Automatic Inbetweening of Keyframes Composed of Spline Curves Satisfying Various Constraints

Ronald Thomas Hardock

CS-89-58

November, 1989

Automatic Inbetweening of Keyframes Composed of Spline Curves Satisfying Various Constraints

by

Ronald Thomas Hardock

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Mathematics

in

Computer Science

Waterloo, Ontario, Canada, 1989

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Ronald Hardock

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Ronald Hardock

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

This thesis presents a method one can use to obey a certain class of constraints between spline curves in a keyframe inbetweening environment. The general problem is ensuring that constraints, which are true for keyframes, will also be true for the computer-generated inbetween frames.

Various types of constraints are outlined. A set of conditions on the interpolation method are outlined such that *linear mapping* constraints can be obeyed. Mention is also made of various constraints that are not expressible as linear mappings, to point to directions of future research.

Acknowledgements

I would like to thank the following people:

Dr. Kellogg S. Booth, Dr. Wendy Seward, and Kevin Schleuter, for their many valuable comments on this thesis.

Dr. Richard Bartels, for his patience, guidance, insights, and support in seeing this thesis to completion.

The members of the computer graphic's lab for their help and friendship.

My wife, Annie, for her love and patience.

My parents, for always being there when I need them, and for their support.

And to the Lord, for His help and guidance.

Financial support was provided by the Strategic, Equipment, and Operating grants programmes of the Natural Sciences and Engineering Research Council and by the Information Technology Research Centre. Equipment was provided by donations from the Digital Equipment Corporation and Silicon Graphics.

To Annie, with love.

Table of Contents

List of Illustrations

1. Problem Introduction

This thesis was born as an attempt to solve the "*Mickey's Nose problem*" (or "MN problem" for short). Mickey refers to the well known Walt Disney mouse character. Mickey Mouse was created by Walt Disney studios in the environment of two-dimensional celluloid (abbreviated as "cel") keyframe animation.

Cel-keyframe animation consists of a pipeline (Figure 1.1) going from the story in the designer's mind to the finished product of a complete film. The MN problem is associated with the *inbetweening* stage of this pipeline. The problem arose from an attempt on our part to inbetween spline curves by simply inbetweening their control vertices.

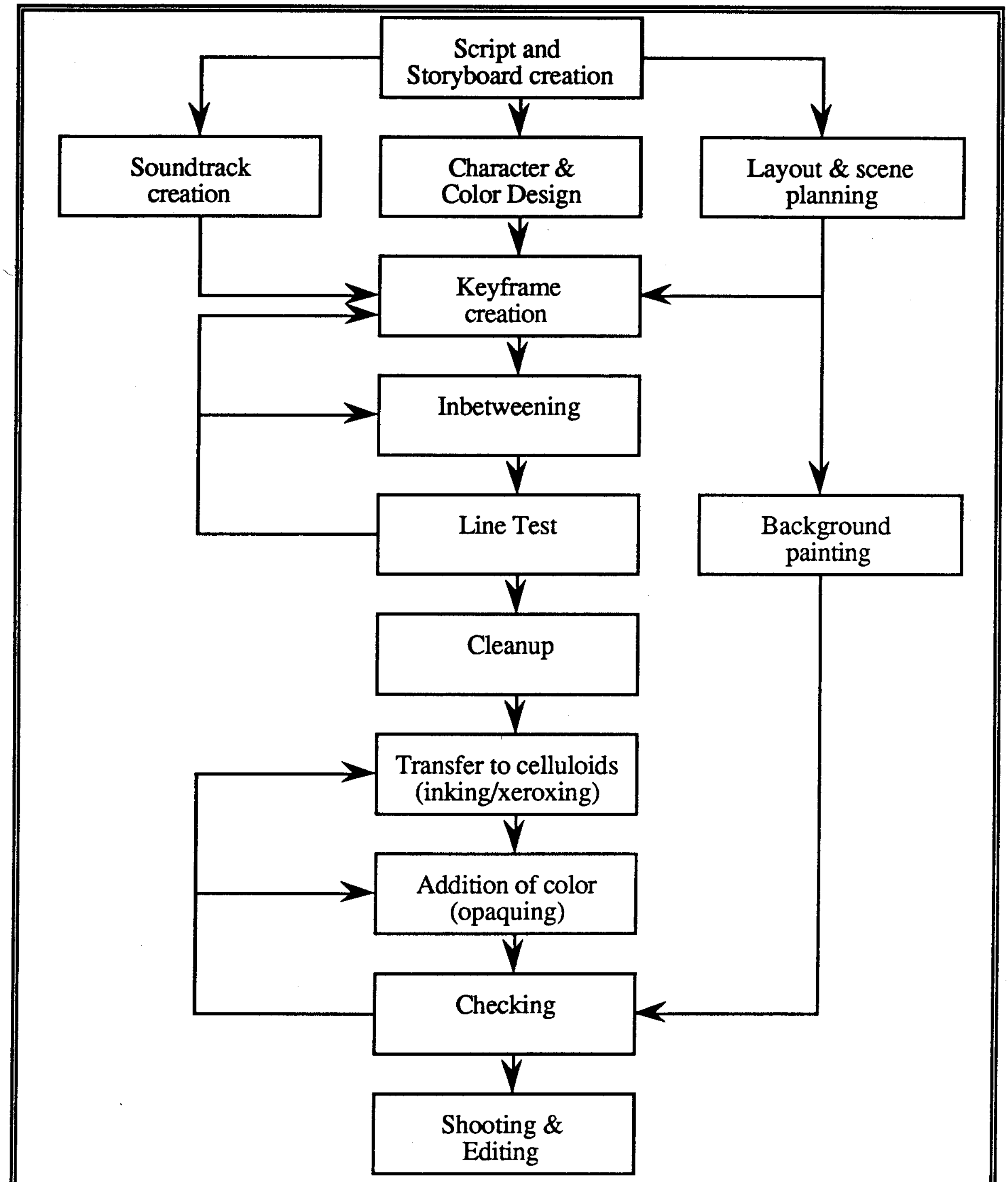

Figure 1.1 The cel animation pipeline (from [Hardtke87])

The MN problem can be stated as the following: "Given keyframes in which Mickey's snout (a spline curve) is always attached to his nose (another spline curve), how can we ensure that the inbetween frames also obey this constraint?".

Walt Disney Studios does not have a problem with Mickey's nose, or any of the other hundreds of constraints one needs to obey to give a sense of realism to any animation scene. They have, however, very highly skilled animators to do the construction of the keyframes and the inbetweens. The MN problem occurs when we try to replace these skilled inbetweeners with a computer that works with key curves represented by splines. Each keyframe will contain a number of objects, and each keyframe object is represented by a number of key curves. Chapter 2 defines and discusses splines.

In addition to cel-keyframe animation, this thesis work is also useful for the application area of swept surface creation. A swept surface involves taking a curve and moving it along a path. The result is a surface, in three-dimensional space, swept out by the curve. A swept surface can also be created by taking as keyframes various cross-sectional curves along the surface and inbetweening the keyframes to fill in the missing parts of the surface. Thus, two application areas for this thesis are cel-keyframe animation and swept surface modelling. Chapter 3 will briefly outline these two applications and give some examples of how keyframe inbetweening works.

