Working Paper Series
ISSN 1170-487X

One Dimensional Non-Uniform Rational B-Splines for Animation Control

by Abdelaziz Mahoui

Working Paper 00/7
March 2000

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Abstract

Most 3D animation packages use graphical representations called motion graphs to represent the variation in time of the motion parameters. Many use two-dimensional B-splines as animation curves because of their power to represent free-form curves. In this project, we investigate the possibility of using One-dimensional Non-Uniform Rational B-Spline (NURBS) curves for the interactive construction of animation control curves. One-dimensional NURBS curves present the potential of solving some problems encountred in motion graphs when two-dimensinal B-splines are used. The study focuses on the properties of One-dimensional NURBS mathematical model. It also investigates the algorithms and shape modification tools devised for two-dimensional curves and their port to the One-dimensional NURBS model. It also looks at the issues related to the user interface used to interactively modify the sahpe of the curves.

Contents

List of Figures

Chapter 1
Computer Animation

1.1 Introduction

To animate, literally means to bring to life. Animation has been very popular, long before the existence of computer graphics. In traditional animation, motion is created by drawing objects in different positions over successive frames such that when played in sequence, they give the perception of motion. Computer animators have borrowed the same paradigm. A definition of computer animation could be [8]: a technique in which the illusion of movement is created by displaying on the screen, or recording on a recording device a series of individual states of a dynamic scene. This definition highlights the fact that all animation techniques exploit the human visual ability to integrate a sequence of individual images into a sensation of visual continuity.

Computer animation has found its way into a variety of fields ranging from entertainment to practical applications, education and scientific visualization. In the world of television advertising and the film industry, many of the special effects are computer animations, and some movies are generated entirely by computers; Walt Disney and Pixar's movies `Toy Story` and `The Bug's Life` are very good examples of that. Some well known practical applications of computer animation are flight simulators for aircraft which are used to train pilots (civil and military), MRI animation in medicine. Applications of computer graphics and animation in scientific research are grouped under the heading of *scientific visualization*. When scientists experiment with new mathematical and physical models, they use visualization as a mean of validating the designed model and animation becomes a natural way of representing the temporal evolution of the model results.

There is an important distinction to make between animation and simulation. In an animation, a set of frames is generated, in different ways and possibly at different times, and sequentially visualized at a given rate to produce a sequence. Simulation on the other hand can be defined as an animation sequence that reproduces natural phenomena or is a visualization of the temporal evolution of mathematical and physical models in real time.

In any case, the main goal of computer animation is to synthesize the desired motion effect which may correspond to a natural phenomena, an animator's imagination or a scientific visualization. There is a large number of computer animation systems providing tools able to translate this goal into reality. They are as diverse and disparate as the fields that use them. A common way of classifying the mainstream ones is according to the motion control techniques they use [1], [4]. High level animation systems allow users to specify motion in an abstract way, whereas low level systems require the users to specify all the motion parameters. The next section describes the different motion control techniques.

1.2 Motion Control Techniques

Three families of motion control techniques are traditionally distinguished [4], [8]: descriptive or guiding models, generative or Physics-based models, and behavior or task-oriented models. Table 1 summarizes the motion control models and the associated animation systems.

1.2.1 Descriptive or Guiding Models

In guiding systems, the behavior of each animated object is explicitly specified by describing the variations in time of its attributes. One of the fist developed animation techniques and extensively used by commercial animation packages is *Keyframing*. The user explicitly specifies the kinematics by setting transformation values (any parameters affecting the motion) in keyframes; *in-between* frames are generated by the computer using interpolation [6] methods which may be linear, based on splines [11] or on quatrains [6, 7]. Another group of descriptive techniques is *procedural methods*. In this case, the kinematics are specified using procedures. There are *script-based systems* where the procedure is a script written using an animation language:

there are also systems using *direct* and *inverse kinematics* to develop algorithms for generating the motion.

1.2.2 Physics-Based Models

In Physics-based systems, the geometry and shape changes of the animated objects are modelled using mathematical and physical principles. Motion is obtained by the *dynamic equations of motion*. These equations are established using the forces, the torques, the constraints and the mass properties of objects. Consequently, the term simulation is more often used instead of animation. There are two approaches for using dynamics: *direct dynamics*, where forces and torques are known and motion is obtained with minimal control over the system; *inverse dynamics*, where the motion is known and forces and torques unknown and computed.

1.2.3 Task-Oriented Models

In task-oriented systems, the behavior of animated objects is specified in terms of events and relationships [8]. In other words, the objects have a perception of the environment, and an ability for decision and action making and communication abilities. In these systems, the goal is to produce controllers that make decisions according to the information they receive. To do that, they use techniques such as genetic programming techniques, sensor-actuator networks and finite state machines [4].

1.3 Motion Parameters

Regardless of the motion control technique used, computer animation systems have to deal with one or more elements that vary in time: several parameters have to be controlled within every element. The selected parameters are those who's values when changed have a visual effect on the scene. These parameters are called *motion parameters*. They may correspond to the position, velocity, geometry, color, transparency, structure, texture of an object, to changes in lightning, camera position, orientation, and focus, to rotation, scaling operations. The animation system controls each motion parameter by generating the appropriate parameter value for each frame. The parameter values are generated in a variety of ways, including the use of mathematical formulae in physics-based systems, interpolation in generative

systems, or externally derived data. In any case, the result is a table that maps a value of a motion parameter with the associated time or frame number.

1.4 Motivation

The current tendency in computer animation systems development is the design of user-friendly control systems which, from the user's point of view, do not require in-depth knowledge of the underlying algorithms and motion equations. The challenge is to obtain user-controlled motion with reduced computational complexity and computation times.

In this context, there is a powerful approach that enhances the user-friendliness of parts of a control system. Many 3D animation packages use a graphical representation, called a *motion graph*, of the variation in time of the motion parameters. In the appendix are presented some screenshots of motion graphs used in commercial 3D animation packages. Most software packages work in similar ways. Typically, they plot a motion parameter (vertical axis) against time or frame number (horizontal axis). When building an animation, the animator often needs to adjust some of the parameter values. Consequently, he has a need to interactively modify or edit the parameter curve and thereby to directly modify or edit the animation itself. To remedy to that, many of the animation packages use two-dimensional *B-splines* as animation curves. B-splines provide the power to represent free-form curves and surfaces and common analytic curves. They can interpolate, approximate or fit virtually any set of data points. They are suitable for *ab initio* design. Consequently, they are a very good choice for the situation described above.

having said that, the current implementations of motion graphs do not necessarily reduce the computational complexity or the computation costs. To illustrate some of the issues, we will be considering the simple situation where a two-dimensional B-spline curve is used to represent the evolution of the velocity $\mathbf{V}(t)$ of an object over time t (Figure 1.1). The first step is to produce the curve itself. The curve, denoted by $\mathbf{C}(u)$, has two components: $\mathbf{C}(u) = (x(u), y(u))$. In order to generate it, two sets of computation, one for each of the component ($x(u)$ and $y(u)$), have to be carried out. If the number of motion parameters is important,[1] the computation

[1] When the number of parameters is considerable, one has to think wether this type of motion control is appropriate; however, there might be situations where this approach is not avoidable like

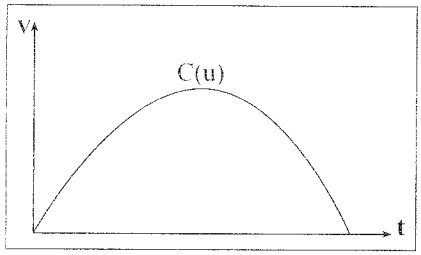


Figure 1.1: Velocity curve.

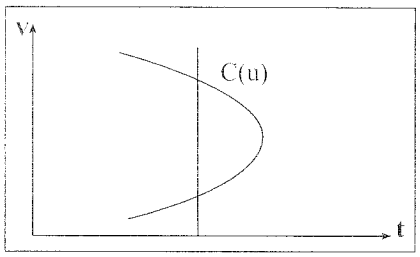


Figure 1.2: A double-valued velocity curve.

cost becomes an issue. This is one aspect of the computation cost.

Obtaining the velocity value V at a given time t is not straightforward since the curve $\mathbf{C}(u)$ is parameterized in terms of u; its components are functions of the parameter u. A reparameterization, known as *arclength* or *chord reparameterization* [1, 6], is usually required that reparameterizes the curve as a function of time t; this reparameterization in itself is not straightforward. Overall, there is not direct and intuitive mapping between the curve input/output and the corresponding sought parameter value. This point shows one aspect of the computational complexity.

Because of the animator ability to modify the B-spline curve freely, a non-desired situation may occur while the animator is editing the curve; this situation is illustrated by Figure 1.2. It shows that the object has more than one velocity at the same time which is a physically meaningless situation. Of course, if the animator is careful enough, this may never happen. However, this is an *intrinsic* behavior of the B-spline, in other words, there is nothing in the underlying model that can completely avoid the situation from happening.

This work investigate wether it is possible to address these issues by means of using one-dimensional Non-Uniform Rational B-splines known as NURBS. NURBS behave in much the same way as B-splines, but each knot can have its own weight

in *ab initio* design in which case the cost has to be cut down somehow.

(see chapter 2), so that the user can affect the amount of pinch the curve makes at the control point. One-dimensional NURBS are actually *functions* mathematically speaking. Because of this property, no matter how the animator edits or modifies the one-dimensional curve, the physically meaningless state described above will never occur. Being functions, with appropriate normalisation, the system can achieve a direct intuitive mapping or correspondence between the curve output and the animated parameter value. And the last advantage of one-dimensional NURBS but not the least is its dimension; the system needs to carry only one set of computation.

The rest of the document is structured as follows. Chapter 2 is a brief overview of the mathematical model used to represent B-splines in general and NURBS in particular. It also describes some of the properties of NURBS. Chapter 3 goes over the fundamental geometric algorithms and shape modification tools developed for the manipulation of NURBS curves. Chapter 4 explores the issues encountered when moving from two-dimensional NURBS to one-dimensional NURBS. The focus is on possible changes in NURBS properties and also the different algorithms associated with. Chapter 5 describes the prototype implemented and the related issues. The conclusion summarizes the work and exposes some possible future investigations.

Chapter 2

Non-Uniform Rational B-Splines

2.1 Introduction

Mathematically, a curve can be represented either in a parametric or a non-parametric form. A non-parametric representation is either implicit or explicit. Example of an explicit non-parametric form is the equation of a straight line $y = mx+b$. Example of an implicit non-parametric form is the equation of a circle $f(x, y) = x^2 + y^2 - r = 0$. Both explicit and implicit non-parametric curve representations are axis dependent. Consequently, the choice of a coordinate system affects their usage. In the parametric form, each of the axis variables is a function of an independent parameter: a plane (two-dimensional) curve is expressed as:

$$\mathbf{C}(u) = (x(u), y(u)) \qquad a \leq u \leq b$$

where the independent parameter u varies over the interval $[a, b]$ which is often normalized to $[0, 1]$ (from now on, we will be using the interval $[0, 1]$). $x(u)$ and $y(u)$ are the univariate functions describing the x and y coordinates of any point on the curve. As u varies from a to b, the functions sweep out the curve. The parametric curve is usually a polynomial or rational polynomial. The polynomial representation can be rearranged into a form that is more intuitive to use in geometric modelling. An example of this representation is the *Bézier curve*. A Bézier curve of degree n can be defined in terms of a set of geometric coefficients called *control points* $\{\mathbf{P}_i\}$ and a set of *basis functions* $\{B_{i,n}(u)\}$; it is given by:

$$\mathbf{C}(u) = \sum_{i=0}^{n} B_{i,n}(u)\mathbf{P}_i \qquad 0 \leq u \leq 1 \tag{2.1}$$

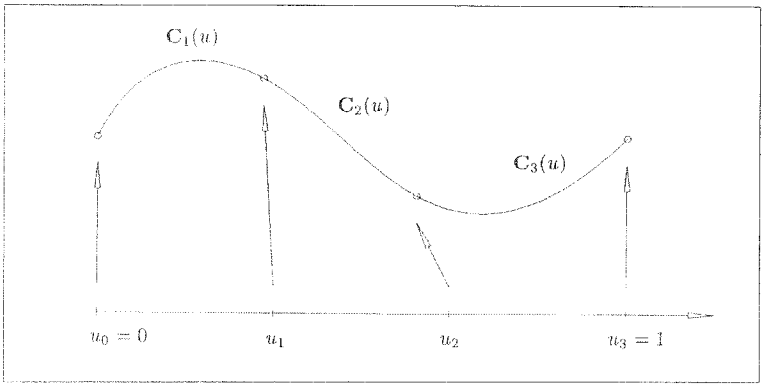


Figure 2.1: A piecewise cubic polynomial curve with three segments. (from [10])

The blending functions $\{B_{i,n}(u)\}$ are the nth-degree Bernstein polynomials given by

$$B_{i,n}(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i} \tag{2.2}$$

This new representation is more flexible and much more useful in computer graphics. Even more, by changing the basis functions, one can obtain a different class of parametric curves. In effect, Bézier curves are just a special case of a more general class of parametric curves, called *B-splines*, which uses a different set of basis functions. The next section describes B-splines in more details.

