A New Set of Tools
to Describe and Tune Trajectories

Jean-Dominique Gascuel   Christophe Lyon[*]

iMAGIS / IMAG [†]

BP 53, F-38041 Grenoble cedex 09, France

email: Jean-Dominique.Gascuel@imag.fr

Abstract

This paper presents an integrated set of tools helping to describe movements of objects in a framework of key-framed animation, both in simple and complex scenes. We want to have a powerful, robust and fast tool for designing either quickly prototyped or finely tuned motions. To achieve that, we will bring together Hermite splines (with an optional simplified interface for tangents, extended from [KB84]) and logarithms of quaternions for orientations [Han93].

We define an interface featuring both by-hand and simplified tangents edition. This gives us a unified simple interface for both translations and orientations. The construction of complex movements is easier to perform by assembling simpler ones, through hierarchy, rendezvous or links with other types of animation (dynamic, articulated structures, ...) We particularly address implementation considerations, and the problems of fast computations, to get interactivity.

1 Introduction

In Computer Animation, the dynamics of motion are very important, and have a lot of influence on the realism of the animation, hence on the overall impression of the spectators. All the Art of an animator is to give the illusion of life with abstract data-bases and programs. Animators are used to (and really need to) perform fine tuning of foreground and important motions, doing several attempts up to get the exact result they want. This is only possible if there is a simple and intuitive relationship between the inputs and the outputs of the animation system.

Physically based simulation and behavioral animation produce good motions, but they are difficult to control and predict. Interpolation methods, on the contrary, provide evident relations between key positions and movement. Moreover, local control permits to adjust only a part of the whole motion. And finally, they are predictable for an animator: it is easy to guess what correction to apply to the key positions to get the desired result.

Interpolation methods are widely spread in present and past animation systems. However, we feel that present models and implementations do not plainly meet our goal: power, simplicity and intuitivity. There are often problems with the type of spline used (for instance overshoot), or with the representation of orientations (which may introduce singularities). In this paper, we propose a set of tools that combines together methods that are known to be intuitive, add important improvements to them, and pack them into a coherent interface. The result is a more natural concept, providing easy simple sketching or advanced fine tuning of the movements, whenever the animator needs it.

Overview

We present a set of tools providing animators with the power of designing nice animations, based on interpolated movements. To achieve this, we split the process into four stages, each corresponding to a different sec-

[*]Current address : INA, Direction de la Recherche, Groupe Tele-Virtualite. 4 Avenue de l'Europe, 94366 Bry sur Marne Cedex, France.

[†]iMAGIS is a Joint Project between CNRS (EP J0038), INRIA (projet iMAGIS), INPG, and UJF.

tion of this paper:

1. A new spline model, based on Hermite polynomials, but slightly more predictable than the model described in Kochanek and Bartels's paper [KB84].
2. How to use these curves to compute a uniform speed trajectory (i.e. where the spline parameter equals the curvilign absissa), and how to have a unified approach for translations and rotations.
3. Speed control, to have a fine control over the dynamics of the action.
4. Building more complex movements by assembling simple ones together with other types of animation (such as physically-based simulation).

Section 7 deals with implementation, and explains which acceleration techniques can be used to get fast computation, which is a must for the interactivity of the whole system.

2 Previous Work

Various types of splines have been used in computer animation [Duf86]. We will follow Kochanek and Bartels [KB84], and use splines based on Hermite's polynomials. The idea here is to have a highly intuitive interface to define a spline segment: namely the two end points, and the two end tangents (see figure 1).

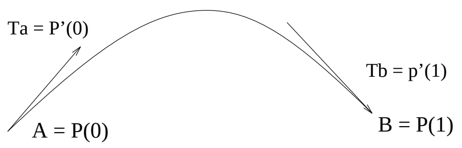


Figure 1: A Hermite spline segment, defined by its end points and tangents.