The following is the general form of the problem that we consider here: "Starting with keyframes in which one spline curve always obeys a certain *constraint* with respect to a second spline curve, and assuming that inbetweening these curves is done by inbetweening their respective control vertices, are there any conditions that would ensure that the inbetween frames also obey the constraint?". From now on in this thesis, when reference is made to the MN problem, it is this form of the problem that is intended. In summary, the MN problem derives from computerized keyframe inbetweening, where the keyframes are stored and manipulated as splines and the inbetweening process is defined in terms of key positions of control vertices.

Chapter 4 gives some examples of various constraints one might like to maintain. Chapter 5 gives conditions that ensure that some of these constraints are maintained.

The MN problem was encountered at the University of Waterloo Computer Graphics Laboratory when an attempt was made to utilize a spline curve editor to produce simple animated sequences. Key curves that were adjacent flew apart in the inbetween frames. About four years passed before work started on this thesis to determine why curves did not maintain their key associations under control-vertex inbetweening.

Studying the MN problem involved first running computer simulations to demonstrate and analyze the problem. These simulations were run, but the MN problem did not occur for the point and tangency constraints used by the simulations. Keyframes consisted of two curves that observed point and tangency constraints, and the computer-generated inbetween frames observed the same constraints exactly (Figure 1.2). Other simulations were run to try to duplicate the results of four years ago, and it was demonstrated that the MN problem did definitely exist (Figure 1.3). The only difference in the two situations was the way the keyframes were constructed; thus it was discovered how the MN problem can be solved for this set of constraints. Chapter 5 outlines this solution; the solution basically is a proof that if certain conditions are met for the keyframes, the interpolation method, and the constraints, then the generated inbetween frames will observe the same constraints that hold in the keyframes. These conditions do not turn out to be overly restrictive.

Chapter 6 revisits the constraints mentioned in Chapter 4 and outlines if and how these constraints can be maintained as a result of the solution we found to the MN problem.

Lastly, Chapter 7 discusses some areas for future work.

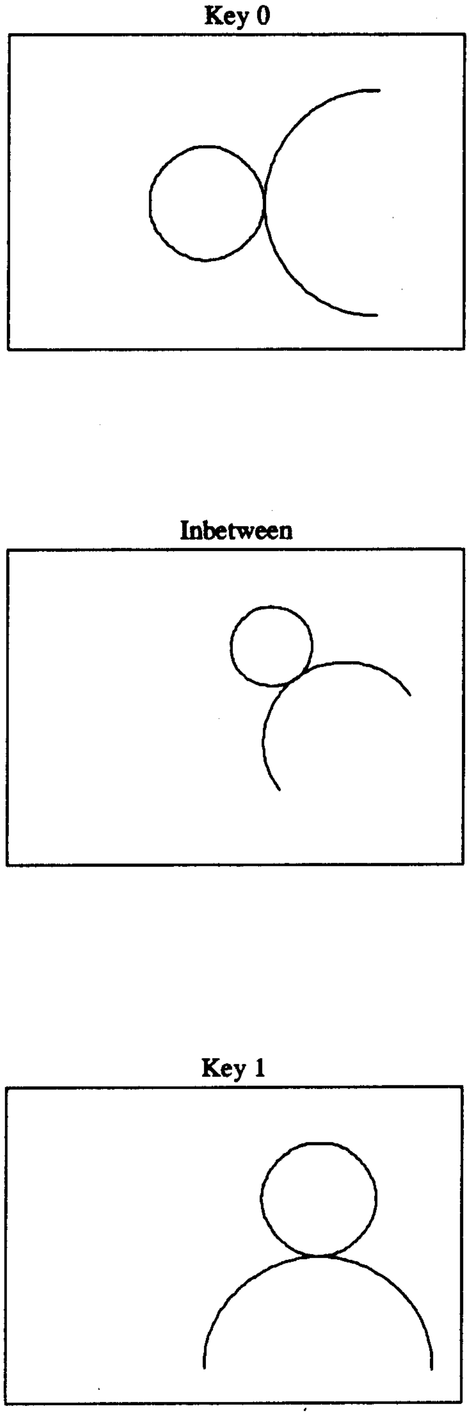

Figure 1.2 Demonstration of Constraints being obeyed.

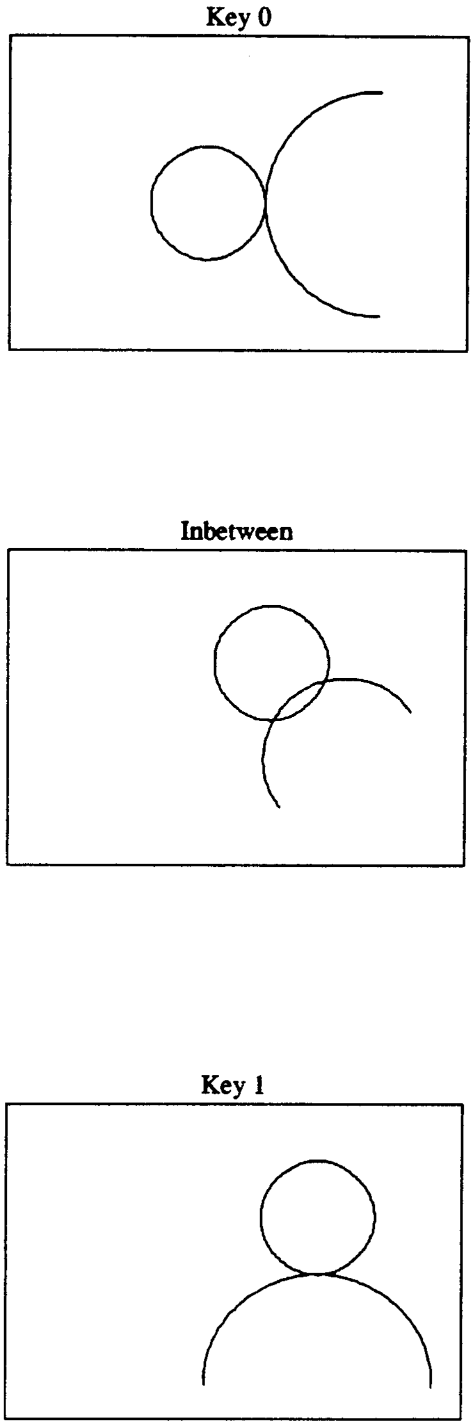

Figure 1.3 Demonstration of Constraints not being obeyed.

2. Introduction to Splines

2.1. Piecewise Polynomials

The concept of *splines* forms the foundation for this thesis. One definition of a spline is to call it a *piecewise polynomial,* where a polynomial $f(x)$ of *degree d* is a function that can be expressed in the form:

$$(2.1) \quad f(x) = \sum_{i=0}^{d} a_i x^i \ .$$

If $a_d \neq 0$, we say that $f(x)$ is of exact degree d. If each segment (that is, each piece) of a spline is of degree *d* then the spline is said to be of *degree d.* Another term one may encounter in the spline literature is *order,* which is simply the degree plus one (so $f(x)$ has order $d + 1$).