2.2 B-splines

B-splines are a generalization of Bézier curves. A B-spline curve is a piecewise polynomial curve constructed from several curve segments joined together at some *breakpoints* with some level of continuity between them. Such a curve is depicted in Figure 2.1. The curve, defined on $u \in [0, 1]$, is composed of three curve segments; the segments are joined together at the breakpoints $u_0 = 0 \leq u_1 \leq u_2 \leq u_3 = 1$. B-splines use as basis a set of piecewise polynomial functions known as the *B-spline basis functions*. A *pth-degree B-spline curve* is defined by

$$\mathbf{C}(u) = \sum_{i=0}^{n} N_{i,p}(u) \mathbf{P}_i \qquad a \leq u \leq b \tag{2.3}$$

where $\{\mathbf{P}_i\}$ are the control points, and the $\{N_{i,p}(u)\}$ are the pth-degree B-spline basis functions. There are different methods to define the B-spline basis functions.

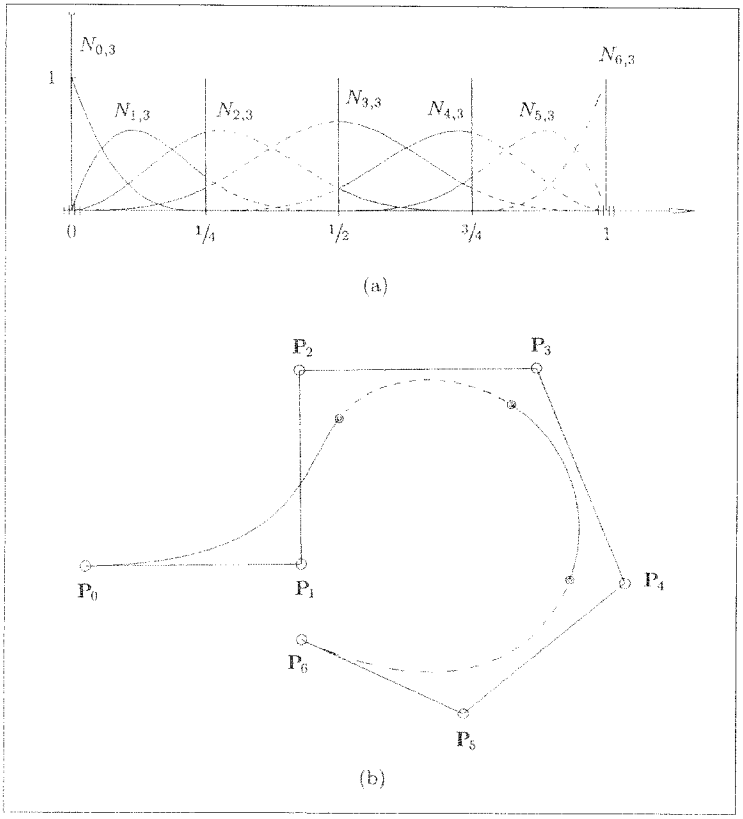


Figure 2.2: (a) Cubic basis functions; (b) A cubic curve using the basis functions of (a). (from [10])

A recurrence definition, known as the Cox-deBoor recursion formula, is widely used because it is well suited for a computer implementation. The ith B-spline basis function of degree p is defined as follows:

$$N_{i,0}(u) = \begin{cases} 0 & \text{if } u_i \leq u < u_{i+1} \\ 1 & \text{otherwise} \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{2.4}$$

Figure 2.2 shows an example of cubic (degree $p = 3$) basis functions and a B-spline curve using them. The u_i in Eq. (2.4) are called *knots*; they are elements of a nondecreasing sequence of real numbers, i.e. $u_i \leq u_{i+1}$, called a *knot vector*. There are *uniform* and *non-uniform* knot vectors. A uniform knot vector has all its interior knot values equally spaced. But the commonest type is the non-uniform knot vector

defined by

$$\mathbf{U} = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1}\} \tag{2.5}$$

The multiplicity of knots at the ends of the vector is equal to the order $p + 1$ of the basis function; this multiplicity is illustrated in Figure 2.2a with small vertical bars at $u = 0$ and $u = 1$ along the horizontal axis. The half-open interval $[u_i, u_{i+1})$ is called the *ith knot span*. As we can see from Eqs. (2.4) and (2.5), the choice of a knot vector affects the B-spline basis functions and consequently the resulting B-spline curve. For example, a knot vector with no interior knot values yields a Bézier curve.

2.3 Non-Uniform Rational B-Splines

The way B-splines are defined so far (Eq. 2.3) does not allow for the representation of conic curves (circles, ellipses, cylinders, cones, spheres, etc). However, they can be represented using *rational polynomial functions* which are defined as the ratio of two polynomials of the form

$$x(u) = \frac{X(u)}{W(u)} \qquad\qquad y(u) = \frac{Y(u)}{W(u)} \tag{2.6}$$

The corresponding curves are the *rational B-splines*. A rational B-spline curve can be thought of [3] as the projection of a non-rational B-spline curve defined in 4D homogeneous coordinate space back into 3D space achieved by dividing the coordinates by the homogeneous coordinate. A general class of rational B-splines are the NonUniform Rational B-Spline curves known as NURBS. A pth-degree NURBS curve is defined by

$$\mathbf{C}(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i \mathbf{P}_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i} \qquad a \leq u \leq b \tag{2.7}$$

where $\{\mathbf{P}_i\}$ are the control points, the $\{w_i\}$ are the weights, and the $\{N_{i,p}(u)\}$ are the pth-degree B-spline basis functions defined on the non-periodic knot vector defined

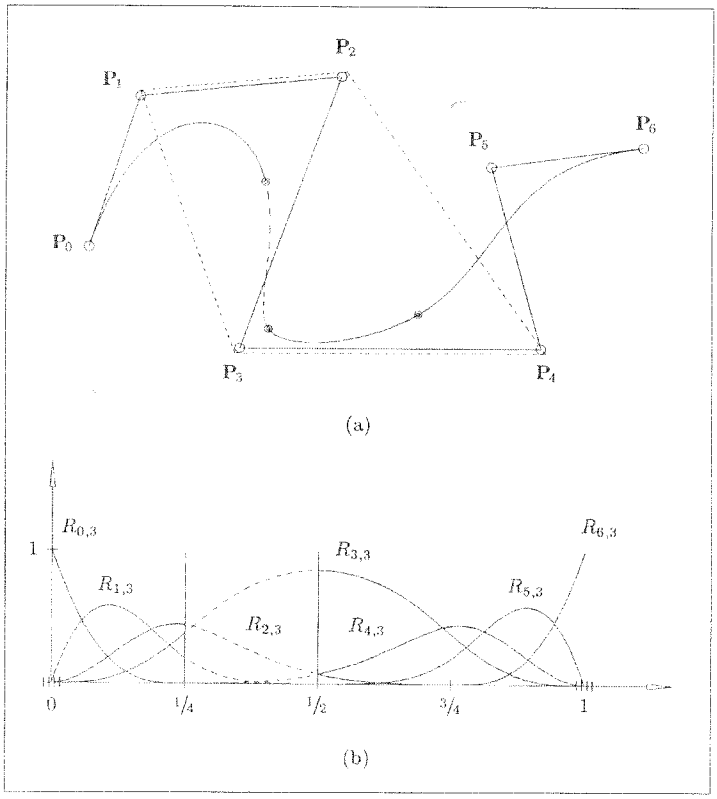


Figure 2.3: (a) A cubic NURBS curve (b) associated basis functions. (from [10])

in Eq. (2.5). By setting

$$R_{i,p}(u) = \frac{N_{i,p}(u)w_i}{\sum_{j=0}^{n} N_{j,p}(u)w_j} \tag{2.8}$$

we can obtain the familiar form

$$\mathbf{C}(u) = \sum_{i=0}^{n} R_{i,p}(u)\mathbf{P}_i \tag{2.9}$$

The $\{R_{i,p}(u)\}$ are the rational basis functions; they are piecewise rational functions on $u \in [0, 1]$. Figure 2.3 shows an example of a cubic NURBS curve and its associated rational basis functions.

2.4 Properties

As we can see from Eqs. (2.7) to (2.9), NURBS and their rational basis functions $\{R_{i,p}(u)\}$ are a generalisation of non-rational B-spline curves and their basis functions $\{N_{i,p}(u)\}$ which are in turn a generalization of Bézier curves and their Bernstein polynomials. Consequently, they inherit nearly all the analytical and geometric characteristics of the non-rational basis functions and curves and also those of Bézier curves. Next is a enumeration of the most important properties [10].

P.1 *Nonnegativity*: each basis function is positive or zero for all parameter values, i.e. $R_{i,p}(u) \geq 0$ for all i, p and $u \in [0,1]$. None of the basis functions $R_{i,p}(u)$ in the example of Figure 2.3b is negative over the interval $u \in [0,1]$. This is also true for the $N_{i,p}(u)$ in the example of Figure 2.2a.

P.2 *Partition of unity*: For any knot span $[u_i, u_{i+1})$, $\sum_{j=i-p}^{i} R_{j,p}(u) = 1$ for all the parameter values $u \in [u_i, u_{i+1})$.

P.3 For $p > 0$, each $R_{i,p}(u)$ has precisely one maximum on the interval $u \in [0,1]$.

P.4 *Local support*: $R_{i,p}(u) = 0$ if u is outside the open interval $[u_i, u_{i+p+1})$. In any given knot span $[u_i, u_{i+1})$, at most $p+1$ of the $R_{i,p}(u)$ are nonzero, namely the functions $R_{i-p,p}(u), \ldots, R_{i,p}(u)$.

P.5 All derivatives of $R_{i,p}(u)$ exist in the interior of a knot span, where it is a rational function with nonzero denominator. At a knot, $R_{i,p}(u)$ is $p-k$ times continuously differentiable, where k is the multiplicity of the knot.

P.6 If $w_i = 1$ for all i, then $R_{i,p}(u) = N_{i,p}(u)$. The non-rational $N_{i,p}(u)$ are special cases of the rational $R_{i,p}(u)$.

P.7 Since $\mathbf{C}(u)$ is just a linear combination of $R_{i,p}(u)$ and because of property P.5, $\mathbf{C}(u)$ is infinitely differentiable on the interior of knot spans and is $p-k$ times differentiable at a knot of multiplicity k.

P.8 $\mathbf{C}(u)$ is a piecewise polynomial curve since $R_{i,p}(u)$ are piecewise polynomials. The number of control points, $(n+1)$, the number of knots, $(m+1)$, and the degree p of the basis functions are related by

$$m = n + p + 1 \tag{2.10}$$

i.e. `number of knots = number of control points + order`

P.9 *End point continuity*: $\mathbf{C}(0) = \mathbf{P}_0$ and $\mathbf{C}(1) = \mathbf{P}_n$; in other words, the curve $\mathbf{C}(u)$ interpolates the first and last control points; this comes from the fact that $R_{0,p}(0) = R_{n,p}(1) = 1$. In the example of Figure 2.3a, the curve interpolates the points P_0 and P_6.

P.10 *Affine invariance*: any affine transformation (translation, rotation, scaling) can be applied to the curve $\mathbf{C}(u)$ by applying it to the control points $\mathbf{P}_i$; i.e., the curve is transformed by transforming the control points; NURBS curves are also invariant under perspective projections which is not the case of non-rational B-splines.

P.11 *Strong convex hull property*: the curve $\mathbf{C}(u)$ is contained in the convex hull of its *control polygon* which is the polygon defined by the control points $\mathbf{P}_i$. Precisely, if $u \in [u_i, u_{i+1})$ and $p \leq i \leq m - p - 1$ then the portion of the curve $\mathbf{C}(u)$ that is on the interval $u \in [u_i, u_{i+1})$ is in the convex hull of the control points $\mathbf{P}_{i-p}, \ldots, \mathbf{P}_i$. In Figure 2.3a, the *dashed* segment of the curve lies inside the convex hull of the control points $\{\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \mathbf{P}_4\}$ illustrated by a dashed polygon.