The interpolation between the two points A and B, using the tangents T_A and T_B, is given, in matrix notation, by:

$$P(u) = (u^3\ u^2\ u\ 1) \times \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} A \\ B \\ T_A \\ T_B \end{pmatrix} \quad (1)$$

2.1 Tension, Continuity and Bias

To simplify the definition of simple curves, we use the usual notions of tension, continuity and bias. Given a set of control points $P_0 \dots P_n$, the corresponding vectors DS_i and DD_{i+1} (standing for source and destination tangents) for each interval $[i, i+1]$ are defined by Kochanek and Bartels [KB84] as follows:

$$\begin{aligned} DS_i &= \tfrac{(1-t)(1-c)(1+b)}{2} (P_i - P_{i-1}) \\ &+ \tfrac{(1-t)(1+c)(1-b)}{2} (P_{i+1} - P_i). \\ DD_{i+1} &= \tfrac{(1-t)(1+c)(1+b)}{2} (P_{i+1} - P_i) \\ &+ \tfrac{(1-t)(1-c)(1-b)}{2} (P_{i+2} - P_{i+1}). \end{aligned}$$

Note that P_{-1} may be defined to be $2P_0 - P_1$, and P_{n+1} to be $2P_n - P_{n-1}$. Using this formula, the tension t, the continuity c and the bias b offer intuitive control of the shape of the curve. Furthermore, a default value of zero to the three parameters leads to the popular Cardinal spline.

2.2 Parameterization of Orientations

As explained in [Sho85], orientations are more problematic, since the space of rotations is isomorphic to S^4, the unit sphere of $\mathbb{R}^4$. We cannot speak of intuition or natural concepts in this four dimensional, periodic space ! Moreover, Euler angles or matrix representation do not provide a suitable computing basis, because of non unicity or singularity problems.

The only representation of rotation suitable for interpolation is the use of quaternions, which have important properties, such as (quasi) unicity of representation, and no singularity.

Quaternions were first used by [Sho85, Duf86, Sho87] through various geometric constructions, always based on multiple *spherical linear interpolations (slerp)*. More recently, [YG89, HP93] defined the quaternion logarithms, and proposed to use them to speed up calculations.

Here, we use the approach of Hanotaux: Hermite splines applied to quaternions, and fast computations through their logarithm. The interpolation between two orientations, specified by their quaternions Q_a and Q_b and the tangents QT_a and QT_b, is approximated, in matrix notation, by:

$$Q(u) = \exp\left[(u^3\ u^2\ u\ 1)\right.$$

$$
\times \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}
$$
$$
\left. \times \begin{pmatrix} \log Q_a \\ \log Q_b \\ \log QT_a \\ \log QT_b \end{pmatrix} \right] \tag{2}
$$

This equation is exactly equivalent to equation 1 used for translations, hence improvements to translation methods will apply equally to the rotations.

3 Improving Existing Models

Unfortunately, computing end tangents with the Kochanek & Bartels [KB84] formula does not always produce predictable results. For instance, figure 2 shows the case of control points lead down left to right, while the resulting curve displays a right to left part.

(a) (b)

Figure 2: Both figures show a curve obtained with control points evenly spaced along the y axis. In (a), the control points are also evenly spaced along the x axis. In (b), they are not (the third, topmost point is outside the figure). In this situation where the control points are strictly "left to right", the curve may go "backwards".

This is a consequence of unevenly spaced control points. Nevertheless, the "features" may appear in unwanted places, and should be addressed to improve the predictability of the interpolation tool.

To alleviate this problem, we add a correction that normalizes the distance between control points. The new equations are:

$$
\begin{aligned}
DS_i &= \frac{2l_i}{l_{i-1}+l_i}\left[\frac{(1-t)(1-c)(1+b)}{2}(P_i - P_{i-1}) \right. \\
&\quad \left. + \frac{(1-t)(1+c)(1-b)}{2}(P_{i+1} - P_i)\right] \qquad (3) \\
DD_{i+1} &= \frac{2l_i}{l_i+l_{i+1}}\left[\frac{(1-t)(1+c)(1+b)}{2}(P_{i+1} - P_i) \right. \\
&\quad \left. + \frac{(1-t)(1-c)(1-b)}{2}(P_{i+2} - P_{i+1})\right]
\end{aligned}
$$

Where:

$$
\begin{aligned}
l_{i-1} &= ||P_i - P_{i-1}|| \\
l_i &= ||P_{i+1} - P_i|| \\
l_{i+1} &= ||P_{i+2} - P_{i+1}||
\end{aligned}
$$

The corresponding curves are drawn in figure 3.