A piecewise polynomial is a composite of several polynomials defined over a succession of adjoining intervals. For example, the following is a piecewise polynomial:

$$(2.2a) \quad g(x) = \begin{cases} 0 & \text{if} \quad x < 0 \\ x & \text{if} \quad 0 \leq x \leq 1 \\ 1 & \text{if} \quad x > 1 \end{cases}$$

The piecewise polynomial $g(x)$ is made up of pieces of three polynomials, where two of the polynomials are defined over infinite intervals and are of exact degree 0, and one polynomial is of exact degree 1 and defined over the finite interval $x \in [0,1]$. We can regard $g(x)$ as being a piecewise polynomial of degree 1.

The spline $g(x)$ in (2.2a) is a continuous curve. In general, this continuity is not necessary; hence, the following could also be thought of as a piecewise polynomial:

$$(2.2\text{b}) \quad g(x) = \begin{cases} -1 & \text{if} \quad x < 0 \\ x & \text{if} \quad 0 \le x \le 1. \\ 2 & \text{if} \quad x > 1 \end{cases}$$

From our definition of a spline, both $g(x)$ (2.2) and $f(x)$ (2.1) could be referred to as splines: $f(x)$ is a piecewise polynomial composed of only one piece, while $g(x)$ is a piecewise polynomial of three pieces.

When we consider collections of piecewise polynomials, we require that they be standardized in terms of their intervals, degree, and their continuity between adjacent intervals. Hence, the definition of a spline given by [Thomas86] is "A spline is a piecewise curve with a specified continuity constraint at the knots between the pieces (or spans)."

The term *curve* is used in Thomas' definition since a spline need not be a function with respect to x or y. Splines may be defined in parametric form (see Section 2.4). The splines that we will be dealing with are *piecewise parametric polynomials*. An example of a spline in parametric form is:

$$(2.3) \quad (x, y) = \begin{cases} \text{if} \quad 0 \le t \le 1: \begin{cases} x(t) = t \\ y(t) = t \end{cases} \\ \text{if} \quad 1 < t \le 2: \begin{cases} x(t) = 1 \\ y(t) = 2 - t \end{cases} \\ \text{if} \quad 2 < t \le 3: \begin{cases} x(t) = 3 - t \\ y(t) = 0 \end{cases} \end{cases}$$

(Figure 2.1 shows this curve).

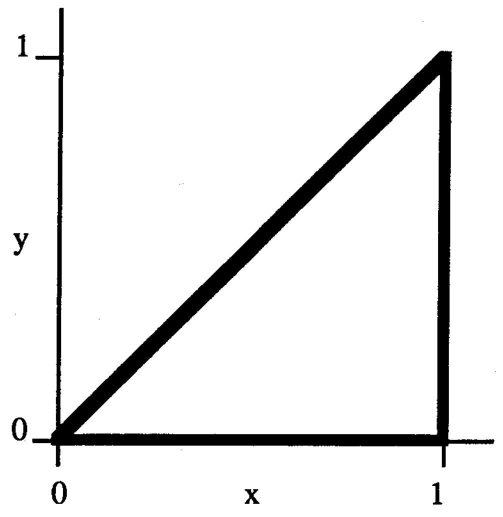

Figure 2.1 An example of a parametric spline (defined by equation 2.3).

Thomas' definition introduces two terms used throughout the spline literature: *knots* and *continuity*. A *joint*, which contains one or more knots, refers to the parametric value where one polynomial piece joins the next piece. Thomas refers to the individual pieces as *spans*, whereas we will use the term *segment*. Two adjacent segments are said to be C^k continuous if they agree in their 0^{th} through k^{th} derivatives at the joint between the segments. C^{-1} continuity denotes no agreement at all. Knots are a mechanism for counting discontinuity. Two splines of degree *d* will meet with C^{d-r} continuity at a joint if the joint contains *r* knots. These conventions are discussed more fully in [Bartels87].

2.2. Motivation

We will use splines for both representing the curves in the keyframes and for the interpolation path used to do the inbetweening. Animators have used splines to define animation trajectories for a number of years (see [Kochanek82]). One advantage of using a spline interpolation path is in the type of controls available to create the interpolation path.

The controls provided might depend upon the application. For example, in the realm of keyframe animation, it is discussed in [Kochanek84] how interpolating cubic splines with local control of tension, continuity, and bias is useful for the animator. It is also mentioned in [Kochanek84] how cubic spline interpolation (compared to linear interpolation) can give smoother animation sequences.

One other advantage of spline interpolation over some other computer animation inbetweening methods such as p-curves and skeletons ([Kochanek82] and [Burtnyk76]) is that information is only needed about the positions of the keyframe points being interpolated to generate the interpolation path. This advantage will be further explained in Section 2.3.

As well as treating the interpolation path as a spline, we also wish to have our keyframes composed of splines. This desire imposes the restriction that each keyframe instance of a curve must have the same, fixed number of segments. This is evident from the fact that we need a correspondence between the entities that we are interpolating. That is, we need to know precisely what segment in one keyframe is being mapped to which segment in the next keyframe. The simplest mapping is a one-to-one mapping at the segment level, and this implies that the two keyframes being interpolated need precisely the same number of segments. Actually, we are interpolating the segments by interpolating their control vertices; a segment of degree d can be represented as a collection of $d + 1$ *control vertices* and a *basis* (see Section 2.5 for an explanation of these terms). Thus, we require that each keyframe have the same number of control vertices. Relaxing this mapping (allowing keyframes composed of a different number of control vertices to be inbetweened) is outside the scope of this work.

2.3. Spline Interpolation

Consider Figures 2.2 and 2.3 which show a point P at four *key* positions. We are concerned with the problem of interpolating between key i and key $(i + 1)$; that is, we need to find a path (a function $P(t)$) to take us from P_i to P_{i+1}. The problem of finding a specific path can also be viewed as a problem of specifying enough conditions so that all *degrees of freedom* of the path function are taken up. A polynomial of degree d has $(d + 1)$ degrees of freedom. A spline of n segments, where each segment is a polynomial of degree d thus has $n(d + 1)$ degrees of freedom. Some of these degrees of freedom will usually be taken up by continuity conditions at the knots.