P.12 *Variation diminishing property*: no plane has more intersections with the curve than with the control polygon (replace 'plane' with 'line' for two dimensional curves).

P.13 A NURBS curve with no interior knots, i.e. the knot vector is of the form

$$\mathbf{U} = \{\underbrace{0, \ldots, 0}_{p+1}, \underbrace{1, \ldots, 1}_{p+1}\}$$

is a rational Bézier curve. NURBS curves contain non-rational B-spline and rational and non-rational Bézier curves as special cases.

P.14 *Local approximation*: moving the control point $\mathbf{P}_i$, or changing the weight w_i affects only the portion of the curve $\mathbf{C}(u)$ that is on the interval $u \in [u_i, u_{i+p+1})$; this comes from the fact that $R_{i,p}(u)$ is zero outside its supporting knot spans $u \ni [u_i, u_{i+p+1})$. Increasing the weight w_i has the effect of pulling the curve toward the corresponding control point $\mathbf{P}_i$, and decreasing the weight has the

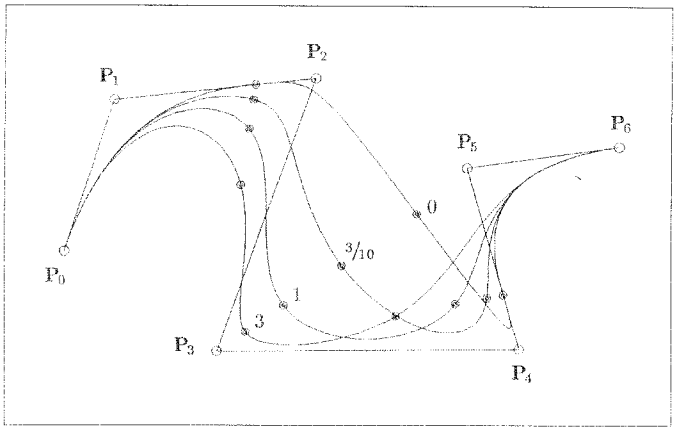


Figure 2.4: Rational cubic B-spline curves with w_3 varying. (from [10])

opposite effect. Figure 2.4 illustrates this behavior with the weight of control point $\mathbf{P}_3$.

P.15 The control polygon formed by the control points $\mathbf{P}_i$ represents a piecewise linear approximation to the curve $\mathbf{C}(u)$. In general, the curve gets closer to its control polygon when the degree gets lower.

The combination of all the enumerated properties make NURBS curves a powerful tool for representing free-form curves and surfaces and common analytic shapes. Many of these properties are very important when developing shape modification algorithms. Property P.14 in particular is very important for interactive shape design. The definitions and properties described above apply to two and three-dimensional NURBS without loss of generality. However, as we shall see in chapter 4, some of the properties may not hold for one-dimensional NURBS.

Chapter 3

Geometric Algorithms for NURBS

3.1 Introduction

NURBS have a rich geometric domain, in that they provide precise mathematical form capable of representing the common analytical shapes and free-form curves and surfaces. They are also unconditionally stable to floating points computation [9]. All of this comes mainly from the properties and the flexibility of rational B-spline basis functions. This flexibility allows for different types of control *handles* used to modify the shape of NURBS curves. As described in previous chapter, a NURBS curve is defined by its control points $\mathbf{P}_i$, weights w_i and knot vector U. Consequently, control can be achieved by

- changing the knot vector: inserting or removing a knot value, refining the knot vector, using multiple knot values in the knot vector;
- changing (elevating or reducing) the degree p of the basis functions;
- changing the number of control points and/or control polygon: inserting or removing control points, using multiple control points.

Several geometric algorithms and shape modification tools have been devised to achieve the types of control we have just described. The fundamental geometric algorithms are knot insertion and knot removal, knot refinement, degree elevation and degree reduction. Some of the shape modification tools are based on control point repositioning, weight modification, and shape operators such as *warping*. In the following sections we will describe some of the fundamental algorithms and in the next chapter we will present some of the shape modification tools.

To illustrate some of the fundamental geometric algorithms, we shall use an other method of representing NURBS curves.

$$\mathbf{C}^w(u) = \sum_{i=0}^{n} N_{i,p}(u)\mathbf{P}_i^w \tag{3.1}$$

The curve $\mathbf{C}^w(u)$ in Eq. (3.1) represents a NURBS curve defined on $U = \{u_0, \ldots, u_m\}$, where $\{\mathbf{P}_i^w\}$ are the weighted control points, $\mathbf{P}_i^w = (w_i x_i, w_i y_i, w_i z_i, w_i)$. The superscript w in $\mathbf{C}^w(u)$ and $\mathbf{P}_i^w$ indicates that we are using homogeneous coordinates in four-dimensional space.

3.2 Knot Insertion

When a new knot value, let it be $u \in [u_k, u_{k+1})$, is to be inserted into U, we obtain a new knot vector $\bar{U}$ with a new size $m+2$. We also obtain a new set of basis functions, $\{N_{i,p}(u)\}$, defined on $\bar{U}$. To maintain the property `P.8`, the number of control points is to increase accordingly. The geometry of the curve does not change but it has a new representation of the form

$$\mathbf{C}^w(u) = \sum_{i=0}^{n+1} N_{i,p}(u)\mathbf{Q}_i^w \tag{3.2}$$

where $\{\mathbf{Q}_i^w\}$ are the new control points. The *knot insertion* algorithm concern is to determine these new control points. Because the curve geometry is not modified, the problem comes down to solving the equality between Eqs. (3.1) and (3.2). By taking advantage of the *local support* property (`P.4`), we can obtain the formula for computing the new control points, $\mathbf{Q}_i^w$

$$\mathbf{Q}_i^w = \alpha_i \mathbf{P}_i^w + (1 - \alpha_i)\mathbf{P}_{i-1}^w \tag{3.3}$$

$$\alpha_i = \begin{cases} 1 & i \leq k - p \\ \frac{u - u_i}{u_{i+p} - u_i} & k - p + 1 \leq i \leq k \\ 0 & i \geq k + 1 \end{cases}$$

Figure 3.1a shows an example of a control polygon of a curve modified by the insertion of a single knot; Figure 3.1b shows the effect on the basis functions.

Knot insertion is one of the most important of all B-splines algorithms. Among its

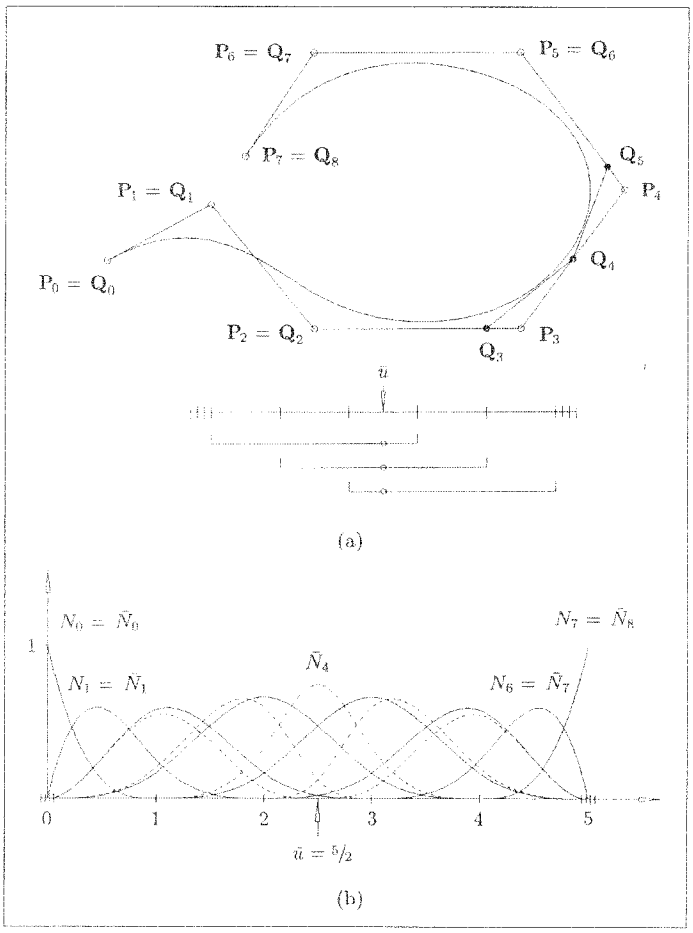


Figure 3.1: Knot insertion into a cubic curve. (a) Control polygon before insertion, $(\mathbf{P}_i)$, and after insertion, $(\mathbf{Q}_i)$; (b) basis functions before insertion (solid), and after insertion (dashed). (from [10])

numerous applications is *curve splitting* (or subdivision). One can split a curve into segments at a desired parameter location u_s by simply inserting the value u_s into the knot vector a number of times equal to the degree p of the curve. The same process can be used to evaluate a point on a curve. Inserting a knot u_s degree times forces the curve to interpolate the new control point generated by the insertion. Another process known as *inverse knot insertion* allows to obtain a new control point on the control polygon. It is called inverse knot insertion because, instead of providing a knot value, the user specify the position of the new control to be obtained, and the process consist in finding the knot value when inserted generate the specified control point.

3.3 Knot Refinement

It is often necessary to insert many (distinct or not) knots at once; the original knots and the newly inserted knots are merged to form a refined knot vector. The process is called *knot refinement*. The necessity is mainly due to the computation cost. Instead of using the knot insertion algorithm to insert the different knots one at the time, there are more efficient algorithms, like the one developed by [2], that allow a simultaneous insertion.

An important application of knot refinement is the decomposition of a NURBS into its constituent (Bézier) polynomial pieces: using the knot refinement algorithm, the multiplicity of the interior knots is increased up to degree times, the breakpoints became interpolated control points; consequently, each segment stands as a Bézier segment. A user can also obtain a polygonal approximation of a curve. In the example of Figure 3.2a and 3.2b, the knot vector was refined by recursively inserting new knots a the midpoints of the spans. This brings the control polygon closer to the curve.

3.4 Knot Removal

Knot removal is the opposite process of knot insertion. In some instances, to achieve a desired curve shape, multiple knot insertions are needed to increase the number of control points. Once the operation is done, the curve representation has to be brought down to the minimum by removing some knots; however the geometry of

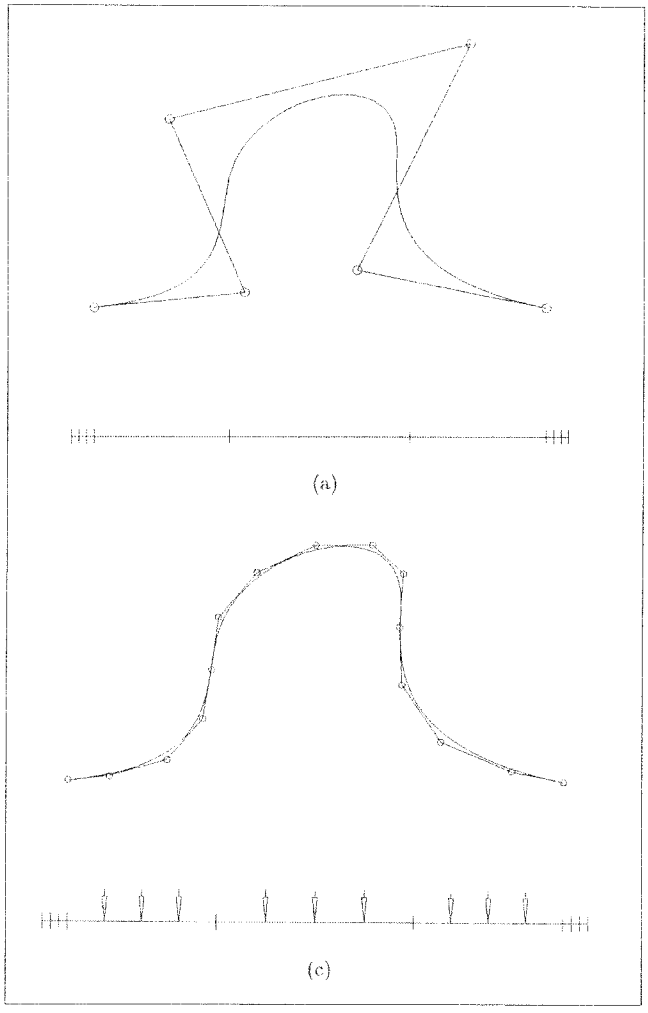


Figure 3.2: Curve refinement. (a) Cubic refinement before refinement; (d) after two midpoint knot refinements. (from [10])

the curve should stay the same. In this context, a knot removal algorithm has to determine if a knot is removable and how many times, and the algorithm has to determine the new control points. Reference [12] has a detailed description of knot removal algorithms. If we consider the curve in Eq. (3.1), and an interior knot $u = u_r$ with multiplicity s, u_r is removable t times if the curve has a representation of the form

$$\mathbf{C}^w(u) = \sum_{i=0}^{n-t} \hat{N}_{i,p}(u)\mathbf{Q}_i^w \tag{3.4}$$

Using the notation $\mathbf{P}_i^j$ which indicates a new control point with index i obtained after j removals, the equations for computing the new control points obtained after removing the knot $u = u_r$ t times can be formulated as follows

$$\begin{aligned}
\mathbf{P}_i^t &= \frac{\mathbf{P}_i^{t-1} - (1-\alpha_i)\mathbf{P}_{i-1}^t}{\alpha_i} \qquad r-p-t+1 \le i \le \frac{1}{2}(2r-p-s-t) \\
\mathbf{P}_j^t &= \frac{\mathbf{P}_j^{t-1} - \alpha_j\mathbf{P}_{j+1}^t}{1-\alpha_j} \qquad \frac{1}{2}(2r-p-s+t+1) \le j \le r-s+t-1
\end{aligned} \tag{3.5}$$

$$\text{with} \qquad \alpha_i = \frac{u-u_i}{u_{i+p+t}-u_i} \qquad \alpha_j = \frac{u-u_{j-t+1}}{u_{i+p+1}-u_{j-t+1}} \tag{3.6}$$

