Scripting and Artistic Control in Computer animation

Alan Watt
Mark Watt

IMPA - Instituto de Matematica Pura e Aplicada
Estrada Dona Castorina, 110
22460 Rio de Janeiro, RJ, Brasil

Xaos, San Francisco, EUA

Abstract

This paper presents experience in using curves as a script base in a variety of computer animation techniques. We present a general model of motion control in animation and look at practical examples in the context of rigid body animation, the animation of soft bodies or deformation and the animation of articulated structures. Experience suggests that the conceptual difficulty that a non-numerate or non-computer literate animator may have in relating the script to its effect, is soon overcome. An animator very quickly becomes used to using and editing curves that specify some aspect of motion. The methods employed are discussed in tne context of how best to give complete artistic control to the animator while at the same time insulating him from the tedium of specifying movement in a low level or detailed form.

1) Introduction

The script or interface process can be categorised as enabling a specification and interactive modification of motion. We can call the means that facilitates this process an animation language. Thus we have the analogy between an animation language advantages that accrue in high level languages carry over into animation languages - in particular being able to develop scripts of greater and greater complexity by developing simpler scripts (bootstrapping), and being able to store past experience as procedural modules reusing them when required.

The motion control process can itself be subdivided into two parts. In the first process we supply an interface with which the animator interacts. The information given by the animator is then interpreted by the second process which provides low level information for the renderer. Thus we have an animation language/compiler model.

Consider again rigid body animation. To protect the animator from supplying a list of three-dimensional coordinates as a function of unit time intervals (a low level motion specification) we provide an interface or script facility that enables the animator to describe the motion in high level terms (Figure 1). We distinguish between an interface and a script at this point by saying that a script provides a record of the animation that can subsequently be retrieved and edited. Also we distinguish between full explicit control, where the animator specifies the changes in the animated attribute of the object in every frame, and high level or automatic control, where the animator may specify , for example, that an 'actor' is to walk from a point to a destination by the most direct route. There is always a trade-off between artistic control and the work involved in creating an animation. As we tend towards higher level facilities more control is taken from the animator by the software writer. In the authors' experience animators tend to prefer to do more work than use a system that tends to limit their art form at a detailed level. A practical example of this trade-off is given in Section 3.

In all that follows we shall be talking about full explicit control. Full explicit control is necessary for general purpose animation wherein the animator is free to create novel motion. Other important attributes that any scripting system should possess are as follows. Editing should be possible so that incremental design is facilitated and previous experience can be built upon. A script should be a high level summary of the motion so that it is obvious to the animator what alterations are necessary to achieve the desired effects. The script form should be accessible to a non-computer-literate animator.

The success of any script form depends to some extent on the conceptual distance between the motion and its specification. So that, for example, a script written in ASAS (an animation language built on top of LISP) is difficult to 'read'. That is, it is difficult to imagine the movement by reading the program or script. This distance can be reduced if the animator is able to participate in a real time loop, previewing the animation as the changes are made to the script. This is usually possible nowadays on graphics workstations, at least for wire frame objects. (Such a facility - wire frame previews - is somwhat similar to the line tests made by traditional animators. Here an animator makes pencil sketches without colour and background to test out the character of the movement.)

Three general types of animation languages have emerged. [Foley et al(1990)] categorise these as linear list notations, general purpose languages with embedded animation directives and graphical languages. We concentrate, in this paper, on using curves as a basis for animation graphical language. In particular we confine ourselves to a general discussion of curves and motion control. Facilities that would be required to make a complete animation language are not discussed.

An early example of linear -list notation is given in [Catmull(1972)]. The motion of a rigid body is specified to take place between two frames. For example:

```
37,48, A, ROTATE "BOX", 1,30
```

specifies that the object called BOX is to be rotated through 30 degrees about axis 1. The motion is to take place between frames 37 and 48 and the rotational increments are to be deteremined from table A.

In the second category general purpose languages are enhanced to include animation utilities and types. Examples are ASAS [Reynolds(1982)], a superset of LISP and CINEMIRA [Magenat-Thalmann(1985)] - an enhanced version of PASCAL. Because these languages retain the advantages of general purpose high level languages, actions can be embedded in procedural modules. A script now becomes a program but the greater flexibility of this approach, over the linear list notation, is paid for by the requirement that the animator must be a skilled programmer. For example, in CINEMIRA animattion types can be declared and the following defines a vector that starts at time 10 and moves with constant speed <<3,0,0>> from the point <<0,10,4>> stopping at time 13:

