987.

3. BARTELS, R., AND HARDTKE, I. Speed adjustment for key-frame interpolation. In *Proceedings of Graphics 'Interface '89* (June 1989), pp. 14–19.

4. BENOUAMER, M., MICHELUCCI, D., AND PEROCHE, B. Boundary evaluation using lazy rationnal arithmetic : A detailed implementation. *2nd ACM, IEEE symposium on Solid Modeling & Applications '93* (May 1993).

5. BORGNE, M. L. Quaternion et contrôle sur l'espace des rotations. Tech. Rep. 751, INRIA, Nov. 1987.

6. CROS, F., AND BROCK, P. A method for providing full interactive control of the shape of 3d curves & surfaces. In *Eurographics'88* (1988), pp. 443–455.

7. DUFF, T. Splines in animation and modelling. In *State of the Art in Image Synthesis* (1986). SIGGRAPH'86 Courses Notes, number 5.

8. GILL, P., MURRAY, W., AND WRIGHT, M. *Practical Optimization*. Academic Press, Inc., London, 1982.

9. HANOTAUX, G. Interactive control of orientation interpolations. In *Third Eurographics Workshop on Animation and Simulation* (Cambridge, 1992).

10. KLOK, F. Two moving coordinate frames for sweeping along 3d trajectories. *Computer Geometric Aided Design 3* (1986), 217–229.

11. KOCHANEK, D., AND BARTELS, R. H. Interpolating splines with local tension, continuity and bias control. *Computer Graphics 18*, 3 (July 1984), 33–41. Proc. SIGGRAPH'84.

12. OVERVELD, C. V. Application of a perspective cursor as a 3d locator device. *Computer Aided Design 21*, 10 (Dec. 1989).

13. PLETINCKS, D. Quaternion calculus as a basic tool in computer graphics. *The Visual Computer 5* (1989), 2–13.

14. RAO, K., AND MEDIONI, G. Useful geometric properties of the generalized cone. *Proc. Computer Vision, Graphics and Image Processing 27* (1988), 129–156.

15. SHOEMAKE, K. Animating rotation with quaternion curves. *Computer Graphics 19*, 3 (1985), 245–254. Proc. SIGGRAPH'85.

16. STEKETEE, S., AND BADLER, N. Parametric keyframe interpolation incorporating kinetic adjustement and phrasing control. *Computer Graphics 19*, 3 (1985), 255–262. Proc. SIGGRAPH'85.

17. WITKIN, A., AND KASS, M. Spacetime constraints. *Computer Graphics 22*, 4 (Aug. 1988), 159–168. Proc. SIGGRAPH'88.

18. YAHIA, H., AND GAGALOWICZ, A. Interactive animation of object orientations. In *PIXIM'89* (1989), pp. 265–275.