When removing knots, the number of control points decreases to maintain property (P.8). This leads to a less accurate representation of the curve. To control the knot removal process, a value, TOL, is used to specify how much is the new curve allowed to deviate from the original one. In the case of NURBS curves, the deviation has to be less than

$$\frac{\mathtt{TOL}(1+|\mathbf{P}|_{max})}{w_{min}}$$

where w_{min} is the minimum weight, $|\mathbf{P}|_{max}$ is the distance of any point on the original curve from the origin. The bound on deviation, d is obtained using the *convex hull* property (P.11) and TOL is defined as

$$\mathtt{TOL} = \frac{dw_{min}}{1+|\mathbf{P}|_{max}} \tag{3.7}$$

Figure 3.3a and Figure 3.3d show an example of a cubic curve before knot removal and after knot removal. One can see in Figure 3.3d the slight deviation of the new

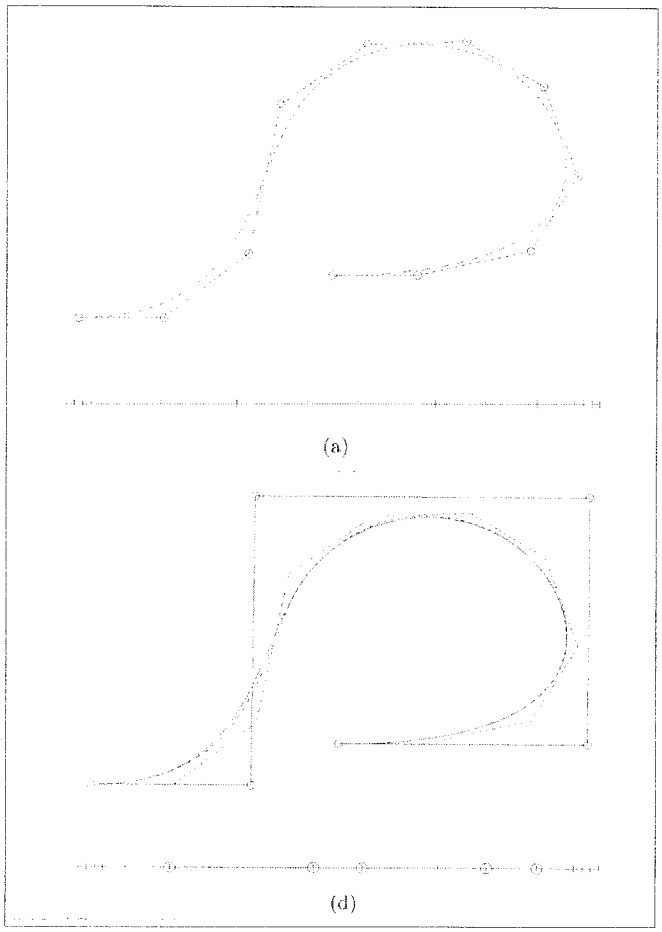


Figure 3.3: (a) the original cubic curve; (d) knots removed with tolerance TOL = 1.2. (from [10])

curve (solid) from the original one (dashed).

3.5 Degree Elevation

There are situations where a user wants to combine several NURBS curves into a single NURBS curve. To join the endpoints, the curves have to be of the same degree to achieve smoothness and continuity at the joinpoints. Degree elevation algorithms are used to bring the curves to a common degree. If the curve in Eq. (3.1), denoted $\mathbf{C}_p^w(u)$, has its degree elevated to $p+1$, then, there must exist a new representation

of the form

$$\mathbf{C}_p^w(u) = \mathbf{C}_{p+1}^w(u) = \sum_{i=0}^{\hat{n}} N_{i,p+1}(u)\mathbf{Q}_i^w \tag{3.8}$$

There are three unknowns in Eq. (3.8), $\hat{n}$, $\hat{U}$ and $\{\mathbf{Q}_i^w\}$. The formulas for these three entities can be found in [10]. In the same reference, the author presents a simple degree elevation algorithm:

```
decompose the curve into its Bezier segments
repeat
  extract the ith Bezier segment from the curve
  degree elevate the ith Bezier segment
  remove unnecessary knots between (i - 1)th and ith segments
until done
```

The first step can be achieved using the knot refinement algorithm. The problem is reduced to elevating the degree of Bézier curves in which case there is a well defined formula for computing the new control points

$$\begin{aligned} &\mathbf{Q}_i^w = (1-\alpha_i)\mathbf{P}_i^w + \alpha_i \mathbf{P}_{i-1}^w \\ &\text{where} \qquad \alpha_i = \frac{i}{p+1} \qquad i = 0, \ldots, p+1 \end{aligned} \tag{3.9}$$

Figure 3.4 shows an example of a cubic curve ($p = 3$) who's degree is elevated by four ($p = 7$).

3.6 Degree Reduction

Degree reduction is the inverse process of degree elevation. A curve can always be degree elevated, but it may not be degree reducible. The curve in Eq. (3.1) is degree reducible if there exists a representation of the form

$$\mathbf{C}_p^w(u) = \mathbf{C}_{p-1}^w(u) = \sum_{i=0}^{\hat{n}} N_{i,p-1}(u)\mathbf{Q}_i^w \tag{3.10}$$

The degree elevation algorithm concern is to determine the three unknowns, $\hat{n}$, $\hat{U}$ and $\{\mathbf{Q}_i^w\}$. One can use the same strategy as for degree elevation except for the key

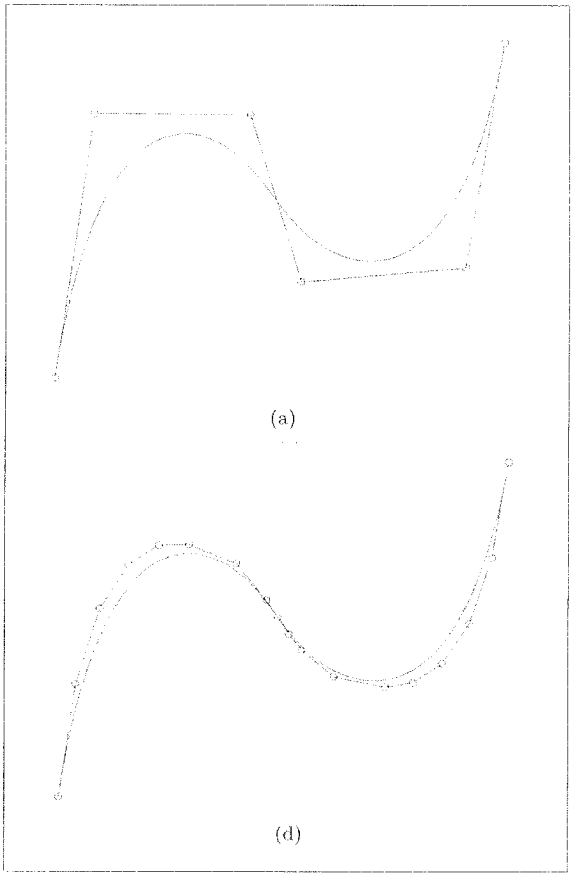


Figure 3.4: Curve degree elevation example. (a) the original cubic curve; (d) the degree is elevated by four. (from [10])

part where the Bézier segments are degree reduced instead:

```
  decompose the curve into its Bezier segments
  repeat
    extract the ith Bezier segment from the B-spline curve
    degree reduce the ith Bezier segment
    remove the unnecessary knots between the (i - 1)th and the ith
segments
  until done
```

Similar to knot removal, a value `TOL` is used to define a threshold. The deviation of the curve has to be below the threshold for the curve to be declared degree reducible.

Chapter 4

One-Dimensional NURBS

4.1 Introduction

We recall that the motivation of this work is to investigate the use of one-dimensional NURBS curves for animation control. Readers may be familiar with the terms *spatial curves* which refer to curves represented in three dimensions (space), *plane curves* which refer to curves represented in two dimensions (plan). In this context, what is it that we are referring to by a one-dimensional NURBS curve?

Let's consider the case of a two-dimensional NURBS curve represented in Eq. (2.9). The curve correspond to a set of points $\{\mathbf{C}(u)\}$ with each point having two components $\mathbf{x}(u)$ and $\mathbf{y}(u)$. Each of the control points $\mathbf{P}_i 0$ has two coordinates, x_i and y_i.[1] We can expand Eq. (2.9) to show the two dimensions as follows

$$\mathbf{C}(u) = \begin{cases} \mathbf{x}(u) = \sum_{i=0}^{n} R_{i,p}(u)\, x_i \\ \mathbf{y}(u) = \sum_{i=0}^{n} R_{i,p}(u)\, y_i \end{cases} \qquad a \leq u \leq b \tag{4.1}$$

It transpires from Eq. (4.1) that the dimensionality of the curve is directly related to the dimensionality of the geometric coefficients $\{\mathbf{P}_i\}$. Having said that, if we decide to drop one of the coordinates from the control points, subsequently, they can no longer be considered as geometric coefficients; they become *scalar coefficients* or *control values*, the term we will be using from now on. The curve in Eq. (4.1) will then loose one of its components and the new representation of the resulting curve

[1] If we are using Eq.(3.1), the control points will have the coordinates $w_i x_i$, $w_i y_i$ and w_i instead.

can be formulated by

$$\mathcal{C}(u) = \sum_{i=0}^{n} R_{i,p}(u)\,\mathcal{P}_i \qquad a \leq u \leq b \tag{4.2}$$

where $\{\mathcal{P}_i\}$ are the new control values and $\{R_{i,p}(u)\}$ are the rational basis functions. Note that $\mathcal{C}(u)$ is now a scalar value. The entity represented in Eq. (4.2) is what we loosely call a *one-dimensional NURBS curve*. The graphical representation of the 1D NURBS curve correspond to a plot, on a display device, of the set of curve points defined by $\{(u, \mathcal{C}(u))\}$ with $u \in [a, b]$. When dealing with a mathematical problem, we will refer by one-dimensional NURBS curve to the model in Eq. (4.2). When presenting a user interface related issue, we will refer by one-dimensional NURBS curve to the graphical representation of the model in Eq. (4.2).

The functions $\{R_{i,p}(u)\}$ are not affected by the change in the dimensionality of the curve: they are function of the independent parameter u and the definition in Eq. (2.8) is rigorously the same. Equation (4.2) is not exactly a parametric curve representation. A more appropriate denomination would be *a mathematical function*: $\mathcal{C}(u)$ is a linear combination of the rational basis *functions* $R_{i,p}(u)$.[2] Being a function, its graphical representation will not exhibit any possible looping behavior as in Figure 1.2: for each input value u, there will be only one $\mathcal{C}(\bar{u})$ output value. This is precisely the property we were seeking when we decided to reduce the dimension of NURBS curves (section 1.4) to one dimension.

Another consequence of this property is the mapping between the one-dimensional NURBS curve input/output and the motion parameter input/output. By using functions to normalize the inputs and outputs, it is easy to come up with a simple mapping between the inputs/outputs of the 1D curve and the motion parameter. If we consider u as the input for the one-dimensional curve and $\mathcal{C}(u)$ as the curve output; and if we denote by t the input for the motion graph and by $\mathcal{M}(t)$ the motion graph output, the mapping can be formulated by

$$u = f(t)$$
$$\mathcal{M}(t) = g(\ \mathcal{C}(u)\ )$$

The function $f(t)$ normalizes the values of t to the range $[0, 1]$ of u. The function

[2] A linear combination of functions yields a function.