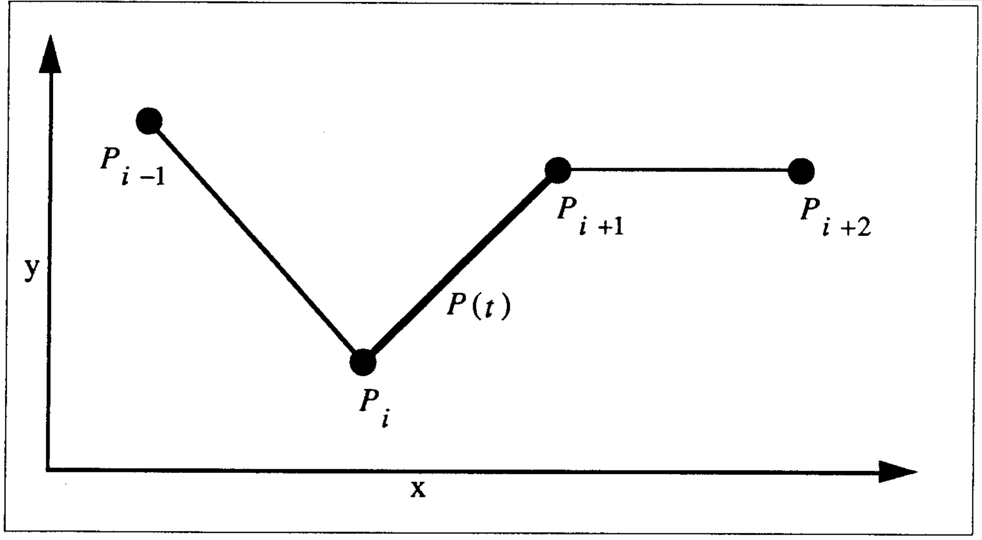

Figure 2.2 Linear interpolation (point P_i belongs to key i)

The linear interpolating function is the following:

$$P(t) = t\,P_i + (1 - t)\,P_{i+1}.$$

As t goes from 0 to 1, the function traverses the linear path from P_i to P_{i+1}. By specifying the two points to be interpolated (P_i and P_{i+1}), one has used up the two degrees of freedom present in linear polynomials.

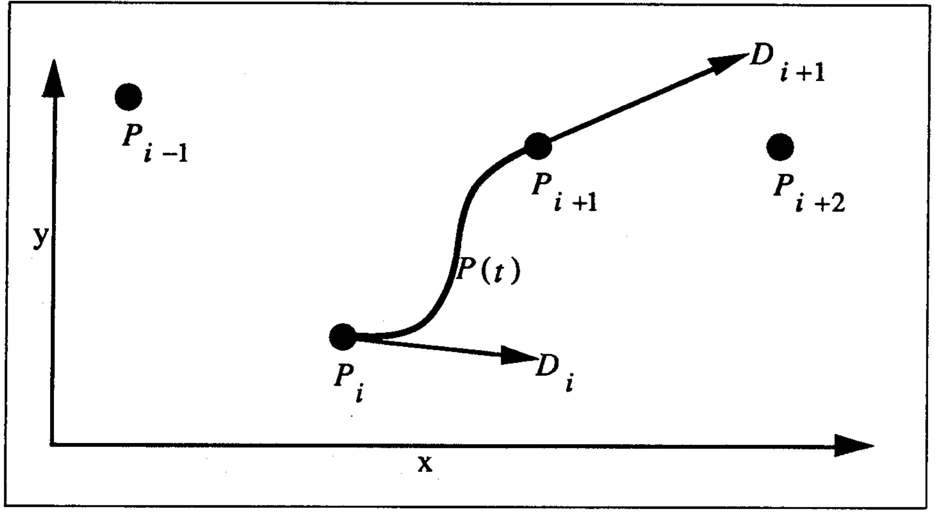

Figure 2.3 Hermite interpolation

By increasing the degree of the interpolating function, one gets more degrees of freedom, and thus greater control over the function. Cubic Hermite interpolation [Kochanek84] is an example of this (Figure 2.3). As cubics have four degrees of freedom, using four positions of P in specifying the interpolation path from P_i and P_{i+1} will use up all the degrees of freedom.. The interpolating function (also called a Catmull-Rom spline) is the following:

$$P(t) = P_i\, h_1(t) + P_{i+1}\, h_2(t) + D_i\, h_3(t) + D_{i+1}\, h_4(t)$$

where $D_i = \frac{1}{2}(P_{i+1} - P_{i-1})$,

$t \in [0,1]$, and

$$\begin{aligned}
h_1(t) &= 2t^3 - 3t^2 + 1 \\
h_2(t) &= -2t^3 + 3t^2 \\
h_3(t) &= t^3 - 2t^2 + t \\
h_4(t) &= t^3 - t^2
\end{aligned} \tag{2.4}$$

The functions $h_j(t)$ are called the Hermite interpolation basis. Using the Hermite interpolation basis leads to the following properties:

$$\begin{aligned}
P(0) &= P_i \\
P(1) &= P_{i+1} \\
P'(0) &= D_i \\
P'(1) &= D_{i+1}
\end{aligned} . \tag{2.5}$$

This discussion has concentrated on the path from key i to key $(i + 1)$; this path can be thought of as a single segment of a spline, where the entire spline goes from key 0 to key m (we have $(m + 1)$ keys). Figure 2.2, for example, shows a three segment linear interpolating spline which has six degrees of freedom. Maintaining value continuity (C^0) at P_i and P_{i+1} provides two conditions, and the four key positions of point P provide four more conditions to satisfy all six degrees of freedom (and give us a unique linear interpolating spline curve). An issue arises with end-point interpolation involving splines of order higher than linear. For example, in Figure 2.3 we used four keys ($(i - 1)$ to $(i + 2)$) to interpolate between two keys (i to $(i + 1)$); if one wished to use the same method to interpolate between keys 0 and 1, one would require a key indexed by (-1). One accommodation we can make is to have extra (perhaps invisible) frames at the end-points, frames which would not be interpolated but result in giving us more control over the trajectory. Another is to impose extra conditions (pertaining to velocity or acceleration, for example) at the beginning and ending of a trajectory.

Providing control for the interpolating spline is very important in order to achieve the desired results. Control can be increased by increasing the number of degrees of freedom in excess of the degrees of freedom needed to interpolate the given keys. The preceding discussion showed (by using examples of linear and cubic interpolation) how increasing the degree of the interpolating spline increases the number of degrees of freedom. Another way to increase the number of degrees of freedom (and thus the control) in the interpolating spline is to increase the number of segments. That is, instead of having a single segment joining P_i to P_{i+1} (as shown previously), one can use two or more segments. Having a multi-segment spline between P_i and P_{i+1} will permit introducing extra keys between P_i and P_{i+1} to satisfy the extra degrees of freedom; these extra keys increase our control over the path of motion.

2.4. Parametric Curves and Surfaces

Recall that our splines are piecewise parametric polynomials. A polynomial in 2-dimensional (x, y) space, where y is a function of x, can be represented in non-parametric form as: $\{x = x, \quad y = f(x); \quad x \in [a, b]\}$. The *parametric form* of the curve involves expressing both x and y as functions of some parameter t. For example, the line segment $\{ y = x; \ y, x \in [a, b]\}$ can be represented in parametric form as $\{x = t, \ y = t; \ t \in [a,b]\}$. The circle $\{x^2 + y^2 = r^2\}$ can conveniently be placed in the following parametric (or polar) form: $\{x = r\cos(t), \ y = r\sin(t); \ t \in [0, 2\pi]\}$.