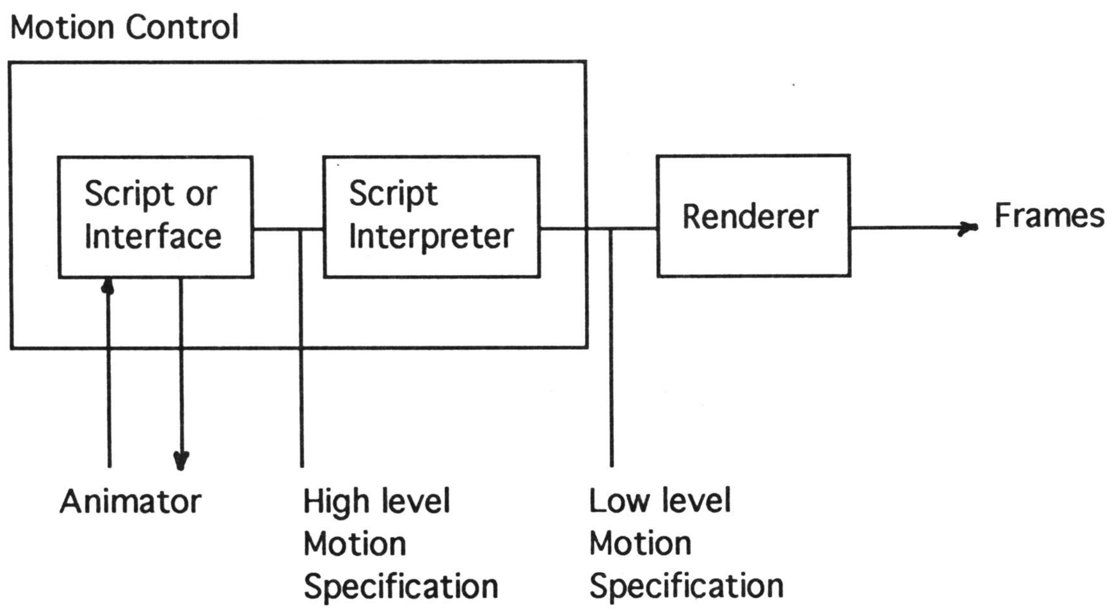

Figure 1 A model of a scripting system for motion control

```
type TVEC = animated VECTOR;
 val <<0,10,14>> .. UNLIMITED;
 t me 10..13;
law <<0,10,14>> + <<3,0,0>> *
                    (CLOCK - 10);
end;
var VEC: TVEC;
```

The following example is a single **animate** block in ASAS. It contains two actors - a green spinning cube and a blue spinning cube:

```
(script spinning-cubes

 (local: (runtime 96)
         (midpoint(half runtime)))

 (animate (cue (at 0)
          (start (spin-cube-actor green)))

      (cue (at 0)
      (start (spin-cube-actor blue)))

         (cue (at runtime)
      (cut)))
```

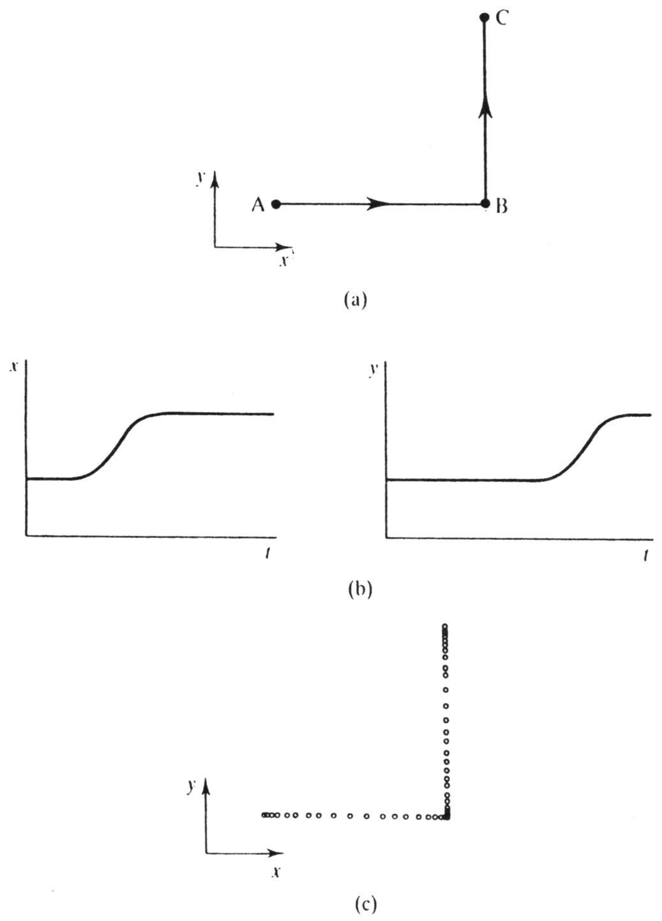

Figure 2 Baecker's motion control specification
a) The path of a body moving from point A to point B
b) Motion characteristic as two distance time curves.
c) Motion characteritic as a P-curve. The dots represent points on the path curve at which frames are generated.