Figure 3: After the correction we propose, the curve always follows the control points, and there is no longer a loop when the control points are unevenly distributed.

We should make two important remarks about this enhancement. First, one can compute the length of the spline segment between P_i and P_{i+1}, and show that it becomes a linear function of l_i, that no longer depends on the lengths l_{i-1} or l_{i+1}.

On the other hand, the joint between spline segments will no longer be $\mathcal{C}^1$, but only $\mathcal{G}^1$ (the tangents have the same direction, but not the same length). This is not important, since the Hermite spline does not have a uniform speed anyway (the parameter speed du/dt is distinct from the object space speed $dP(u)/dt$). Hence to have a predictable speed along the curve, we should use a speed uniformization step (see section 5.1). This reparameterization of the curve will not, of course, affect the curve shape.

Direct 3D Manipulation

Our goal, depicted in the introduction, is to design tool offering the maximum level of control when needed. The t, c, and b parameters are efficient, but limitative. For instance, if the four points $P_{i-1} \dots P_{i+2}$ lie in the same plane, the tangents DS_i and DD_{i+1} will also lie in the plane, and the curve will be planar between P_i and P_{i+1}. We propose to let the user choose between two modes:

TCB mode: Tangents are computed automatically from control points. The default value (zero) produces a cardinal spline. Every time a control point is moved, the tangents are recomputed accordingly.

Full mode: The user defines the half tangents with a 3D rotation widget, these tangents will not be recomputed from *tcb* when the associated control points are moved.

Switching from the former mode to the latter can be done just by selecting and dragging one of the tangents. Changing from the latter to the former mode should be done carefully, because extra information put in by the user should not be silently discarded.

4 Movement Interpolation

In this section, we explain how the spline model described above is used for animation to control the motion of a single object over time.

For a solid object, motion can be split into a translation and a rotation around the object barycenter. The translation is obtained from the movement of a single point, that is to say an interpolation in usual 3D space. So the spline curve of section 3 can directly be used. Let us explain how to control orientations.

A Better Interface for Orientations

In order to compute the motion, the key orientations have to be defined. We propose to use "ghosts" (or clones) of the object, placed in the scene by the animator, through the use of standard 3D translation and rotation widgets.

Providing a good interface for defining the tangents for orientations is more problematic. Hanotaux [Han92] represents the quaternion space as a point moving on a unit sphere of $\mathbb{R}^3$ (omitting one of the degrees of freedom). In this way, tangents are simply geodesic curves drawn on the sphere. For simple cases we found this editing mode over killing. If S^4 is counter-intuitive, S^3 (the usual unit sphere) is not sufficient to view orientations, and it seems hard to really understand the effects of rotation speed discontinuities for instance.

We introduce a new unified approach for rotations and translations. Because logarithms of quaternions are rotation vectors, it is easy and fast to apply the modified Kochanek and Bartels equations 4 to deduce the tangents from higher level parameters.

Moreover, as the 3D translation space is more intuitive than the S^4 rotation space, it is a good idea to have the same *tcb* parameters for the two tangents computations, both in translation and in rotation. By this way, any special editing done to introduce a speed discontinuity for some bounce causes an abrupt change in the rotation direction. Again, this must be considered as a default, that do help to produce high quality results through successive experiments.

5 Speed Control

In the previous sections, we discussed only the shape of the trajectories. In an animation, objects move along their trajectories, with some speed. This curvilinear speed is very important to express the "mood" of the action. Having the final speed of an action staying constant or decreasing to zero makes the difference between a slamming door, and one that is gently closed.