One can move up to curves in 3-dimensional space by having three independent functions:

$$(2.6)\qquad S(t) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} f^x(t) \\ f^y(t) \\ f^z(t) \end{bmatrix}$$

If each of the three functions in (2.6) is a piecewise polynomial, and if we assume that the joints occur at the same parameter values for each of the functions, then we call $S(t)$ a spline curve. If we divide up the spline curve into its pieces, we get a collection of segments $S_i(t)$. Let us use the following notation for this:

$$(2.7)\qquad S(t) = \begin{cases} S_0(t),\ t \in [t_0, t_1) \\ S_1(t),\ t \in [t_1, t_2) \\ \vdots \\ S_i(t),\ t \in [t_i, t_{i+1}) \\ \vdots \\ S_{n-1}(t),\ t \in [t_{n-1}, t_n) \end{cases}$$

The following section will further explain how to represent these segments. It is important to remember that each segment $S_i(t)$ is in parametric form.

A parametric surface occurs when we use two parameters for our curves instead of one. That is, we make x, y, and z functions of two parameters (equation 2.7).

$$S(u, t) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} f^x(u, t) \\ f^y(u, t) \\ f^z(u, t) \end{bmatrix}$$

2.5. Basis Representation

A set of functions $\left\{ B_j(t);\ 0 \le j \le d \right\}$ is defined to be a *basis* for the polynomials of degree d if and only if:

i) the functions are *linearly independent*, and

ii) the functions *span* the set of polynomials of degree *d*. This concept of spanning means that any polynomial of degree *d* can be expressed as a linear combination of the $B_j(t)$ (equation 2.8).

(2.8) $$f(t) = \sum_{j=0}^{d} a_j B_j(t)$$

Let P^d be the set of all polynomials of degree d $(d \geq 0)$. P^d (over the *field* of real numbers) forms a *vector space* of *dimension* $(d + 1)$. The dimension is the number of linearly independent vectors needed to span the vector space, and consequently, it is equal to the number of degrees of freedom of each element within the vector space as well as the order of the elements within P^d. Further details can be found in [Bartels87] and in algebra texts such as [Strang80] and [Coleman73].

As an example of a basis, equation (2.1) illustrated the use of the *power basis* to represent any member of P^d. The power basis is defined by the set of functions:

(2.9) $$\left\{ B_j(t) = t^j ; 0 \leq j \leq d \right\}.$$

Let us focus our attention to the i^{th} segment of a spline curve. In 3-dimensional parametric form, let the three functions $\left\{ \left(x_i(t), \ y_i(t), \ z_i(t) \right) ; \ t_i \leq t \leq t_{i+1} \right\}$ represent this segment. Also, let each of these functions belong to P^d on the interval $\left[t_i, t_{i+1} \right]$. The following form represents any parametric spline segment on the interval $\left[t_i, t_{i+1} \right]$:

$$x_i(t) = \sum_{j=0}^{d_i} V^x_{i,j} \; B_{i,j}(t)$$

$$\text{(2.10)} \quad y_i(t) = \sum_{j=0}^{d_i} V^y_{i,j} \; B_{i,j}(t)$$

$$z_i(t) = \sum_{j=0}^{d_i} V^z_{i,j} \; B_{i,j}(t)$$

The coefficients $\left\{ V^x_{i,j}, V^y_{i,j}, V^z_{i,j} \right\}$ are called the *control points* or *control vertices* of the spline, with respect to the basis $B_{i,j}(t)$. For a particular i^{th} segment, each of the three functions in (2.10) uses the same degree d_i and the same basis.

There are many different sets of bases $B_{i,j}(t)$ that one can use. Two examples shown thus far are the power basis (2.8), and the cubic Hermite basis (2.4). One can represent splines such as *Bézier* (2.11a) and *B-spline* (2.11b) by using (2.10) with the proper choice of a basis. Depending on which basis one chooses, certain geometric associations might hold between the control points and the actual curve (examples of some properties were shown in equation (2.5)).

The Bézier basis of degree *d* on the interval $[t_i, t_{i+1})$ (where $t_i < t_{i+1}$) is:

$$\text{(2.11a)} \qquad B_{i,j}(t) = \binom{d}{j} \bar{t}^{\,j} (1 - \bar{t})^{d-j} \quad , \quad \text{where } \bar{t} = \frac{t - t_i}{t_{i+1} - t_i} .$$

The uniform cubic B-spline basis on $[t_i, t_{i+1})$ is:

$$
\text{(2.11b)}\qquad
\begin{aligned}
B_{i,0}(t) &= \frac{1}{6}\left(1 - 3\bar{t} + 3\bar{t}^2 - \bar{t}^3\right)\\
B_{i,1}(t) &= \frac{1}{6}\left(4 - 6\bar{t}^2 + 3\bar{t}^3\right)\\
B_{i,2}(t) &= \frac{1}{6}\left(1 + 3\bar{t} + 3\bar{t}^2 - 3\bar{t}^3\right)\\
B_{i,3}(t) &= \frac{1}{6}\bar{t}^3
\end{aligned}
$$

where $\bar{t}$ is as was given in (2.11a). This basis is suitable for use if $t_i = i$ and provides automatic C^2 continuity at each joint [Bartels87].

Some bases are conveniently used in a multiple segment format. The idea is to combine the basis functions (such as (2.11b)) to be a single piecewise polynomial (that is, a basis spline). The uniform cubic B-spline basis can be written as a piecewise polynomial over the entire $(-\infty, +\infty)$ range as follows:

$$
\text{(2.12)}\quad B_i(t) = \begin{cases}
\frac{1}{6}\bar{t}^3 & t_i \le t < t_{i+1},\ \bar{t} = \left(\dfrac{t - t_i}{t_{i+1} - t_i}\right)\\
\frac{1}{6}\left(1 + 3\bar{t} + 3\bar{t}^2 - 3\bar{t}^3\right) & t_{i+1} \le t < t_{i+2},\ \bar{t} = \left(\dfrac{t - t_{i+1}}{t_{i+2} - t_{i+1}}\right)\\
\frac{1}{6}\left(4 - 6\bar{t}^2 + 3\bar{t}^3\right) & t_{i+2} \le t < t_{i+3},\ \bar{t} = \left(\dfrac{t - t_{i+2}}{t_{i+3} - t_{i+2}}\right)\\
\frac{1}{6}\left(1 - 3\bar{t} + 3\bar{t}^2 - \bar{t}^3\right) & t_{i+3} \le t < t_{i+4},\ \bar{t} = \left(\dfrac{t - t_{i+3}}{t_{i+4} - t_{i+3}}\right)\\
0 & \textit{otherwise}
\end{cases}
$$

Notice that each $B_i(t)$ is a spline and at any t value at most four basis functions $B_i(t)$ will be non-zero. The four non-zero basis functions will be identical to the polynomials in (2.11b).