These examples demonstrate the point that access to the animation facilities requires a high degree of programming skill.

Both linear-list notation and enhanced high level languages have drawbacks as far as our list of desirable attributes is concerned. We propose that these can be overcome by using curves as a script form and we demonstrate the efficacy of such a technique in three animation forms:

rigid body animation
animation of deformation - soft body animation
articulated structure animation

2) Curves in rigid body animation

This form of computer animation is ubiquitous. Perhaps its most common manifestation is 'tumbling logo' sequences on TV. Much work has been done in this area and this section describes an implementation of an approach that has been implemented in different practical manifestations by other authors - notably [Baeker(1969)] and [Steketee-Badler(1985)].

The work described in this paper has its origins in a system developed by Baecker in 1969 [Baeker(1969)]. In this work Baecker introduces the concept of a P-curve, describing this concept with the following example. Consider the motion of a body moving from point A to point C, diagonally opposite, by following a path along a wall to corner B, then from B to C (Figure 2a). We can plot the variation of the position in x and y of the body as a function of time and from Figures 2b and c we can see that the body starts from rest , accelerates and then decelerates into the corner. A pause occurs at B then the body accelerates and decelerates on its path to point B. Figure 2d is the same as Figure 2a except that superimposed on the path characteristic are markers that represent equal intervals in time. These, in the context of animation, are frame markers - the points at which the renderer ,using information given in the characteristics, generates a frame. Baecker terms such a characteristic a P-curve and we note that it contains both positional information together with the dynamics of the motion. If we were using such curves as an interface then we would control the dynamic characteristic , either by changing the shape of the curves in Figures 2b and c, or, by altering Figure 2d to change the spacing between frame markers.

In traditonal animation such scripting of the dynamics of a character is achieved by altering the difference or distance between the character's position in consecutive drawings. Acceleration and deceleration is called 'easing in' and 'easing out'. Such control via the

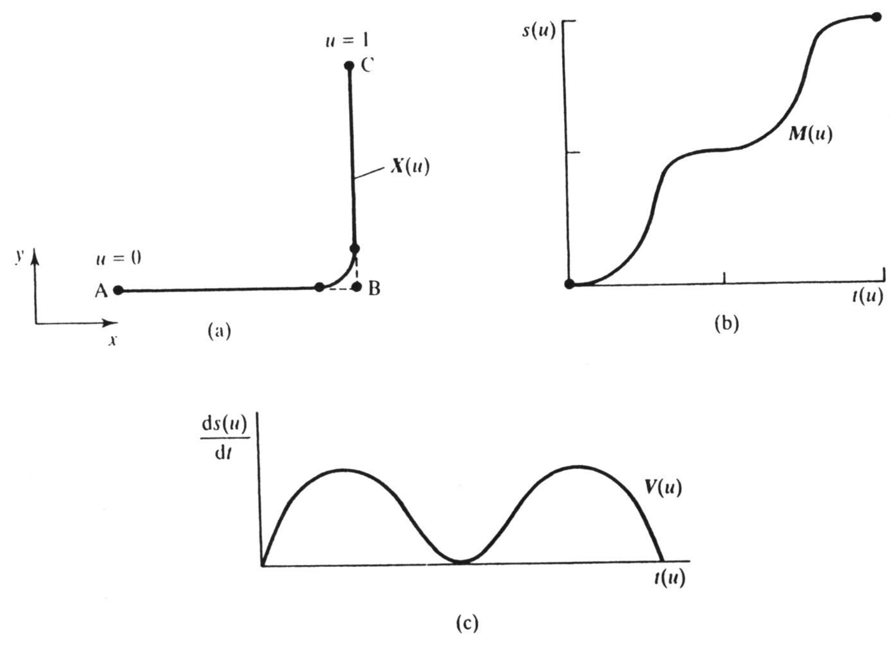

Figure 3 a) Path characteristic as a parameteric function.
b) and c) Single motion characteristics specifying velocity or acceleration along the path.

drawings is particular for the character and is a fine skill of the animator. The character could be a walking figure with arm and leg movements or it could be a motor car.

A curve system expresses the dynamic characteristics globally - the movement is scripted, summarized and is in a form that can be easily edited - by changing the shape of the curves. Having an animation sequence specified by a global characteristic is an attractive proposition and its the key to interactive animation design. By curve editing the animator can view and alter the motion characteristics of the entire sequence. Also note that the dynamic or velocity characteristic can be separate from the position characteristic. Unlike the method of traditional animation the designer can concentrate separately on the design of the position and velocity characteristic. For example, he can experiment with different velocity characteristics while using the same position characteristic.