$g(C)$ normalizes the values of $\mathcal{C}(u)$ to the range of values expected for $\mathcal{M}$. This approach reduces greatly the computation complexity of one-dimensional curves.

4.2 Properties

The properties enumerated in section 2.4 apply to both two and three-dimensional NURBS curves without loss of generality. We shall look in this section at the evolution of these properties when considering one-dimensional NURBS curves as defined in the previous section.

The properties of NURBS curves are of two main origins. Some directly map the properties of the rational basis functions $R_{i,p}(u)$; others are related to the geometric nature of the control points. Because the basis functions, as defined in Eq. (2.8), are independent of the dimensionality of NURBS curves, the related properties are preserved when moving to one-dimensional NURBS curves. Examples of these properties are nonnegativity (P.1), partition of unity (P.2), local support (P.4) and local approximation (P.14). This is an important result since many of the fundamental algorithms take advantage of these properties.

Some of the geometric properties will not be retained in 1D curves. Because we no longer have geometric control points, a concept such as convex hull (P.11) can not be defined for 1D NURBS curves. This result has possible impacts on certain algorithms such as knot removal and degree reduction (see next section). The other geometric property that does not hold is the affine invariance (P.10). We can not apply an affine transformation to the control values: affine transformations apply only to cartesian or polar coordinates. The last example is the variation diminishing property (P.12).

It is important to mention that not all of these geometric properties, in particular affine invariance and variation diminishing properties, have a major role when NURBS curves are used for animation control. Usually, the interaction of the user with the motion parameter curve does not involve applying complex geometric transformations. It is rather limited to simple deformations (adjustments) or applying predefined geometric shapes (straight lines, ease-in ease-out curves, etc) to portions of the motion parameter curve. Consequently, loosing these properties should not be an issue for 1D NURBS curves.

4.3 Fundamental Algorithms

From a mathematical point of view, the development of a knot insertion algorithm for 1D NURBS curves is exactly the same as for 2D/3D NURBS curves. The only change we have to take into account in the mathematical development is the switch to a different set of control points. Instead of solving the equality between Eqs. (3.1) and (3.2), we solve the system

$$\sum_{i=0}^{n} R_{i,p}(u)\,\mathcal{P}_i^w = \sum_{i=0}^{n+1} R_{i,p}(u)\,\mathcal{Q}_i^w \tag{4.3}$$

where $\{\mathcal{P}_i^w\}$, $\mathcal{P}_i^w = (x_i, w_i)$, are the original control values and $\{\mathcal{Q}_i^w\}$ are the new control values to be computed. We know that the local support property (P.4) is preserved in 1D NURBS curves. This makes the solution of the above system leads to a similar set of equations for computing the new control values as for 2D/3D NURBS curves:

$$\mathcal{Q}_i^w = \alpha_i \mathcal{P}_i^w + (1-\alpha_i)\mathcal{P}_{i-1}^w \tag{4.4}$$

$$\alpha_i = \begin{cases} 1 & i \leq k-p \\ \frac{\bar{u}-u_i}{u_{i+p}-u_i} & k-p+1 \leq i \leq k \\ 0 & i \geq k+1 \end{cases}$$

Since the knot refinement algorithm is just a fast variant of the knot insertion algorithm, there is no fundamental problem in developing it for 1D NURBS curves. As a matter of fact, in our implementation (see chapter 5), we used, for knot insertion and knot refinement algorithms, implementations that are based on those developed for 2D/3D NURBS curves with no major changes.

A similar analysis applied to the development presented in section 3.5, using control values instead, can be used for the adaptation of the degree elevation algorithm in the case of 1D NURBS curves.

The algorithms inverse knot insertion, knot removal and degree reduction all have the same adaptation problem when moving to 1D NURBS curves. They all involve the control polygon of the curve.

As explained in section 3.2, the inverse knot insertion algorithm involves the positioning of a new control point on an edge of the control polygon. In 1D NURBS

curves, we don't have a control polygon as such; in other words, there is no obvious way sustained by the 1D NURBS model to illustrate a control polygon using scalar values. As a consequence, the system will not be able to figure out where, in the parameter space, to insert the new knot.

Knot removal and degree reduction algorithms need to compute a quantity `TOL` that defines the deviation of the new curve from the original. The computation is based on the convex hull of the control polygon of the curve. Using the 1D model as it is, the system will not be able to provide tools for shape modification that use these algorithms. Because the issue is mainly related to the user interface provided for 1D NURBS curves, we present in section 4.5 a possible work around for the control polygon problem.

4.4 Shape Modification Tools

To demonstrate the evolution of the shape modification tools when considering 1D NURBS curves, we shall present in each instance, the usage of the tool in the case of 2D/3D curves followed by its "possible" usage in the case of 1D curves.

Control point repositioning

From a mathematical point of view, repositioning control points correspond to modifying the values of the geometric coefficients in Eq. (2.9); this leads to a new set of curve points $\{\hat{\mathbf{C}}(u)\}$. If we consider the NURBS curve represented by $\mathbf{C}(u) = \sum_{i=0}^{n} R_{i,p}(u)\mathbf{P}_i$, and we denote the control point to be repositioned by $\mathbf{P}_k$, its new location by $\hat{\mathbf{P}}_k$ and the translation vector by $\mathbf{V} = \hat{\mathbf{P}}_k - \mathbf{P}_k$. Then, the new curve can be expressed by

$$\begin{aligned}\hat{\mathbf{C}}(u) &= R_{0,p}(u)\mathbf{P}_0 + \cdots + R_{k,p}(u)(\mathbf{P}_k + \mathbf{V}) + \cdots + R_{n,p}(u)\mathbf{P}_n \\ &= \mathbf{C}(u) + R_{k,p}(u)\mathbf{V} \end{aligned} \tag{4.5}$$

Eq. (4.5) shows that repositioning a control point $\mathbf{P}_k$ correspond to translating all the curve points, $\mathbf{C}(u)$, for which $u \in [u_k, u_{k+p+1})$ ($R_{k,p}(u)$ is nonzero in the interval $u \in [u_k, u_{k+p+1})$). In the case of 1D NURBS curves, a similar process correspond to modifying the control values $\mathcal{P}_i$ in Eq. (4.2). If we consider the 1D curve represented by $\mathcal{C}(u) = \sum_{i=0}^{n} R_{i,p}(u)\mathcal{P}_i$, and denote the control value to be modified by $\mathcal{P}_k$, its

new value by $\hat{\mathcal{P}}_k$ and the delta value by $\mathcal{V} = \hat{\mathcal{P}}_k - \mathcal{P}_k$. Then, the new curve can be expressed by

$$\begin{aligned}\hat{\mathcal{C}}(u) &= R_{0,p}(u)\mathcal{P}_0 + \cdots + R_{k,p}(u)(\mathcal{P}_k + \mathcal{V}) + \cdots + R_{n,p}(u)\mathcal{P}_n \\ &= \mathcal{C}(u) + R_{k,p}(u)\mathcal{V} \end{aligned} \tag{4.6}$$

Eq. (4.6) shows that it is mathematically possible to achieve a similar type of control over a 1D NURBS curve through *control value modification* as it is possible with control point repositioning over 2D/3D curves.

Because of the dimension of the curves, there is a slight difference in the interpretation of the modification caused by control value modification. In Eq.(4.5), the vector $\mathbf{V}$ gives the direction (in plane or space) of the modification and its module $|\mathbf{V}|$ gives the amplitude of the modification. In Eq. (4.6), it is the sign of $\mathcal{V}$ that gives the direction of modification[3] and its absolute value $|\mathcal{V}|$ the amplitude of the modification.

Case 1. An easy way for a user to interactively modify a curve is by repositioning the control points. A user picks a control point on the control polygon and displaces it to achieve the desired shape; all the graphics system has to do is update the control point location and redraw the curve. However, this is only true for 2D/3D curves. Figure 4.1 shows an example of a control point repositioning: moving control point $\mathbf{P}_4$ toward $\hat{\mathbf{P}}_4$ produces a new curve (dashed line).

In the case of 1D curves, since we only have control values $\mathcal{P}_i$, it is difficult, from a user interface point of view to visualize, on the display device, these control values in a way that can mimic the intuitive geometric meaning of control points and their control polygon.

Case 2. There is also the situation where the user picks a point on the control polygon that does not correspond to a control point; in other words, the point lies on an edge defined by two control points. When using a 2D/3D curve, the system can turn this point into a control point using the inverse knot insertion algorithm (3.2); we end up in the situation described above.

[3] In the graphical representation of Eq. (4.6), the direction of translation is vertical: the modified portion of the curve moves upward or downward which corresponds to increasing or decreasing the associated control value.

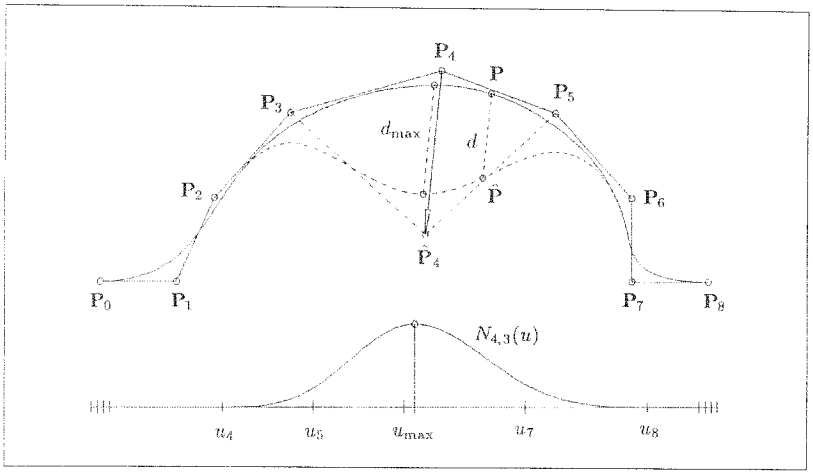


Figure 4.1: Repositioning control point $\mathbf{P}_4$. (from [10])

When using a 1D curve, since we don't have a control polygon as such, we can not apply the inverse knot insertion algorithm because it involves positioning on the control polygon. Given these facts, it is hard for the graphics system to provide this curve modification approach using the 1D NURBS curve model as it is. Since in computer graphics, user interface issues are as important as the geometric models, the difficulty stated earlier can be an obstacle for using 1D NURBS curves in an interactive environment. This problem is of the same nature as the one described above.

Case 3. Another possible scenario is where the user wants to modify the curve by selecting a point on the curve itself. In the case of 2D/3D curves, several steps take place to achieve the desired shape modification. First, the user picks a point $\mathbf{P}$ on the curve. Second, the system computes the parameter value u associated with the selected point ($\mathbf{P} = \mathbf{C}(\bar{u})$). Third, the user picks a point $\hat{\mathbf{P}}$ to which $\mathbf{P}$ is to be moved. Four, the system has to find an existing control point $\mathbf{P}_k$ to reposition in order to achieve the desired translation of $\mathbf{P}$. The common approach used for choosing $\mathbf{P}_k$ is based on the values of the $(p+1)$ basis functions $R_{i,p}(u)$ that are nonzero in the interval containing u. A control point $\mathbf{P}_k$ is selected such that its corresponding basis function $R_{k,p}(u)$ has its maximum closest to u (Eq. (4.5)). In the example of Figure 4.2, a point is picked on the curve, the control point $\mathbf{P}_3$ has the maximum of its basis function closest, hence it is repositioned to produce the new curve.

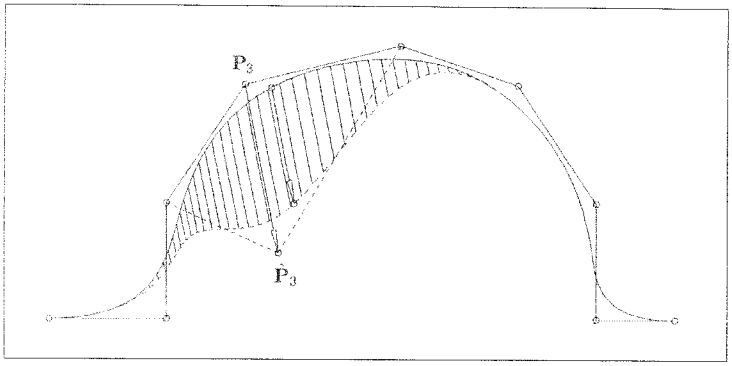


Figure 4.2: Moving curve point $\mathbf{P}$ by repositioning control point $\mathbf{P}_3$ with the closest basis function maximum. (from [10])

When considering 1D NURBS curves, the process required to achieve the desired shape modification is slightly simpler. First, the user picks a point P on the curve. The system does not need to compute the parameter value u: $\bar{u}$ maps directly with one of P's coordinates $P = (\bar{u}, \mathcal{C}(\bar{u}))$. Second, the user picks a point $\hat{P}$ where P is to be moved. The only option the user has is along a vertical translation vector (moving up or down). This is a restriction which is intrinsic to the 1D NURBS model; it helps avoid the looping behavior (Figure 1.2). Third, the system has to find the control value to modify. The approach is similar to the one previously described for 2D/3D curves.