In the remainder of this section, we propose a "three interpolants" technique to get a high level of control on the speed of the object over its trajectory.

5.1 Speed Uniformization of the Spline

Let us first note that Hermite splines are not uniform: the curvilinear speed depends on the actual points and tangents used. If l_P is the curvilinear parameter along a spline segment, a small step dl_P is defined by:

$$dl_P = \sqrt{dx^2 + dy^2 + dz^2} = ||P(u + du) - P(u)||$$

Where dx, dy, and dz are the variation of the coordinates of the interpolated point $P(u)$ between parameter values u and $u+du$. Similarly for quaternion space, distances could be defined by using the small angle dl_Q between the orientations $Q(u)$ and $(Q(u + du)$, by:

$$dl_Q = ||Q(u + du) \cdot \overline{Q(u)}\,||$$

We combine them for instance using the relative masses and inertia (if the full matrix inertia is to be used, it should be done during the dl_Q calculation. In our implementation, we use a scalar, approximating the inertia matrix by its trace):

$$dl = m\ dl_T^2 + j\ dl_Q^2$$

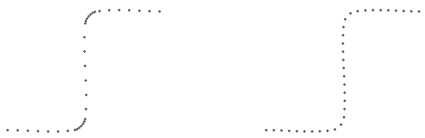
These formulas can be used to numerically compute the curvilinear abscissa, $l(u)$.

Let the animator set a specific time t_0 for the beginning of the trajectory, and t_n for its end. The uniform motion along the curve is obtained by having a linear relationship between the time t and the curvilinear abscissa l:

$$l(t) = l_{\max} \frac{t - t_0}{t_n - t_0}$$

Where $l_{\max}$ is the total curvilinear length of the trajectory.

This, in turn, gives the spline parameter u of some interval i, which is finally used to compute the interpolations $P(u)$ and $Q(u)$. Section 7.2 will explain how to get a fast reverse mapping from l to u.



(a) (b)

Figure 4: The effect of speed uniformization on five $\mathcal{G}^1$ connected Hermite spline segments: without uniformization in (a), with curvilinear speed uniformization in (b).

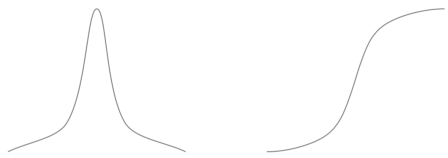
This calculus applies for any set of spline segments, and in particular for the $\mathcal{G}^1$ joints introduced by the correction of section 3. Such an example is drawn in figure 4.

5.2 Adding Speed Variations

More complicated relations between t and l are often useful, because it enables to give some mood or character to an action (remember the slamming door). To preserve soundness of the trajectory, we should keep $l(t_0) = 0$, $l(t_n) = l_{\max}$, and l an increasing function of t.

We propose to use a 2D graphic interface to define the speed variations over time. If we know the speeds v for some key times in $[0..1]$, we can then interpolate them to have the speed variation during the full interval (see figure 5).

Let $L(t)$ be the numeric integral of the function v. Then, the following formula renormalizes the speed



(a) (b)

Figure 5: Interface concepts to add speed variations. In (a), the speed curve given by the animator. In (b) its integral, rescaled to fit the $[0..l_{\max}]$ interval.

(a) (b)

Figure 6: The effect of speed variations on a set of Hermite spline segments: with uniform speed in (a), with animator given speed variations in (b).

variations, and introduces them in the previously defined speed uniformization scheme, to get a motion in which you have a good control of both space and time (see figure 6):

$$\begin{aligned} L(x) &= \int_0^x v(t)dt \\ l(t) &= \frac{l_{\max}}{L(1)} L\left(\frac{t - t_0}{t_n - t_0}\right) \end{aligned}$$

6 Complex Motions

We have explained so far how to define movements suitable for simple isolated objects. This is obviously too restrictive for any real scenario. On the other hand, a real scenario needs more than just complex interpolated motions. This is why we have implemented this new set of tools as part of *Fabule* [Gas94], the research environment for computer animation developed at iMAGIS. This environment provides us with a set of C++ classes dealing with primitives, physically based motions, articulated objects built by using geometric constraints [GG94], and among others implicit deformable objects responding to collisions. A *Tcl* interpretor enables to design an *OSF/Motif* user interface, and to write demonstrators.