The system that we have used takes the above basic idea with certain simple changes. First we use a parametric representation for the path and velocity characteristic. Figure 3a shows the positional characteristic as a parametric function **X**(u), where point A occurs at u = 0 and point B occurs at u = 1. This allows us to have a single velocity characteristic that specifies the motion along the path and this can either be **M**(u) or **V**(u) as shown in Figures 3c and d. The other advantage that this scheme confers is that we can use splines for the characteristic thus facilitating a convenient editing device for changing the shape of the curve. We have found that non-computer-literate animators become very quickly used to this dual parametric specification and the curves can be displayed in windows alongside the window in which the wireframe animation is displayed. The parametric representation is invisible to the animator and a low level motion specification is derived from the curves. Our experience has shown that animators find the concept intuitively simple. In three space the path or motion characteristic is of course a function of three spatial variables requiring something like an interface display of three projections.

At this stage we mention that although we are only discussing the issues involved in an animation language, there are problems of some significance that the compiler has to deal with. The most significant of these concerns the interpretation of the information that is presented as a parametric function. To derive the information necessary for the frame generation we need to measure equal distances along the path length. Unfortunately equal intervals in u do **not** correspond to equal distances along the curve (the arclength) and when such a system is implemented this distinctly non-trivial problem must be addressed. Of course, this is all invisible to the animator and full details of finding equal arclength intervals is given in [Watt-Watt(1992)].

The velocity curve can , of course, be used for any motion parameter; there is no reason for it to be restricted to arclength. The term then encompasses anything that moves in the animation sequence apart from the usual kinetic variables such as position and orientation. Movement could also include colour, transparency or any attribute that is represented by numbers.

A similar system was developed by Steketee and Badler [Steketee-Badler(1985)]. and they were the first to recognize the value of separate specification of motion and velocity characteristics. Thus the use of such an approach is now fairly standard and we will now show how it can be extended to script more unusual modes of animation, namely articulated structures and deformation.

3) Curves in articulated structure animation

We now look at a case study of an articulated structure animation that uses forward kinematics. All the advantages of using curves as a script carry over into this animation form. In addition it is possible to script the animation of complex articulated structures with many connected parts. Of course it could be argued that the animation of an articulated figure, that performs a task such as walking, playing a musical instrument or a sport, is best carried out by using a high level or goal directed system. However, apart from the fact that such systems are in their infancy, they are usually artistically restrictive. There is always a trade-off between the freedom that an animator requiresto design a unique sequence (implying general or low level facilities) and the facilities that may be provided by a goal directed system. After all we are talking about the ability to provide characterization or personality to the movement - a requirement that is often absent from commercial animation systems. For example, say we require a character to walk from point A to point B. We may, in a goal directed system, to specify the speed of the walk, but in all probability we would be unable to specify that the character of the walk was to be a 'pimp roll'. The subtleties of the pimp roll are shown in Figure 4.