2.6. Basis Matrix Representation

Letting $S_i(t)$ be the i^{th} segment of a degree d spline, it is sometimes convenient to rewrite (2.10) in the following matrix form:

$$S_i(t) = B_i V_i,$$

where $S_i(t) = [\, x_i(t) \;\; y_i(t) \;\; z_i(t) \,]$,

$$V_i = \begin{bmatrix} V^x_{i,0} & V^y_{i,0} & V^z_{i,0} \\ V^x_{i,1} & V^y_{i,1} & V^z_{i,1} \\ \vdots & \vdots & \vdots \\ V^x_{i,d-1} & V^y_{i,d-1} & V^z_{i,d-1} \\ V^x_{i,d} & V^y_{i,d} & V^z_{i,d} \end{bmatrix},$$

and $\quad B_i = [B_{i,0}(t) \; B_{i,1}(t) \; \ldots \; B_{i,d-1}(t) \; B_{i,d}(t)].$

A more convenient matrix representation can be found by observing that each $B_{i,j}(t)$ is a member of P^d; thus, we can rewrite $B_{i,j}(t)$ using the power basis:

$$(2.13) \quad B_{i,j}(t) = \sum_{k=0}^{d} b_{i,j,k} \, t^k .$$

One of the parametric functions in (2.10) can then be rewritten as:

$$(2.14) \quad x_i(t) = \sum_{j=0}^{d} V^x_{i,j} \sum_{k=0}^{d} b_{i,j,k} \, t^k$$

What we have done here is to convert one basis, being $B_{i,j}(t)$ in (2.10), to another basis, the power basis in (2.14)). Once in the power representation, we can convert the form of (2.14) to following matrix representation:

(2.15) $S_i(t) = T\,\bar{B}_i V_i$,

where $T = [\,t^0\ t^1\ \dots\ t^{d-1}\ t^d]$, and

$$(2.16)\quad \bar{B}_i = \begin{bmatrix} b_{i,0,0} & b_{i,1,0} & \cdots & b_{i,j,0} & \cdots & b_{i,d,0} \\ b_{i,0,1} & b_{i,1,1} & \cdots & b_{i,j,1} & \cdots & b_{i,d,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & b_{i,j,k} & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{i,0,d-1} & b_{i,1,d-1} & \cdots & b_{i,j,d-1} & \cdots & b_{i,d,d-1} \\ b_{i,0,d} & b_{i,1,d} & \cdots & b_{i,j,d} & \cdots & b_{i,d,d} \end{bmatrix}.$$

Many of the current graphic workstations (the Silicon Graphic Iris for example) have built-in support for this basis matrix format. The most convenient such representations are those for which the matrix $\bar{B}_i$ can be standardized so that one version of $\bar{B}_i$ works for all intervals (ie. $\bar{B}_0 = \bar{B}_1 = \dots = \bar{B}_n$). Bézier and uniform B-spline curves can be handled in this way.

2.7. Multiple Segment Representation

A multiple segment spline such as given by (2.12) can be expressed with the following summation:

$$(2.17)\quad S(t) = \sum_{i=0}^{n} V_i B_i(t),$$

where $t \in [\,t_0, t_n)$ is the range in which the spline is defined. The control vertices are denoted as V_i, and we have a total of $n + 1$ control vertices.

For example, suppose we have a cubic spline. As cubics have 4 degrees of freedom, we need four basis functions to represent any cubic. Thus, except for the joints, we require that exactly four $B_i(t)$ functions will be non-zero at any t; these four functions form the basis for a cubic segment. A joint is at any particular t where one (or more) $B_i(t)$ functions becomes zero, and is replaced by a (or some) new $B_i(t)$ that becomes non-zero; that is, a joint is a place where one segment stops and a new segment begins.

An example of $B_i(t)$ for the uniform cubic B-spline (2.12) is shown in Figure 2.4. For any t at most four $B_i(t)$ are non-zero. Note that to form a basis for cubic polynomials one needs precisely four basis functions. Hence, a range [3,6] is shown in Figure 2.4; this range specifies where one has enough basis functions to have each segment of the spline able to span all cubic polynomials. The range [3,6] is called the *domain* of the spline and, for a spline $S(t)$, will generally be denoted as $[a_S, b_S]$ throughout this thesis.

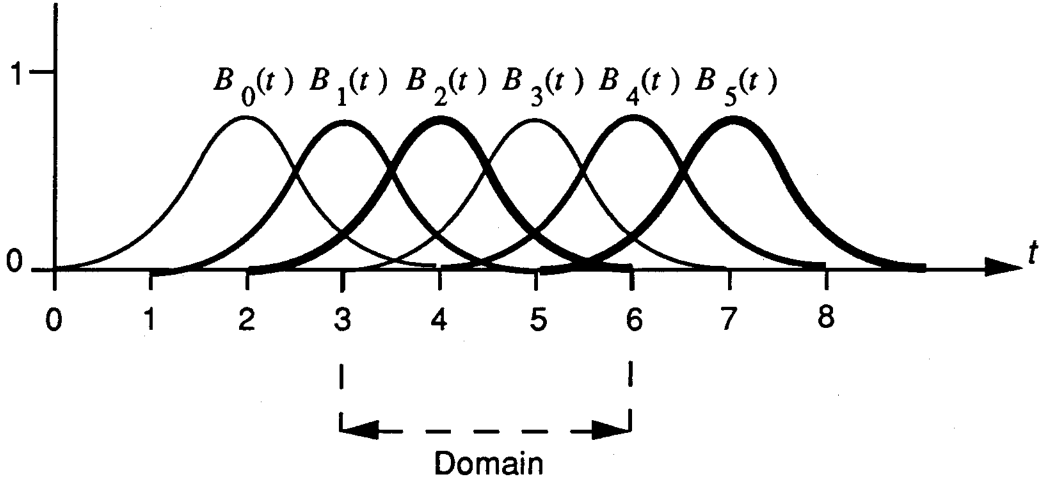

Figure 2.4 The uniform cubic B-spline basis.

Using the basis functions of Figure 2.4, one can define a three segment cubic spline by the following equation:

$$S(t) = \sum_{i=0}^{5} V_i B_i(t), \text{ where } t \in [3,6].$$

By changing the values of V_i (the control vertices), one can have each segment of the spline represent any cubic polynomial defined over a unit interval. The spline has six degrees of freedom, with each control vertex representing one degree of freedom. An arbitrary three segment cubic piecewise polynomial has twelve degrees of freedom. With cubic B-splines, however, as one has positional, first derivative, and second derivative continuity at the joints, each joint uses up three degrees of freedom. Hence, as a three segment uniform B-spline has two joints in the range $(3,6)$ (one at 4 and one at 5), continuity constraints use up six degrees of freedom. Thus six degrees of freedom are left, and these are satisfied by the control vertices.

3. Applications

3.1. The Common Element

Two application areas for which the results of this thesis are useful are cel-keyframe animation and swept surfaces. With respect to this thesis, both of these applications can be viewed in a similar manner. The common view is that one starts out with a set of *keys* and uses an *interpolation* method to calculate *inbetweens*.