This section briefly presents three ways of building up more complex motions by combining simpler ones, using several features of the animation system, *Fabule*.

6.1 Hierarchy of Motions

The first, classical, solution for describing an articulated object is to use a hierarchy of motions [Las87, WW92]. In this model, each joint of the figure is interpolated over time, and the actual position of some member is obtained by composing the displacements between it and the root member. This way of designing complex motions and animating articulated figures is widely spread today, even if making the end of some articulated chain follow a given path is very tedious and counter-intuitive.

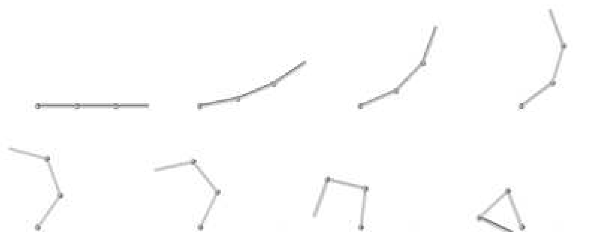


Figure 7: A hierarchy of motions. Each joint of the articulated arm just defines rotation in its parent referential.

In *Fabule*, this is simply done by using a special class that composes the basic movement generated by individual objects. This approach permits to link any object generating a movement on top of any other module that also generates a movement. This way, a hierarchy of key framed, physically based, or any other generators can be built.

6.2 Sticks Muppets

A second point of view to animate articulated figures consists in specifying only the trajectories of some components (for instance, hands and feet of a character). The other parts just follow the movement, through dynamic simulations, articulation resolution, inverse kinematics, or any suitable approach. This mode is best suited to automatically compute the secondary motions of an action, such as the swing of the beards of the Seven Dwarfs in the famous Disney animation.

Fabule proposes the notion of connections for that purpose. These features connect two (or more) movement generators, and compute interactions among them. Several types of connections exist, enabling to simulate the effects of geometric constraints [GG94] (to build articulated structures), springs, goal directed effectors, ... See figure 8 for such an example.

6.3 Motions with Rendez-vous

We propose a third approach, plainly original, consisting in attaching some of the key positions to moving referentials. If you want to animate a hand catching a moving glass, one way to do this is to consider the grasping key position of the hand to be fixed in the glass space. This ensures that whatever the motion of the glass is, the hand will correctly reach its target.

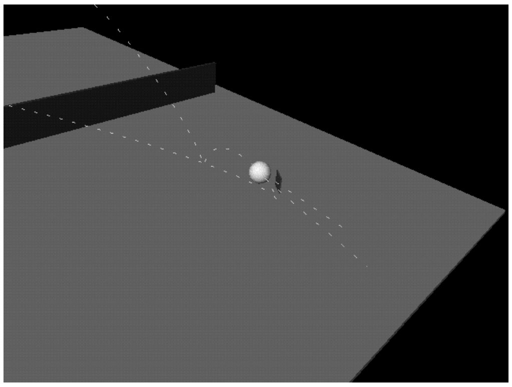


Figure 9: Rendez-vous between a tennis ball and a racket.

In this rendez-vous, one (or more) key position is moving, under the action of some generator. This means that local parameters (such as tangents) and global ones (such as $l_{\max}$) have to be re-computed at each time step. Fortunately this is not always the case, and even when it is, not all computations have to be redone every time, as explained in the next section.

7 Optimized Implementation

In this section, we propose some insights on how we have actually implemented the computations for our interpolation scheme, in order to achieve the necessary interactivity.