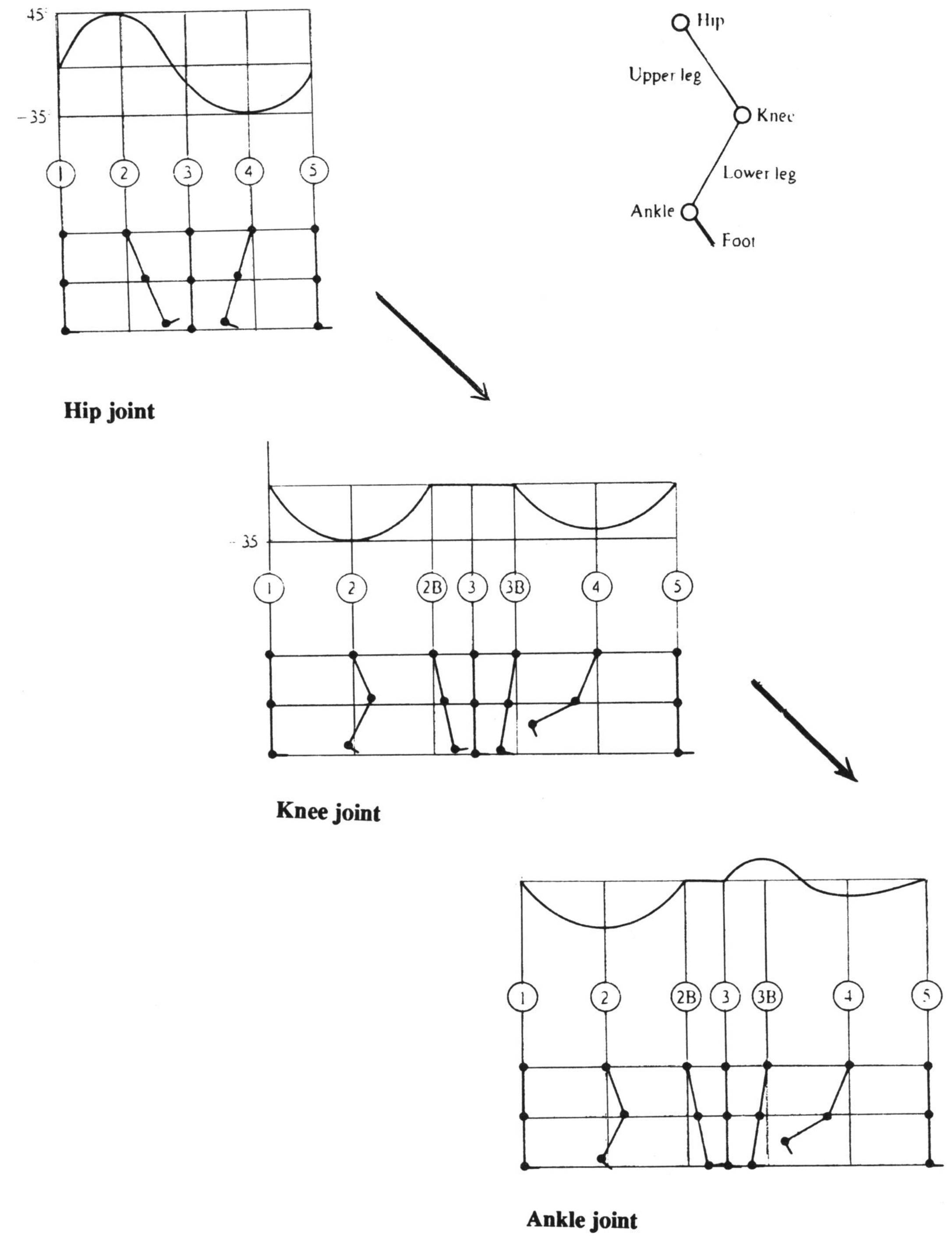

Figure 7 Animating a simple articulated structure.

Let us consider the nature of the problem. An articulated figure is normally a structure that consists of a series of (usually rigid) links connected at joints. Animating such a structure means assigning movement to all the nodes, in general. Thus the nodes are treated as individual objects and animated in the same way that we animated rigid bodies in the previous section. The difference now is that the objects or nodes are linked. The structure is represented as a hiearchy and any movement assigned to a node effects all nodes beneath it. In our example rotating the shoulder rotates the entire arm. In such a system an animator has to start at the top and work down, perfecting the animation at each node before going deeper into the hierarchy. The need to animate each node is the source of the workload in a full explicit or forward kinematic motion control system for an articulated structure.

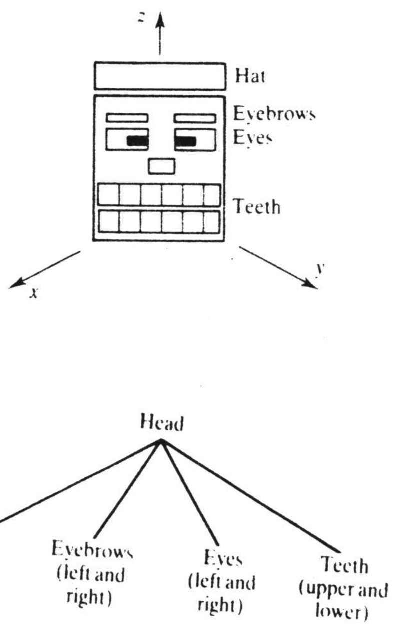

Figure 5 The head of a character and the local hierarchy. The links in the hiearchy are elastic and the hat, eyebrows, eyes and teeth can inherit the movement of the head or not as required by the sequence.

The following examples are taken from a production made for a rock music program shown on Channel 4 , UK in 1988. The complete animation features a band of musicians which are composed, in the main, of spheres and cylinders. Since only a very few modelling primitives are used for the characters, the success of the animation derives entirely from the complexity and subtlety of the movement. Since each character consists of a somewhat complex hierarchy of linked moving parts, the animation of the entire sequence would have been extremely difficult and tedious without the use of curves.

In the first example the nodes themselves are animated as objects. The links between the objects are invisible and elastic as the first example will demonstrate. The second example is a more conventional articulated structure. Here the links are the objects, in this case limbs.

As an example we consider controlling the animation of just the head of one character. This has a hierarchy shown in Figure 5. The head is subject to three possible movement categories.