Weight Modification

We have seen an example of weight modification in Figure 2.4. In the same example, when w_3 decreases, the curve is pushed away from the control point $\mathbf{P}_3$; when w_3 increases, the curve is pulled toward $\mathbf{P}_3$; when w_3 tends to infinity ($w_3 \to \infty$), the curve will ultimately interpolates the corresponding control point $\mathbf{P}_3$, and the curve will end up with a sharp corner at the same control point. This approach is frequently used to obtain straight edges and sharp corners.

Now we look at the effect of modifying the weight of a control value on a 1D NURBS curve. Let's consider a 1D NURBS curve represented in Eq. (4.2), a fixed parameter value $u \in [u_k, u_{k+p+1})$ and the weight w_k of a control value $\mathcal{P}_k$; when expanding Eq. (4.2), we obtain:

$$C(u) = R_{0,p}(u)\mathcal{P}_0 + \cdots + R_{k,p}(u)\mathcal{P}_k + \cdots + R_{n,p}(u)\mathcal{P}_n \tag{4.7}$$

If w_k is allowed to vary ($0 \leq w_k < \infty$), then the rational basis function $R_{k,p}(u)$ associated with the control value $\mathcal{P}_k$ becomes a function of w_k and can be expressed by

$$R_{k,p}(u) = \frac{N_{k,p}(u)w_k}{\sum_{j=0}^{n} N_{j,p}(u)w_j} = \frac{N_{k,p}(u)w_k}{\sum_{j\neq k} N_{j,p}(u)w_j + N_{k,p}(u)w_k} \tag{4.8}$$

If $w_k = 0$, $R_{k,p}(u) = 0$; it follows that $\mathcal{C}(u, w_k = 0) = \sum_{i\neq k} R_{i,p}(u)\mathcal{P}_i$. We can prove that $\mathcal{C}(u, w_k = 0) < \mathcal{C}(u, w_k \neq 0)$. Geometrically speaking, when w_k decreases, the curve value moves away from the control value $\mathcal{P}_k$. This behavior is similar to the behavior of 2D/3D curves.

If $w_k \to \infty$, the term $N_{k,p}(u)w_k$ in Eq. (4.8) becomes very large and the term $\sum_{j\neq k} N_{j,p}(u)w_j$ becomes negligible compared to $N_{k,p}(u)w_k$. We obtain the following result:

$$\overline{\lim}_{w_k\to\infty} R_{k,p}(u) = 1 \tag{4.9}$$

Using the partition of unity property (P.2), the local support property (P.4) and the result of Eq. (4.9), we can arrive to the following statement: when $w_k \to \infty$, all the rational basis functions $R_{j\neq k,p}(u)$ in the interval $[u_k, u_{k+p+1})$ are nil. The geometrical interpretation of this result is that when $w_k \to \infty$, the curve pulls toward the control value $\mathcal{P}_k$ and flattens over the interval $[u_k, u_{k+p+1})$.

Shape Operator: Warping

We saw that it is possible to modify the shape of a curve (2D/3D) by repositioning control points or by modifying the curve points directly. There is another way which takes a higher level approach. Instead of modifying the curve by acting on a single point in an incremental manner, one can apply a shape operator to a segment of the curve by repositioning the set of control points involved and in a synchronised fashion. *Warping* is one of these shape operators. It is used to deform a local segment of the curve using control point repositioning and the formula

$$\hat{\mathbf{P}}_i = \mathbf{P}_i + f(l)d\mathbf{W} \tag{4.10}$$

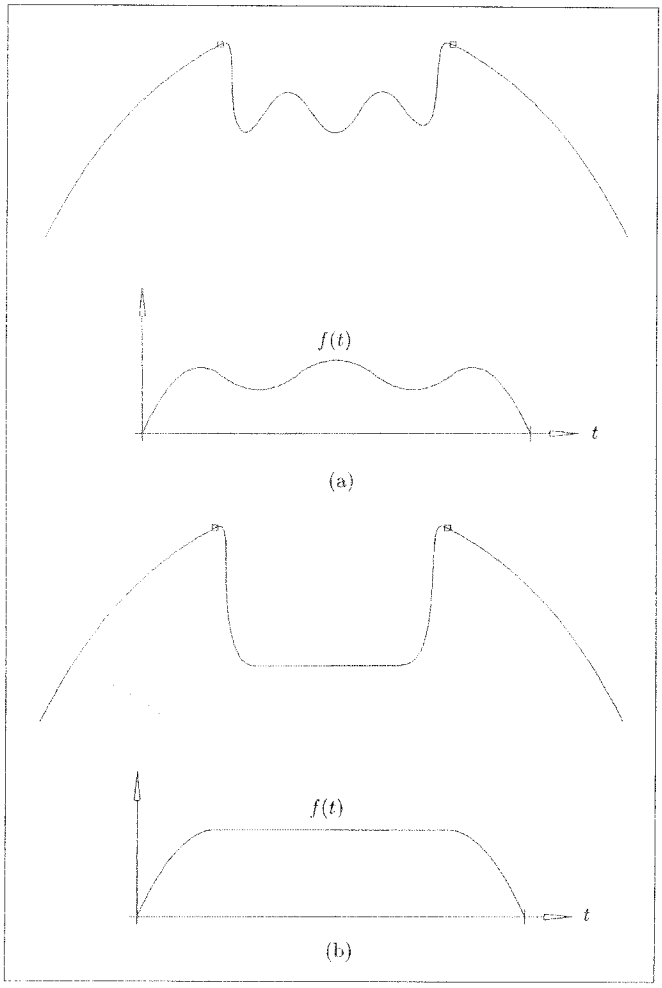


Figure 4.3: Curve warps with different shape functions. (from [10])

$f(t)$ is the function that control the warp (defines the shape of the operator); d is a constant for limiting the warp distance; $\mathbf{W}$ gives the direction of the warp. If $[u_s, u_e]$ defines the segment of interest, the knot refinement is applied to get more control points for the segment. If p is the degree of the curve, the control points involved for the refined segment $[u_s, u_e]$ are $\mathbf{P}_i, i = s, \ldots, e-p-1$. To apply Eq. (4.10), we need to associate a function value $f_i = f(t_i)$ for each control point $\mathbf{P}_i$ to be repositioned. Figure 4.3 shows examples of curve warping using different functions $f(t)$.

Now let's consider the case of 1D NURBS curves. Because warping is implemented using control point repositioning, for on 1D curves, we will be using control

value modification. Equation (4.10) can be adapted for 1D NURBS curves as follows

$$\hat{\mathcal{P}}_i = \mathcal{P}_i + f(t)d \tag{4.11}$$

$\hat{\mathcal{P}}_i$ is the modified control value; $f(t)$ is the function that control the warp; d is a constant for limiting the warp *amplitude*. Notice that we don't have the term $\mathbf{W}$ that controls the direction of the warp; this is because we can modify a 1D NURBS curve only along one direction (up/down). The tricky part is to choose the parameter values t_i and hence the function $f(t_i)$ such that the modified control values produce the desired warp. This often depend on the function $f(t)$ used.

4.5 User Interface Issues

One important result seems to emerge from the discussions of the previous sections. The mathematical model of 1D NURBS curves allows the user to get most of the features present in the 2D/3D NURBS curve model. But the main issue is that not all of these features can be made available to the user in an interactive and intuitive way similar to the way 2D/3D NURBS curves can provide. The most dominant user interface issue orbits around the control polygon as it transpires from the discussions of section 4.3 and 4.4.

To get around the problem, we can provide, in the graphical representation of a 1D NURBS curve, an *emulation* of the control polygon. To do that, we make use of the shape modification approach presented in case 3 of section 4.4. When the user modifies the shape by picking a point on the curve itself, it is usually the control value $\mathcal{P}_k$ of the basis function $R_{k,p}(u)$, whose maximum lies closest to the picked point, that is modified to get the desired shape. It seems natural then to map the horizontal positions of the control values along the parameter u axis with the positions of the maximums of the associated basis functions. The control polygon will be emulated by a set of *virtual control points* P_i defined by:

$$P_i \;=\; (x_i, y_i) \;=\; \begin{pmatrix} u \\ \mathcal{P}_i \end{pmatrix} \;; \quad R_{\max} = R_{i,p}(u) \tag{4.12}$$

By joining these virtual control points we obtain what we named an *emulated control polygon*. With this new interface, the user can modify a control value by picking its

associated virtual control point. A user can also select a point on the virtual control polygon, this time, the system will have enough information to apply the inverse knot insertion if needed. It is also possible to use this virtual control polygon to compute the quantity TOL.

Unlike the case of 2D/3D curves where the three scenarios (case 1, 2 and 3) lead to potentially different shapes, using this new interface, the three approaches use the same heuristic to achieve the modification; and consequently, the user gets the same shape for each scenario.

Chapter 5
Implementation

5.1 Introduction

The prototype we are presenting in this chapter is not intended for the general user. It is not a working program for an animator to use. It is rather an experimental tool which exposes some of the insides of the model that should be hidden from the user in a real world application. The main objective of this tool is to demonstrate the possibility of using 1D NURBS curves for interactive design of motion graphs.

Even though we did investigate many aspects of NURBS curves and also have presented many algorithms and shape modification tools, not all of the algorithms and tools were implemented in this prototype. Because of the scope of this project, we had to limit ourselves. We implemented the single knot insertion, multiple knot insertion and knot refinement algorithms. We also implemented a simple interface that allows the user to modify the shape of the curve using control value and weight modifications.

We used implementations provided in [10] for the single knot and multiple knot insertion and knot refinement algorithms. We also used some of the utility algorithms provided. Examples are algorithm for computing a single non-rational basis function for a given parameter value, algorithm for computing all the nonzero basis functions at the same time, algorithm for finding the knot span and algorithm for computing a curve point. The implementations in [10] were intended for 2D/3D non-rational B-spline curves. Hence, we had to adapt them for the context of 1D rational B-spline curves.

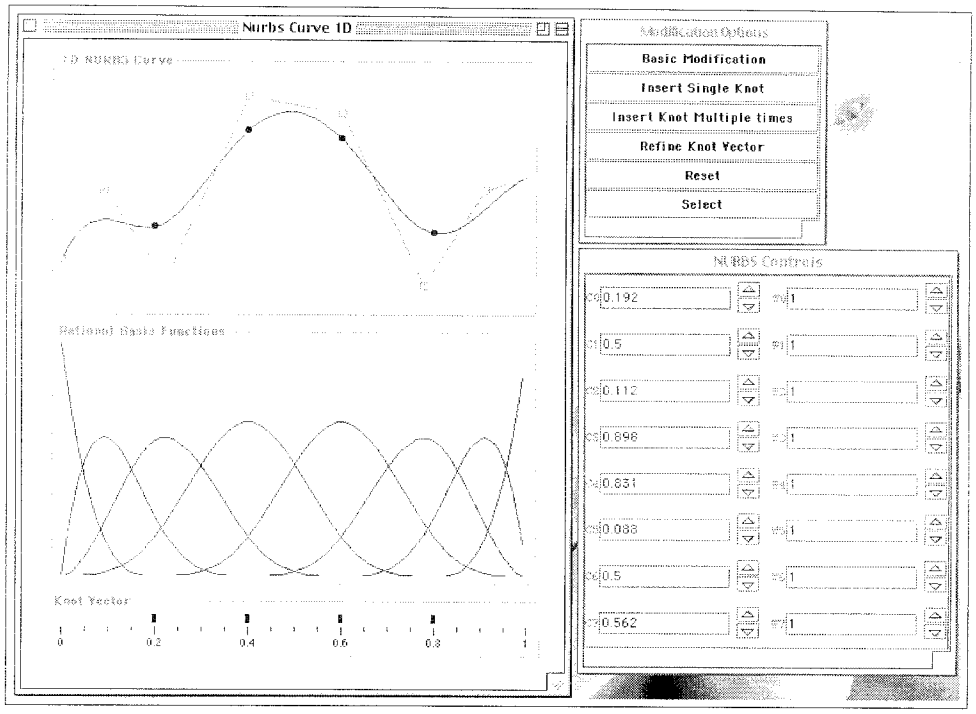


Figure 5.1: Screenshot of the different windows of the prototype application.