The first thing to remark is that we are often only interested by translation or orientation interpolation, but not both (for instance, think about the joints of the articulated figure 7). The first optimization is

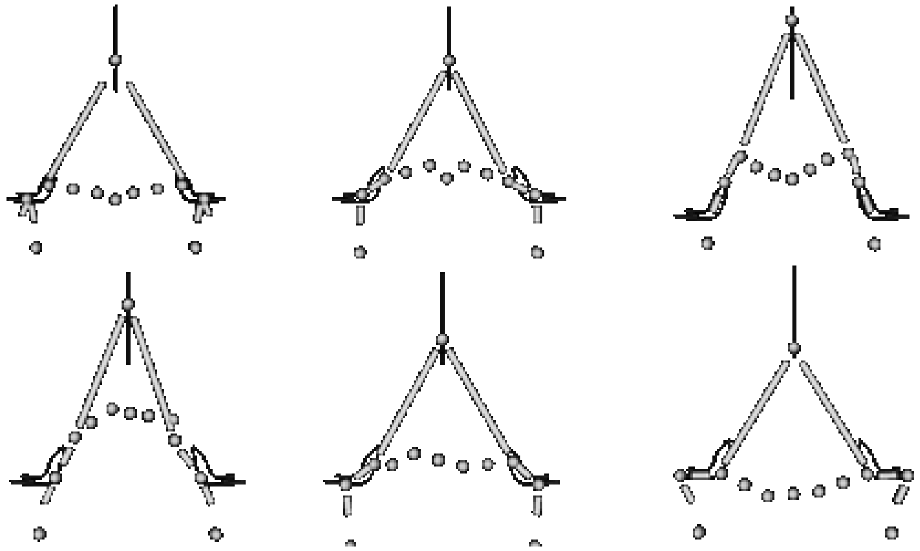


Figure 8: A few animation steps of a letter A. The "knees" and the head of the letter are directed by three cyclic key framed movements. The rest is following, through more or less loose articulations, implemented using *Displacement Constraints*.

to remove the unnecessary computation in such cases. Other optimizations are presented below.

7.1 Time Coherence

Usually, computer animations are described with a few key positions, then computed sequentially with a relatively large number of intermediate positions. This means that equations 1 and 2 should be factorized in the following form to split between what changes at each time step, and what does not:

$$\begin{aligned} P(u) &= (u^3\ u^2\ u\ 1)\ \mathrm{Hermite}(A, B, T_A, T_B) \\ Q(u) &= \exp\left[(u^3\ u^2\ u\ 1)\ \mathrm{Hermite}\right. \\ &\qquad \left.(\log Q_A, \log Q_B, \log QT_A, \log QT_B))\right] \end{aligned}$$

Using this decomposition, the matrix $\times$ vector products, and the log computations need to be performed only once when we change from one spline segment to the next one.

This is implemented transparently as a cache. We store the eight pre-computed vectors, and the index of the segment in which they are valid. When the index matches, the computation is reduced to two linear combinations of vectors, and an exponentionation to get the quaternion.

7.2 Curvilinear Abscissa and Speed Integral Caching

The curvilinear abscissa $l(u)$ is used in section 5.1 to compute the u parameter from a given abscissa l, which means that the function must be inverted. [WW92] proposes to use a Newton search coupled with a Runge-Kutta integration of the curvilinear abscissa.

To get the fastest possible results, we preferred to tabulate the $l(u)$ function that maps the parameter value u to the abscissa. We choose a fixed number of subdivisions between two successive key-positions, and store a cumulative table of the curvilinear abscissa function of u. A fast binary search to find the interval, and a linear interpolation inside it are sufficient to map any abscissa l to the corresponding parameter u. This table is partially invalidated each time a keyframe is moved, and completed when needed.

Also, the speed function v has to be interpolated and integrated to get the L function. This is also done through a stored table that is partially invalidated and recomputed when needed. But two differences should be noticed. First, search is useless, because the table is used in the direct way (the table entry t directly gives the distance $L(t)$ covered so far). Also, linear interpolation between positions stored in the table corresponds to a succession of constant speed steps. This is not what the animator would expect from a smooth speed curve. Hence, a single trapeze has to be integrated between the last point in the table and the point to compute for.