1) Global movement, where the entire head is animated using motion inherited from movement of the entire structure. This is equivalent to considering the entire character as a rigid body whose global movement would be scripted as in the previous section.

2) Local movement of the head. In this case the head can tilt and rotate about the z axis.

3) Local movement of the hat, eyebrows, eyes and teeth. As well as inheriting the global and local movement of the head, these structures can have their own movement. Their links to the head are 'elastic'.

Considering that this is just one of a small part of one character in the animation, scripting without the use of curves would have been practically impossible.

We consider now a small characterization in the complete sequence. The character (called Mad Bastard) snaps at the camera then turns his head away; but the eyes continue to stare at the viewer before turning also. To animate the local movement of the head and eyes we use just two curves similar to P-curves of the previous section. The path characteristic is unnecessary because the local movement of the head is restricted, in this sequence, to rotation about the z axis, and the eyes can only translate over the cylindrical surface of the head. Thus we combine motion and velocity specification in one curve as shown in Figure 6a and b. Figure 6a shows the head's z rotation that was used. In this case the eyes inherit the head rotation and move as if they were rigidly linked to the head. The concept of an elastic link is implemented by applying a compensatory motion to the eyes in the form of a z rotation as shown in Figure 6b that causes the eyes to lag behind the head movement.

In the second example the objects to be animated are the links themselves. We do this by scripting their rotation about the joints which are the nodes. Consider the development of a simple walk cycle. Starting at the hip joint we specify a hip rotation script - angular displacement as a function of time (Figure 7). This completed, the knee and then the ankle is scripted. This is somewhat simplified - other refinements are necessary - the hip, for example, does not move in a straight line. Although, even with curves, full explicit control may seem extremely tedious, bear in mind that an animator can be supplied with a standard script for a walk cycle giving the facility for characterization by altering a previously stored script.

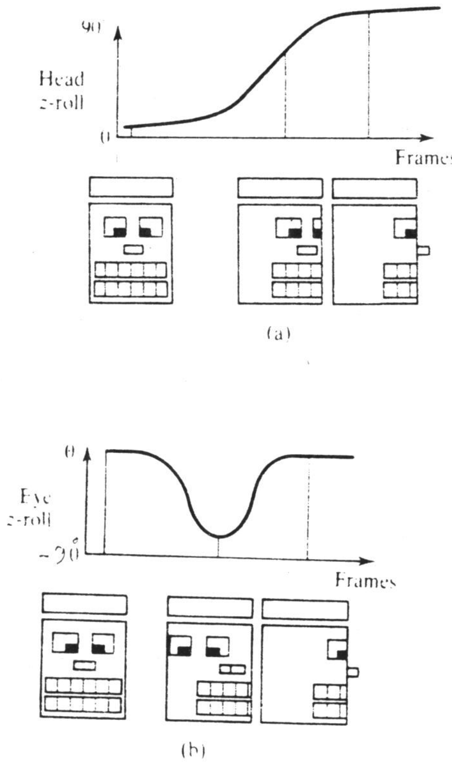

Figure 6 Script curves for animating the head z-roll and the eye z-roll. Note that the eye z-roll can be viewed as compensation for the movement inherited from the head.

4) Curves in the animation of deformation

One of the most useful techniques for deforming an object was first suggested by Bezier in [Bezier(1972)] and was introduced into computer graphics by Sederburg and Parry [Sedeburg-Parry(1986)], where it was given the name FFD (Free Form Deformation). To understand how to control the motion of the deformation of an object we need to explain some facts concerning this method. We define a Bezier hyperpatch as the solid:

$$Q(u,v,w) = \sum_{i=0}^{3} \sum_{j=0}^{3} \sum_{k=0}^{3} p_{ijk} B(u) B(v) B(w)$$

a volume of space parametrized by the parameters u, v and w. Such a hyperpatch is specified by 64 control points. For example, if the control points are equally distributed in a cube then the hyperpatch is a unit cube in parametric space. Moving any of the 64 control points distorts or deforms the hyperpatch. Now if we use the hyperpatch as a space in which we embed , say, a polygon mesh object, we can deform the object by moving the hyperpatch's control points. This distorts the hyperpatch itself which pulls the vertices of the object embedded in it. Just as the distortions of the hyperpatch that result from moving control points are 'intuitive' (that is, the control point movement induces a distortion that is concepttually related to the this movement) so also is the distortion of the embedded solid.