5.2 The Prototype

Figure 5.1 is a screenshot of the different parts of the application. The main window, titled "Nurbs Curve 1D", contains three panels. The top panel is the graphical representation of the NURBS curve; the middle panel displays the rational basis functions; the bottom panel displays the knot vector. On the curve canvas, the horizontal axis represents the variation of the parameter u in the range $[0, 1]$; the vertical axis represents the values $\mathcal{C}(u)$; the origin is at the bottom left corner of the panel; the circles on the curve correspond to the breakpoints (section 2.2); they mark the knot value positions. The squares are what we described in section 4.5 as the virtual control points. The polygon is the virtual control polygon also described in section 4.5.

The window titled "Nurbs Controls" contains a list of the control values, labelled c_i and their associated weights labelled w_i. The window titled "Modification Options" provides direct means to apply some of the algorithms mentioned earlier. The labels are self explanatory.

The user can interact with the prototype in several ways. The user can directly

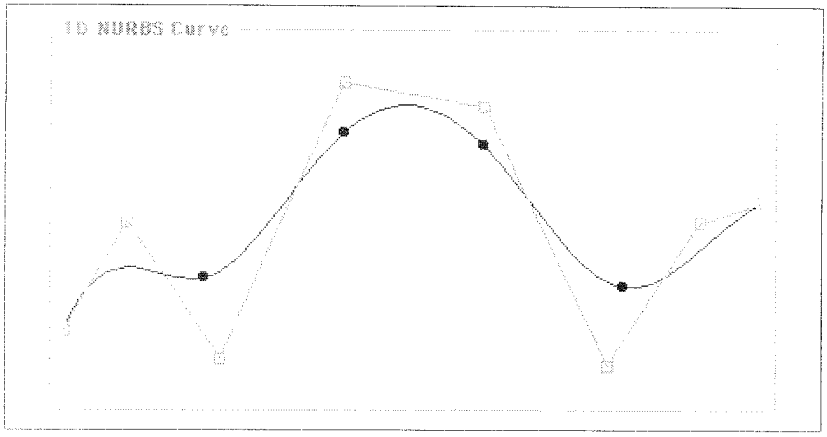


Figure 5.2: Example of a cubic 1D NURBS curve.

modify the control values and the weights by changing the values in the corresponding text fields. The user gets an instant feadback of the result. This is provided as an experimental feature to show that control value modification for 1D NURBS curves achieves the same type of control as control point repositioning for 2D/3D curves. This also applies for the weight values to a certain extend.

The user can modify the curve by dragging any point on the curve itself. Modifying the curve by dragging the virtual control points is not yet active. A user can select any of the modification options provided in the window "Modification Options". For example, by selecting "Insert Single Knot", the user can pick any location on the curve and the knot will be inserted at the right position in the knot vector.

There is another interaction feature for the user to experiment with. The user can modify the knot vector directly. It is not known of any intuitive or meaningful way to predict the reesult of change the knot positions in the knot vector. The user can dragg the knot handles to observe the effect it has on the basis functions and the curve.

Figure 5.2 is a closer look at the curve in Figure 5.1. The curve is cubic (order 4) with height control values and a uniform knot vector. It will serve as a reference when we show the results of applying some of the algorithms and shape modification tools to the same curve. Figure 5.3 is the result of inserting a single knot value in the knot vector of the curve of Figure 5.2: initially, there were height control values, a single knot is inserted, the number of control values increases by one, the number

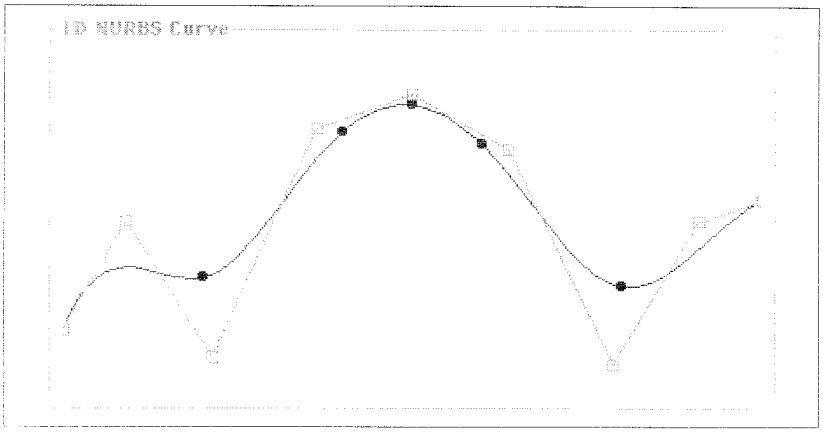


Figure 5.3: The insertion of a knot value ($u = 0.5$) produces a new control value.

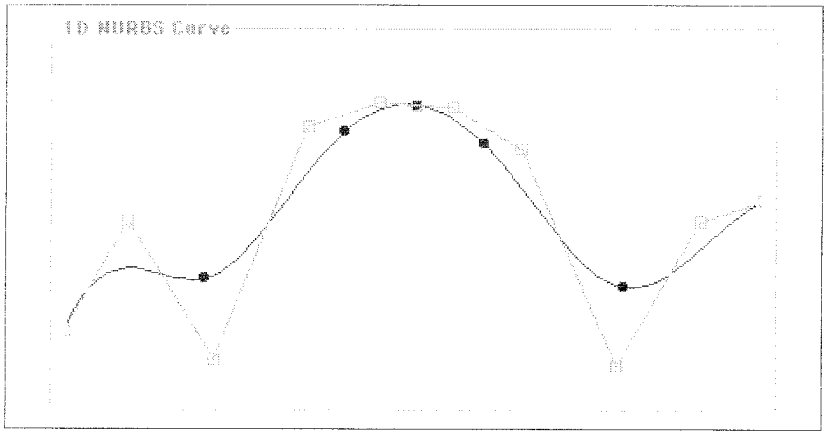


Figure 5.4: The knot value ($u = 0.5$) is inserted degree times.

of modified control values is (p+1) = 4. Figure 5.4 shows the result of inserting the same knot value *degree* times. Because the knot is inserted degree times, this forces the curve to interpolate the corresponding virtual control point. Figure 5.5 shows the result of repositioning the interpolated control point. An example of the application of the knot refinement algorithm is illustrated in Figure 5.6. The behavior of the virtual control polygon is similar to the behavior of the control polygon of 2D/3D NURBS curves (see Figure 3.2, Chapter 3): the control polygon gets closer to the curve.

The result of Eq. (4.9) is demonstrated in Figure 5.7. The weight of the control point P_6 tends to infinity ($w_6 \to \infty$), in other words, w_6 takes a very large numeric value (1.0e+9): the corresponding basis function $R_{6,3}(u)$ takes the value 1 in the

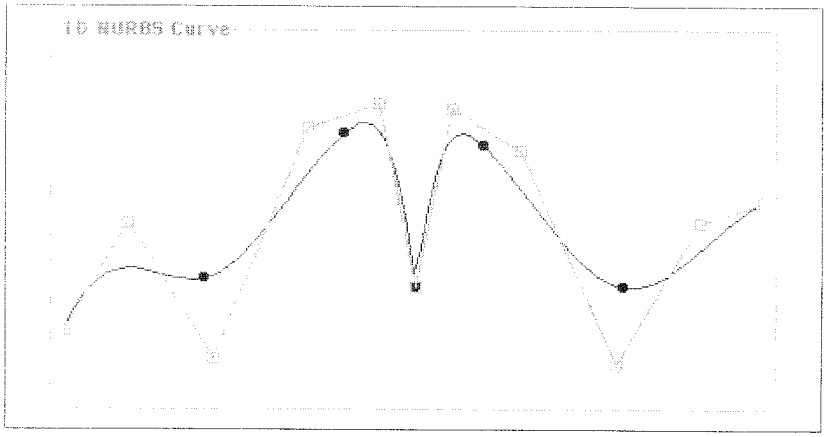


Figure 5.5: The control point resulting from the multiple knot insertion of Figure 5.4 is repositioned.

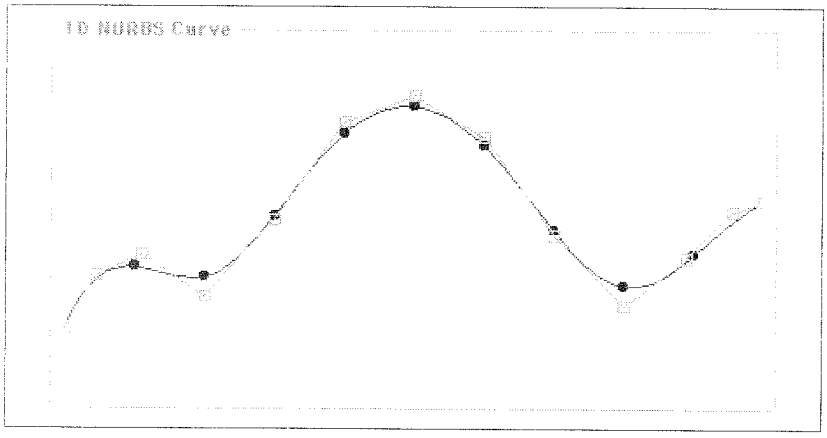


Figure 5.6: The curve after one midpoint knot refinement.

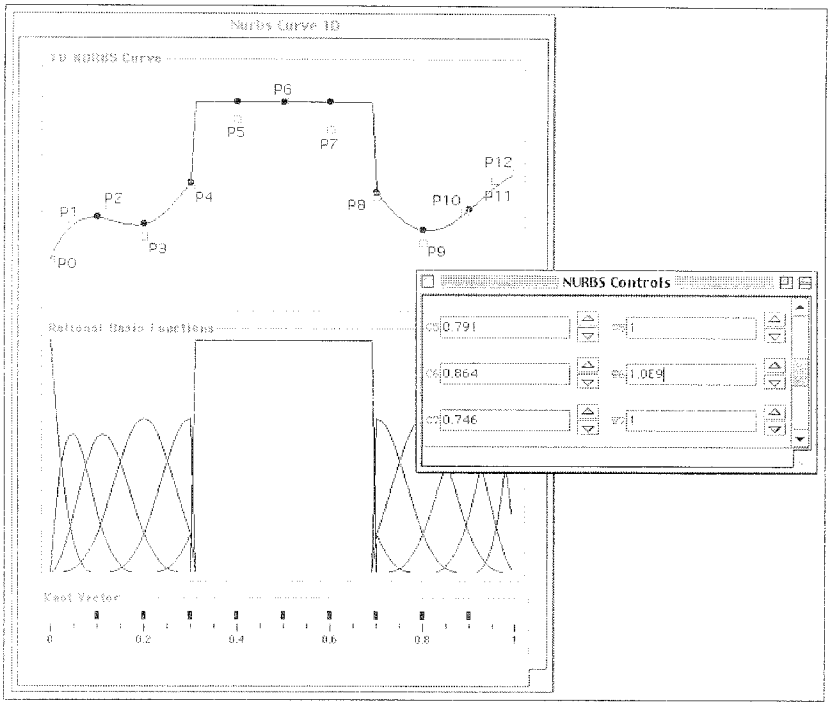


Figure 5.7: The curve after one midpoint knot refinement.

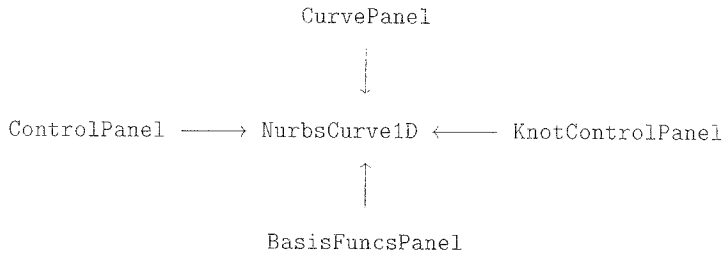
interval $u \in [0.3, 0.5)$ and the curve flattens over the same interval.

From the different examples we presented so far, we can clearly notice that the behavior of a 1D NURBS curve when applying the different algorithms is similar in many ways to the case of 2D/3D curves. This experiment consolidates the results stated in the different sections of chapter 4.