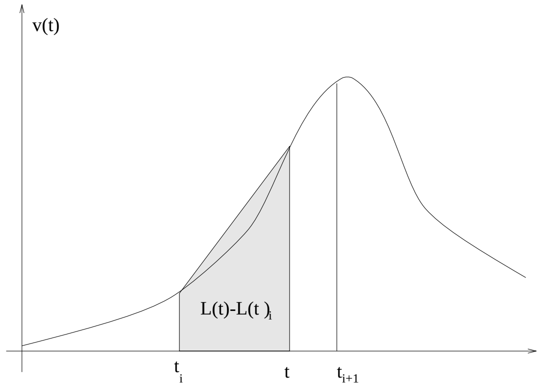


Figure 10: The speed integral table, and the last trapeze.

8 Conclusions

In this paper, we advocate the use of Hermite splines, and introduce several extensions to the [KB84] approach. First, our dual mode enables to do both *by-hand* and *tcb* style edition. Next, the use of logarithms of quaternions enables to apply exactly the same fast formula used for translations, and also to share *tcb* parameters to simplify the user interface. Design of complex motions is done using *Fabule* features to link or compose more basic descriptions. The uses of a tabulated "three interpolants" algorithm enables us to have a powerful, robust and fast tool to design quickly prototyped or finely tuned motions.

Acknowledgements

Part of this work has been implemented by Christophe Lyon, for his *Diplôme d'Etudes Approfondies* (DEA) of computer science during summer 1994 [Lyo94]. Thanks to Agata Opalach for designing the A animation during September 1994.

References

[Duf86] T. Duff. Splines in animation and modeling. *State of the Art in Image Synthesis (SIGGRAPH'86 course notes Number 15, Dallas, TX)*, 1986.

[Gas94] J.D. Gascuel. Fabule : Un environnement de recherche pour l'animation et la simulation. In *Troisième séminaire du groupe de travail "Animation et Simulation" : Les Simulateurs*, pages 31–40, October Paris, France, 1994.

[GG94] M.P. Gascuel and J.D. Gascuel. Displacement constraints for interactive modeling and animation of articulated structures. *The Visual Computer*, 10(4):191–204, March 1994. An early version of this paper appeared in the *Third Eurographics Workshop on Animation and Simulation*, Cambridge, UK, Sept 92.

[Han92] G. Hanotaux. Interactive control of orientation interpolations. In *Third EG Workshop on Animation and Simulation*, 1992.

[Han93] Gabriel Hanotaux. Techniques de contrôle du mouvement pour l'animation. Thèse de doctorat, Ecole Nationale Supérieure des Mines de Saint-Étienne, Université de Saint-Etienne, April 1993.

[HP93] G. Hanotaux and B. Peroche. Interactive control of interpolations for animation and modeling. In *Graphics Interface*, pages 201–208, 1993.

[KB84] D.H.U. Kochanek and R.H. Bartels. Interpolating splines whith local tension, continuity, and bias control. *Computer Graphics*, 18(3):33–41, 1984.

[Las87] J. Lasseter. Principles of traditional animation applied to threedimensional computer animation. *Computer Graphics*, 21(4):35–44, 1987. Proc. SIGGRAPH'87.

[Lyo94] Ch. Lyon. Introduction de l'animation descriptive dans un contexte d'animation sous contraintes. *Rapport de stage*, DEA d'Informatique de l'ENSIMAG/UJF, September 1994.

[Sho85] K. Shoemake. Animating rotation with quaternion curves. *Computer Graphics*, 19(3):245–254, July 1985. Proceedings of SIGGRAPH'85.

[Sho87] K. Shoemake. Quaternion calculus and fast animation. In *SIGGRAPH Course Notes*, volume 10, pages 101–121, 1987.

[WW92] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques. Theory and Practice.* Addison-Wesley, 1992.

[YG89] H. Yahia and A. Gagalowicz. Interactive animation of object orientation. In *PIXIM*, pages 265–275, Paris, France, 1989.