One can use a variety of spline interpolation methods to compute the inbetweens. All the computer simulation done for this thesis used cubic Hermite interpolation (described in [Kochanek82] and [Kochanek84]).

3.2. Two-Dimensional Cel-Keyframe Animation

3.2.1. Why Use Cel Animation?

A *frame* is a single motionless picture (drawing). An animation (moving picture) is made by showing many frames in sequence. For film animation, 24 frames per second are required, while for video animation 30 frames per second are used. This means that for a one minute film animation 1440 frames are needed. To reduce some of the work needed, frames are usually shot in "twos" or "threes", where twos means that each picture is shot twice; shooting in twos thus requires 720 frames for a one minute film animation.

Keyframe animation divides the frames into two types: keyframes and inbetween frames. The keyframes would typically be drawn by more senior animators, while the tedious task of creating the inbetween frames would be done by the inbetweeners (or assistant animators). So, for example, our one minute animation could have 100 keyframes and 620 inbetween frames. The work of this thesis is intended to aid in the task of inbetweening by computer automation.

3.2.2. Cel Animation

Typically the drawings that the animators do in keyframe animation are done on celluloids (abbreviated as "cels", which are semi-transparent acetate sheets), hence the application is called cel keyframe animation. Keyframe animation does not have to be done on cels, and the work of this thesis never uses any aspects of cel animation. As long as the keyframe animation medium can be stored and manipulated in the computer as splines, the work of this thesis should be applicable. We mention cel animation here for two reasons: the world of Mickey Mouse (from which this thesis was born) is cel animation, and cel animation offers a concrete example of an animation method.

Briefly, cel animation is an animation technique that was developed in the early part of this century. It involves a background drawn on paper (this is the static part of the scene), and the objects that are moving are drawn on cels. The various cels are placed over the background and photographed for each individual frame. Cel animation is a technique used to reduce the amount of work needed in drawing the frames by reusing the parts of the frames; it avoids drawing parts of the scene that do not move. Furthermore, cel animation is also a way to break down the animation task into smaller, more manageable units. For example, if three characters are present in a scene, then the animation team can be broken up into three groups, each group concentrating on one of the characters (each character drawn on a separate cel). Parts of a single character may be broken down into further cels.

data for a surface of a forearm. The data consists of cross-section x-rays at four places along the forearm; thus this dataset becomes our four keys. By use of a keyframe sweeping operation the missing inbetween frames can be approximately generated. Thus, this process can lead to an approximate recreation of the forearm. Because we are interpolating, only an approximation of the forearm contained between any two keys can be recreated. The recreated forearm is only a calculated guess of what the real forearm might have looked like; the result is very much dependent on the type of interpolation done, as well as the number of keys and their positions.

Keyframes do not need to be planar, but having planar keys does simplify the understanding of the sweeping process. Planar keys allow us to think of sweeping as an operation that takes a cross-section and moves it along a path; this operation traces out the surface. Keyframe sweeping also changes the cross-section as it moves along the path, hence enabling each key cross-section to be passed through.

3.3.3. Swept Surface Rendering

Once again, consider the forearm example and how one might render the modelled surface. We started with four keys. Interpolation is used to create trajectories which define the surface. Rendering the surface can, at best, give us an approximation to the model; a video display has a fixed number of equally sized pixels, and, hence, only displays approximations to most objects. To render the surface we first create some inbetween frames by interpolation. This creates, say, 100 frames. We are left with the problem of making these 100 frames into a complete surface. Each curve in a frame can be approximated by a series of line segments (splines are usually rendered by this line segment approximation technique). Corresponding line segments in adjacent frames can also be

Thus an arm, which is in motion during a scene, may be on one cel while a leg, which is static during part of the scene, may be on another cel.

3.3. Swept Surface Creation

3.3.1. Surface Modelling

Our goal might be to design a surface; this task is referred to as creating a *model* of a surface. [Coquillart87] mentions three types of tools that a surface modeler may use: constructors, modifiers, and combiners. Briefly, *modifiers* change an existing surface (for example, by moving control points), *combiners* are a set of boolean operators to build a new surface from a set of previously created surfaces , and *constructors* are low-level utilities used to build a set of primitive objects. These primitive objects are then "combined" and "modified" to create the resulting surface. The creation of a swept surface can be thought of as belonging to the constructor class of surface creation. The use of keyframes is one technique that can be used to create a swept surface.

3.3.2. Keyframe Sweeping

Let us compare swept surface creation to 2-D cel-keyframe animation. The primary difference is swept surfaces display all their "frames" at one time while animations display one frame at a time. In essence, swept surfaces have taken away the dimension of time and replaced it with the dimension of space. One can think of an animation as a sampling technique to see cross-sections of the swept surface.

If each keyframe is a planar drawing (all objects in a frame lie in one plane) then these keyframes can be thought of as cross-sections of a surface. The inbetween frames then become an approximation to the surface between the keys. For example, suppose one has joined by lines. This technique produces a polygonal mesh to be rendered. Such a mesh can be made visually pleasing by using a smooth-shaded rendering. Examples of surfaces rendered using this technique appear in [Bartels89].

4. Constraints

A *constraint* consists of a property (or association) that exists between two curves. If a constraint is known to hold in all keyframes, we wish to have the constraint also met in each computer-generated inbetween frame.

In animation a constraint might be used as a mechanism to obey physical laws. For example, the "Mickey's nose problem" is concerned with keeping a nose connected to a head. Another physical law is that two solid objects cannot pass through each other (a person walking should always be at or above the floor).

In the realm of swept surfaces, a constraint can be thought of as a way to enforce rules as to how a cross section of the surface should be constructed.

To follow are some examples of constraints that one might like to maintain. Chapter 6 will explain which of these constraints can and cannot be maintained as a result of the technique presented in Chapter 5.

4.1. Notation

Let $\left\{ R(\bar{u}), S(\bar{v}), T(\bar{w}):\ a_R \le \bar{u} \le b_R,\ a_S \le \bar{v} \le b_S,\ a_T \le \bar{w} \le b_T \right\}$ be the splines between which we wish to maintain a constraint (or constraints). The splines $R(\bar{u})$, $S(\bar{v})$, and $T(\bar{w})$ could all refer to the same spline curve. That is, one may wish to have a constraint between one point on a spline and another point on the same spline: for example, continuity conditions at the knots, or a closed spline where $R(a_R) = R(b_R)$.

Sometimes we will phrase the constraint using a segment-wise notation. To do this we will let $\{r_i(u)\colon 0 \le u \le 1\}$ be the i^{th} segment of $R(\bar{u})$, let $\{s_j(v)\colon 0 \le v \le 1\}$ be the j^{th} segment of $S(\bar{v})$, and let $\{t_k(w)\colon 0 \le w \le 1\}$ be the k^{th} segment of $T(\bar{w})$. Also, $\hat{u}$ will denote a particular u value (in the range $[0, 1]$), and similarly $\hat{v}$ and $\hat{w}$ will denote particular v and w values respectively.