Thus the animator has the concept of a rectangular three-dimensional array of control points called FFD blocks that are motion controlled to effect the animation of deformation. An FFD block is generally a stack of hyperpatches, but for the purpose of this paper we ignore this distinction (see [Watt-Watt(1992)] for further details on this point).The script curve method that we have adopted is best explained by reference to an example. The animated sequence (Figure 8) shows a spoon jumping through the air and landing. The animation is carried out by superimposing deformation animation on the rigid body animation that controls the jump. The deformation animation uses two FFD blocks that control the upper and lower halves of the spoon. These are shown in the illustration as grey transparent blocks. The idea of this sequence is to emulate a human

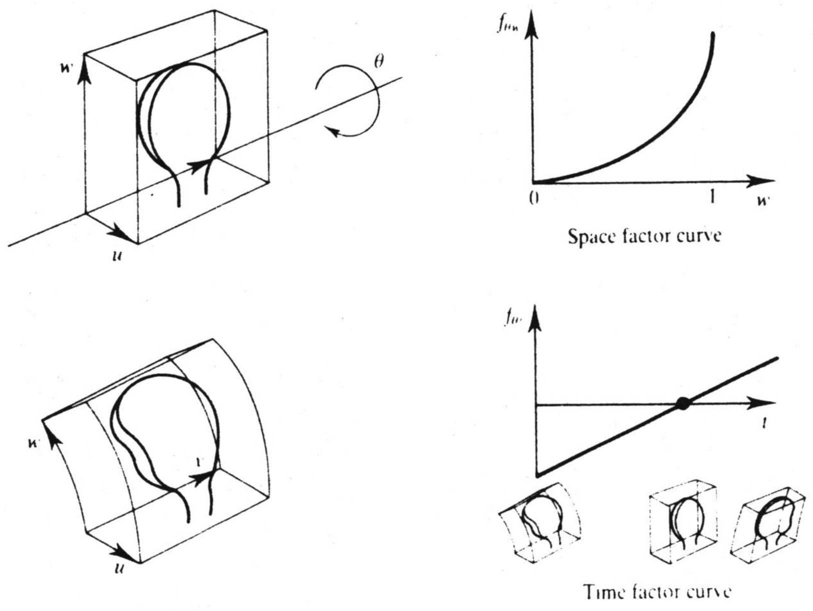

Figure 9 Scripting deformation by controlling the movement of points in an FFD box.

jump . A human, when jumping has its head thrown back, and on landing, has its head brought forward by momentum.

The deformation is applied along the vertical axis of the FFD block (Figure 9). The required animation of the block is rotation about the v axis. Additionally since we are trying to bend the block we need to apply greater values of rotation angle for increasin w. This is achieved by using two curves called 'factor' curves - a space factor curve and a time factor curve that modify the parameters of the deforming transformation according to when and where they are applied.. The space factor curve Ø(w) is aligned along the w axis and increases with increasing w. The higher a point within the FFD block the greater the value of Ø so the more the block gets bent. Ø(w)= 0 ensures that the bottom of the block remains undeformed. The time factor curve Ø(t) is set up with a large negative value at the start corresponding to the upper spoon being bent backwards, moving through zero where the block will be undeformed, to a smaller positive value as the head gets thrown slightly forward. The deforming transformation for a point (u,v,w) is then given by:

$$Ø = Ø(w)\,Ø(t)$$

Animators have found the use of factor curves in controlling deformation highly intuitive.

5) Conclusions

Full explicit control allows complete artistic freedom. This can be acheived by using curves as the basis of an animation language which can then be used in many animation forms. Because curves are a succint summary of motion, complex animation involving articulated structures is possible.

6) References

Baecker, R.M., "Picture Driven Animation",SJCC, AFIPS Press, Montvale, NJ, 1969, 273-288.
Bezier, P. ,"Numerical Control: Mathematics and Applications",Wiley, 1972.
Catmull, E., "A System for Computer Generated Movies", Proc. Annual Conference, ACM, New York, NY, Aug 1972, 422-431.
Foley, J.D., Van Dam, A., Feiner, S.K. and Hughes, J.F. "Computer Graphics: Principles and Practice", Addison-Wesley, Reading, MASS, 1990
Magenat-Thalmann, N., and Thalmann, D., "Computer Animation: Theory and Practice", Springer-Verlag, Toyko, 1985.
Reynolds, C.W., "Computer Animation with Scripts and Actors", SIGGRAPH '82, 289-296.
SEDE85 Sederburg, S.W. and Parry, S.R., "Free Form Deformation of Solid Geometric Models", SIGGRAPH '86, 151-160.
Watt, A. and Watt, M, "Advanced Animation and Rendering Techniques" Addison-Wesley, Wokingham, UK, 1992.