5.3 Design

The architecture of the application follows to a certain extend the Model-View-Controller (MVC) paradigm. The center of the architecture is the class `NurbsCurve1D`. It encapsulates the 1D NURBS curve model. It's the *model* part of the architecture. It defines the state of the curve (class attributes) and the operations used to modify it (class methods). There are four other important classes presented in the folowing

diagram:



`CurvePanel` and `BasisFuncsPanel` represent the *view* components of the application. They allow for the visual representation of the model. They are used to produce respectively the curve panel and basis function panel in window "Nurbs Curve 1D" of Figure 5.1. `KnotControlPanel` and `ControlPanel` represent the *controller* part. They allow for the control of the model. They are used to produce respectively the knot control panel (bottom of "Nurbs Curve 1D" window in Figure 5.1) and the panel in the "NURBS Controls" window. All these elements have a handle on `NurbsCurve1D`. They are also registered listeners of the state of the `NurbsCurve1D` model. Each time the state of `NurbsCurve1D` is modified, these components are notified of the change and react accordingly.

The prototype runs as a Java application. It should run on any virtual machine that support jdk 1.2 and Swing 1.1. The source code provided in [10] was written in C. We had to adapt the original design of the algorithms and port the code to Java to be usefull for in our object-oriented system. The development tools we used are:

- `Language: Java`
- `jdk 1.2 and Swing 1.1.`
- `IDE: Metrowerks CodeWarrior`
- `platform: MacOS`

When the application starts, it loads a text file containing the description of a 1D NURBS curve. If no file is supplied, the application uses a set of default values corresponding to a flat cubic curve. The following is an example showing the format of the 1D NURBS file:

```
Section 1DNURBSCurve {
  ID          "Sample"  ;
  DIMENSION   1  ;
```

```
  ORDER       4  ;
  NUMCONTROL  14 ;
  NUMKNOTS    18  ;
  KNOTS       0 0 0 0  0.138814  0.227824  0.325822  0.425352  0.518801
              0.615333  0.712039  0.81034  0.887647  0.941774  1 1 1 1 ;
  CONTROLS    0  0.0570624  0.149831  0.270263  0.359971  0.427087
              0.472277  0.480392  0.463939  0.419849  0.36352
              0.306316  0.269868  0.249274 ;
  WEIGHTS     1 1 1 1 1 1 1 1 1 1 1 1 1 1 ;
}
```

This example describes a cubic (order 4) 1D NURBS curve; all the weights are set to 1. The knot vector is non-uniform (knots are not equally spaced).

The prototype is designed in a way that provides the flexibility needed if the prototype is to be extended with new features. It is possible to add new algorithms to the model with very little change in the code. It is also possible to enhence the user interface (the view part) without interfering with the model. One can add more controls as needed with very litle changes to the view and no changes to the model.

Conclusion

The current tendency in computer animation systems development is the design of user-friendly control systems. The challenge is to obtain user-controlled motion with reduced computational complexity and computation times. Many 3D animation packages use graphical representations called motion graphs. Many packages use two-dimensional B-splines because of their power to represent free-form curves and common analytic curves. However, the current implementations of motion graphs, even though they provide the right user-friendly control, do not necessarily reduce the computational complexity or the computation costs.

The issues of concern are the looping behavior exhibited by two-dimensional B-splines. When used as a motion graph, it is possible for the curve to loop back which is not a desired effect. The second issue is the complexity involved in mapping the curve input/output with the motion parameter input/output. A reparameterization of the curve is usually needed which in itself not an easy task. The third issue is the computation cost due to the dimension of the B-splines.

The objective of this work was to study the possibility of using One-dimensional Non-Uniform Rational B-spline (NURBS) curves for interactive design of motion control graphs. The motivation behind this objective is the potential One-dimensional B-splines have to solve some of the issues encountered in motion graphs.

In order to replace 2D curves by 1D curves when used for motion graphs, we have to make sure that we preserve most of the power and flexibility of 2D curves that made them a good choice in the first place.

This investigation shows that most of the important properties of 2D NURBS model are exhibited by 1D NURBS curve model. Properties that are directly related to the rational basis functions such as nonnegativity, partition of unity and local support are inherited by 1D NURBS curve model. Geometric properties such as

local approximation and end point continuity, are also maintained. Most of the properties that are not maintained are not of critical importance in the context of motion graph usage.

This study also shows that most of the fundamental geometric algorithms developed for 2D NURBS are still applicable to 1D NURBS curves. In particular, knot insertion, knot refinement and degree elevation can be applied with no significant change. There are though some minor adjustments that have to be made for the knot removal and degree reduction algorithms.

This study highlights some issues related to 1D NURBS curves that one has to take into account if considering to use 1D NURBS model. They are mainly user interface problems. There is no obvious and intuitive way to visualize the control values if there is need to reveal them in the interface. There is no equivalent, in 1D NURBS model, to the control polygon of 2D curves. This is an important problem since many of the shape modification tools rely on the control polygon to achieve the desired modification. However, we showed that there are ways to get around these difficulties and still be able to use similar tools to achieve interactive shape modifications.

1D curves have a distinctive behavior, when the weight of a control value increases to infinity, which is different from the one exhibited by 2D curves. If a weight w_i increases to infinity, the corresponding rational basis function $R_{i,p}$ takes the value 1 in the interval of the parameter u where it is nonzero; as a consequence, the curve flattens over the same u interval.

The implementation, developed during this investigation, is an attempt to provide an experimental tool that demonstrate the possibility of using 1D NURBS curves for interactive design of motion graphs. The user can interact with 1D curves in much the same way as he might do with 2D curves. The implementation was used to confirm all of the results obtained. It did show that the behavior of 1D NURBS curves is similar in many ways to the behavior of 2D NURBS curves; in other words, the power and flexibility of 2D curves is also exhibited by 1D NURBS curves.

1D NURBS model has the added advantage of solving some of the issues exposed earlier. 1D NURBS curve model provides us with curves that are functions, mathematically speaking. This intrinsic property of the curve forces the looping back behavior never to occur. It is also easy to map between the 1D curve input/output

and the motion parameter input/output, hence reducing the computation complexity. The dimension of 1D NURBS curves reduces the computation cost necessary to generate the graphical representations of the curves.

Future Work

This work can be divided in two main parts. The first part is an investigation of the properties of the underlying mathematical model of NURBS curves and the devised algorithms and shape modification tools. The second part is the experimentation with an implementation of the fundamental algorithms and shape modification tools. An extension of this work can take the same approach. It is possible to push further the study of the NURBS mathematical model and extend the implementation by including new algorithms and shape modification tools.

An area we did not explore is the properties of the derivatives of NURBS curves. Changes in curve derivatives are used in techniques known as *constraint-based curve modifications*. In other words, the tangents and curvature of a curve are controlled by specifying constraints on the first and second order derivatives.

The algorithms and shape modification tools that have been mentioned and not implemented in the prototype are a good start for an extension of the prototype. There are also other shape operators such as *flattening* and *bending* that can be included in the implementation.

User interfaces used as medium to interact with 1D NURBS curves are as important as the mathematical model behind NURBS curves. A study of an adequate user interface that hide the inner of the model and simplifies the user learning curve when using all the possible shape modification tools is another good area of investigation.

A motion graph can be a result of an *ab initio* design, but it might also be the result of a collection of experimental data. In this case, a technique known as *curve fitting* is usually used to either *interpolate* or *extrapolate* the data points. It would be very interesting to see how 1D NURBS curves can improve or simplify this process.

Appendix

Figures 5.8 to 5.10 are screenshots of motion graphs used in commercial 3D animation packages (from reference [5]).

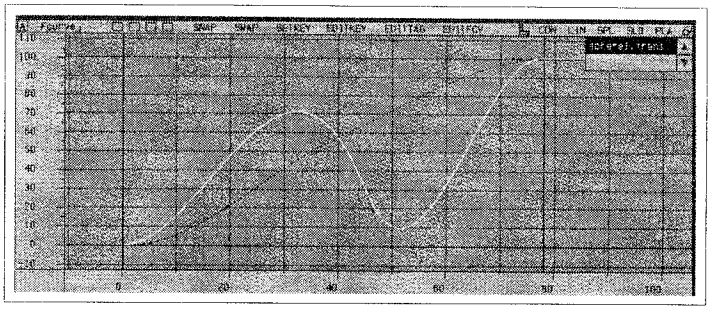


Figure 5.8: In Softimage, a motion graph is called an *FCurve*.

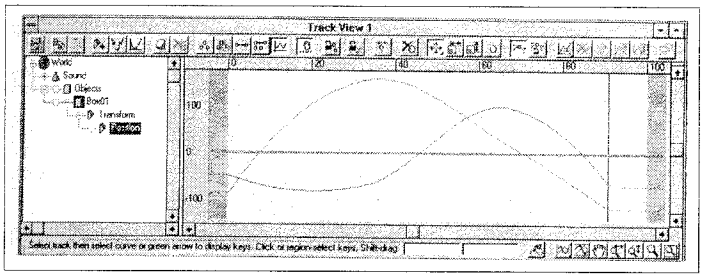


Figure 5.9: In 3D Studio Max, it is called a *Function Curve*.

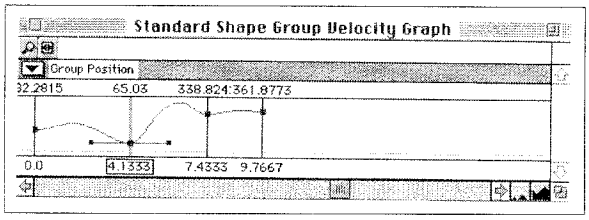


Figure 5.10: In ElectricImage, it is called a *Velocity Curve*.

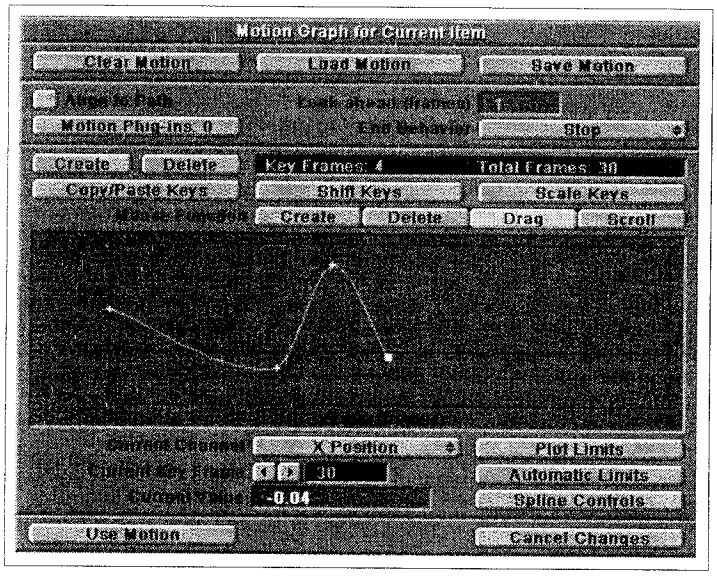


Figure 5.11: In LightWave, it is called a *Motion Graph*.

Bibliography

[1] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques. Theory and Practice*. Addison-Wesley, Reading, MA, USA, 1992.

[2] W. Boehm and H. Prautzsh. The insertion algorithm. *CAD*, 17(2):58–59, 1985.

[3] David F. Rogers and J. Alan Adams. *Mathematical Elements for Computer Graphics*. Mac-Graw Hill, New York, NY, USA, 1990.

[4] Alfredo Pina Eva Cerezo and Francisco J. Serón. Motion and behavior modelling: State of art and new tends. *The Visual Computer*, 15(3):124–146, 1999.

[5] George Maestri. *Digital Character Animation*. New Riders, Indianapolis, IN, USA, 1996.

[6] James D. Foley, Andries Van Dam, Steven K. Feiner and Jhon F. Hughes. *Computer Graphics, Principles and Practice*. Addison-Wesley, Reading, MA, USA, 1996.

[7] B. Jüttler. Visualization of moving objects using dual quaternion curves. *Comput & Graph*, 18:315–326, 1994.

[8] L. Rosenblum, R. A. Earnshaw, J. Encarnacao, H. Hagen, A. Kaufman, S. Klimenko, G. Nielson, F. Post and D. Thalmann. *Scientific Visualisation. Advances and Challenges*. Academic Press, San Diego, CA, USA, 1994.

[9] Les Piegl. *Fundamental Developments of Computer-Aided Geometric Modeling*. Academic Press, San Diego, CA, USA, 1993.

[10] Les Piegl and Wayne Tiller. *The NURBS Book*. Springer-Verlag, Berlin, Heidelberg, Germany, 1997.

[11] T. Spencer-Smith and G. Wyvill. Four dimensional splines for motion control in computer animation. state-of-the-art in computer animation. In *Proceedings of Computer Animation'89*, pages 153–167. Springer-Verlag, 1989.

[12] W. Tiller. Knot-removal algorithms for nurbs curves and surfaces. *CAD*, 24(8):445–453, 1992.