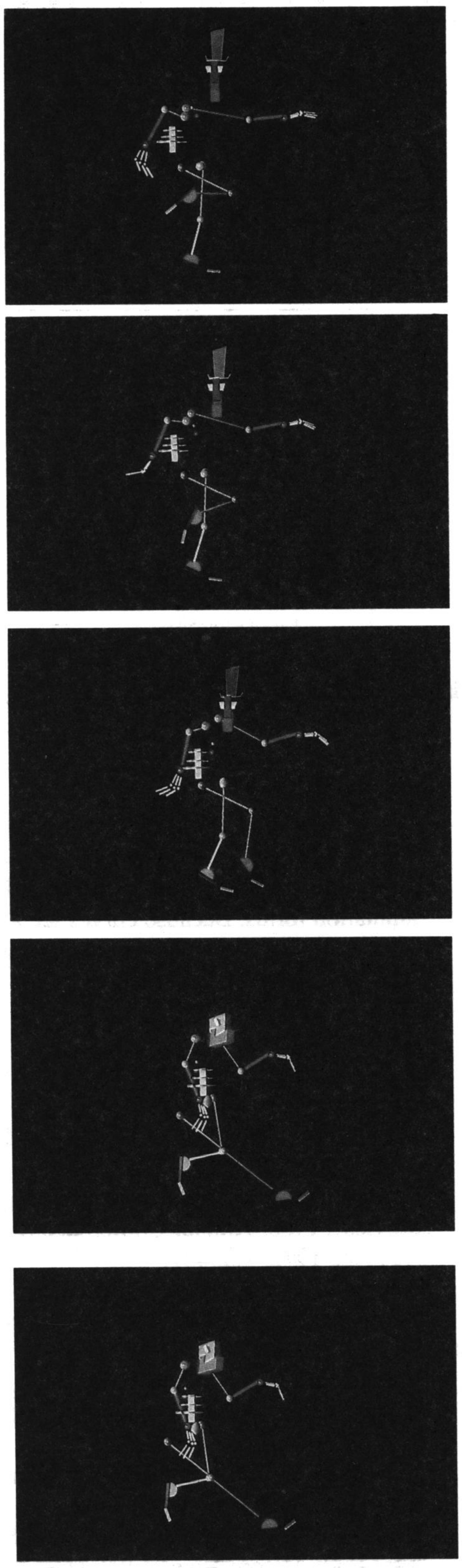
Figura 4

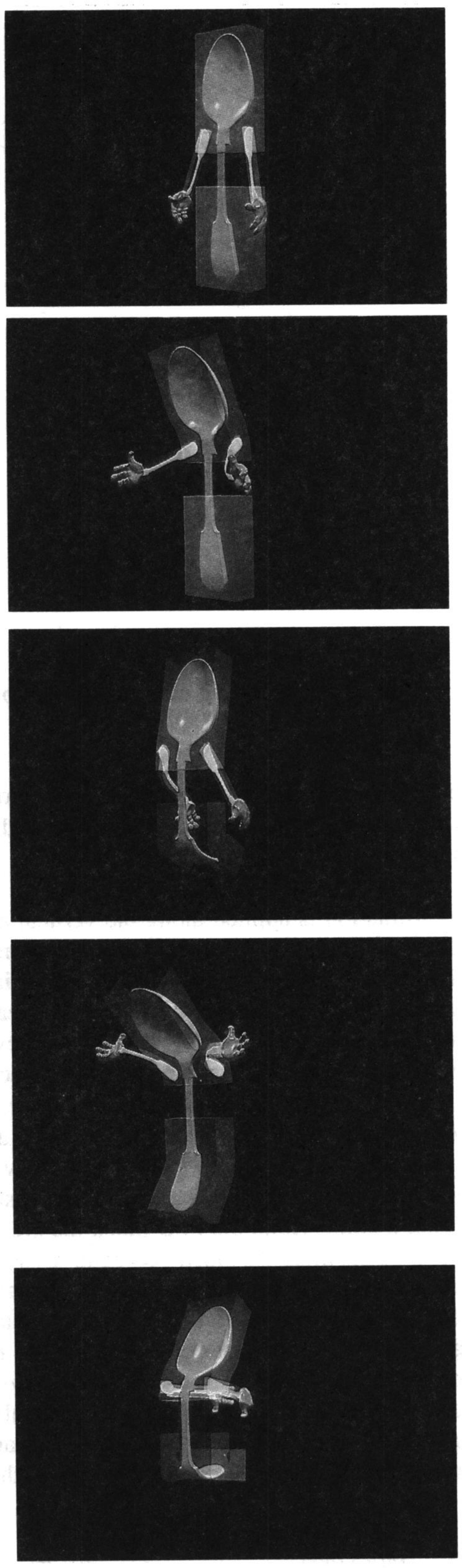
Figura 8