At other times it will be more convenient not to mention a particular segment. Here the spline notation $R(\bar{u})$, $S(\bar{v})$, and $T(\bar{w})$ will be used, and $\hat{u}$, $\hat{v}$, and $\hat{w}$ will be particular values on the ranges $[a_R, b_R]$, $[a_S, b_S]$, and $[a_T, b_T]$, respectively.

4.2. Point Equality

The point equality constraint can be stated as follows:

(4.1) $r_i(\hat{u}) = s_j(\hat{v})$.

Note that $i, j, \hat{u}, and\ \hat{v}$ are all fixed and independent of the keyframe we are in. That is, if we are interpolating between two keyframes and we wish to maintain a point equality constraint, then we fix the four variables ($i, j, \hat{u}, and\ \hat{v}$) and demand that equation (4.1) be satisfied by all keyframes used by the interpolation process.

Equation (4.1) can be generalized to incorporate scale and translation by constants as follows:

(4.2) $r_i(\hat{u}) = \sigma\, s_j(\hat{v}) + \tau$, where σ and τ are any constants.

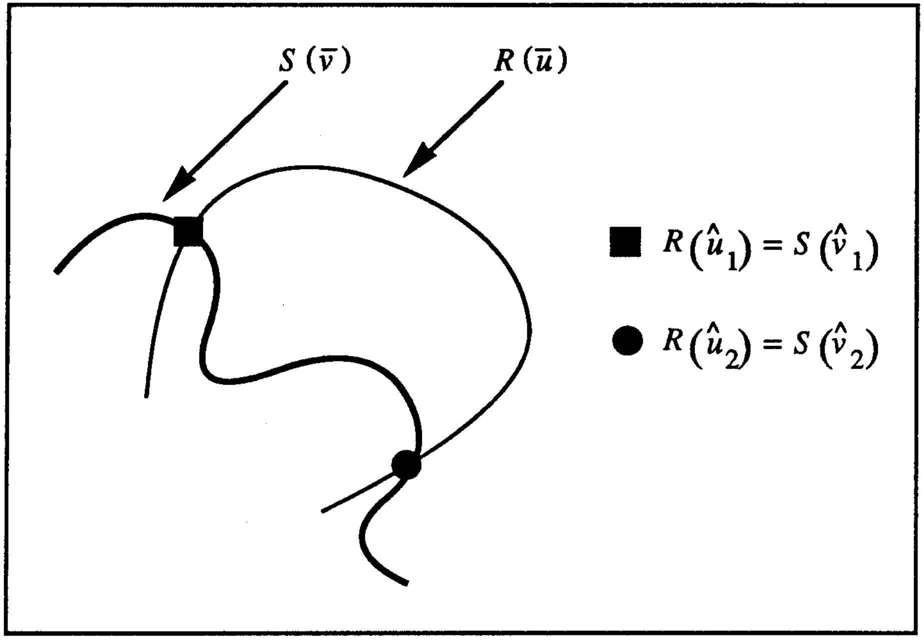
$S(\bar{v})$
$R(\bar{u})$
■ $R(\hat{u}_1) = S(\hat{v}_1)$
● $R(\hat{u}_2) = S(\hat{v}_2)$

Figure 4.1. Demonstration of Point Equality (equation (4.1)) applied twice.

4.3. Tangent (Derivative) Proportionality

Derivative equality at a point is a natural extension of (4.1). The general form of this constraint for the λ^{th} derivative is:

(4.3) $r_i^{(\lambda)}(\hat{u}) = \sigma\, s_j^{(\lambda)}(\hat{v}) + \tau$, where σ and τ are any constants.

4.4. Rotation Generalization

Equations (4.2) and (4.3) incorporate scales and translations. This can be further generalized by using a matrix notation, where a transformation matrix M_{rot} (a 3x3 matrix) can incorporate a rotation about an arbitrary point and axis of rotation (see [Foley83]). The following equation shows this more generalized form of (4.2):

(4.4)

$$\left[r_i^x(\hat{u}),\ r_i^y(\hat{u}),\ r_i^z(\hat{u})\right] = \left[\sigma^x s_j^x(\hat{v}),\ \sigma^y s_j^y(\hat{v}),\ \sigma^z s_j^z(\hat{v})\right] M_{rot} + \left[\tau^x, \tau^y, \tau^z\right]$$

4.5. Multiple Constraints

The number of constraints that can hold on a spline in a keyframe is only limited by the number of degrees of freedom of the spline. For example, a cubic polynomial can be specified by four linearly independent conditions; thus, one can only expect such a polynomial to obey at most four simultaneous constraints. Recall that one can increase the number of degrees of freedom by either increasing the degree of the polynomial, or by increasing the number of segments of the spline.

The original "Mickey's nose problem" is one example of a multiple constraint problem. Here, we wish the following two constraints to be met simultaneously:

$$\begin{aligned} r_i(\hat{u}) &= s_j(\hat{v}) \\ r_i^{(1)}(\hat{u}) &= s_j^{(1)}(\hat{v}) \end{aligned} \tag{4.5}$$

where $\hat{u}$ and $\hat{v}$ are constants that fix the nose position on the head. The splines $R(\bar{u})$ and $S(\bar{v})$ represent the nose and head respectively.

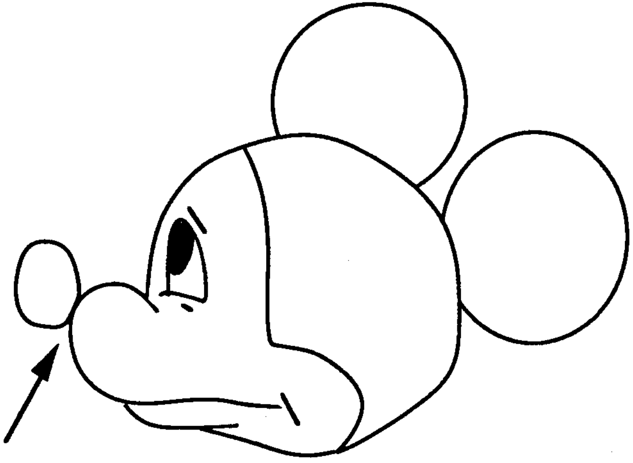

Figure 4.2. Demonstration of Mickey's nose obeying the two constraints of (4.5) where it joins onto his head.

Equation (4.5) shows multiple constraints between two curves. We can also have multiple constraints between three or more curves. A *fillet* is a good example of two constraints between three curves. The following simultaneous equations show how a fillet could be defined:

$$
\begin{aligned}
R(\hat{u}) &= T(a_T) \\
R^{(1)}(\hat{u}) &= T^{(1)}(a_T) \\
S(\hat{v}) &= T(b_T) \\
S^{(1)}(\hat{v}) &= T^{(1)}(b_T)
\end{aligned}
\qquad (4.6)
$$

In this case, entire splines (*R*,*S*, and *T*) are used rather than focusing on particular segments (r_i, s_j, and t_k). This notation is used to show that *T*, the joining spline, can consist of more than one segment. One end-point of *T* touches *R* tangentially while the other end-point of *T* touches *S* tangentially.

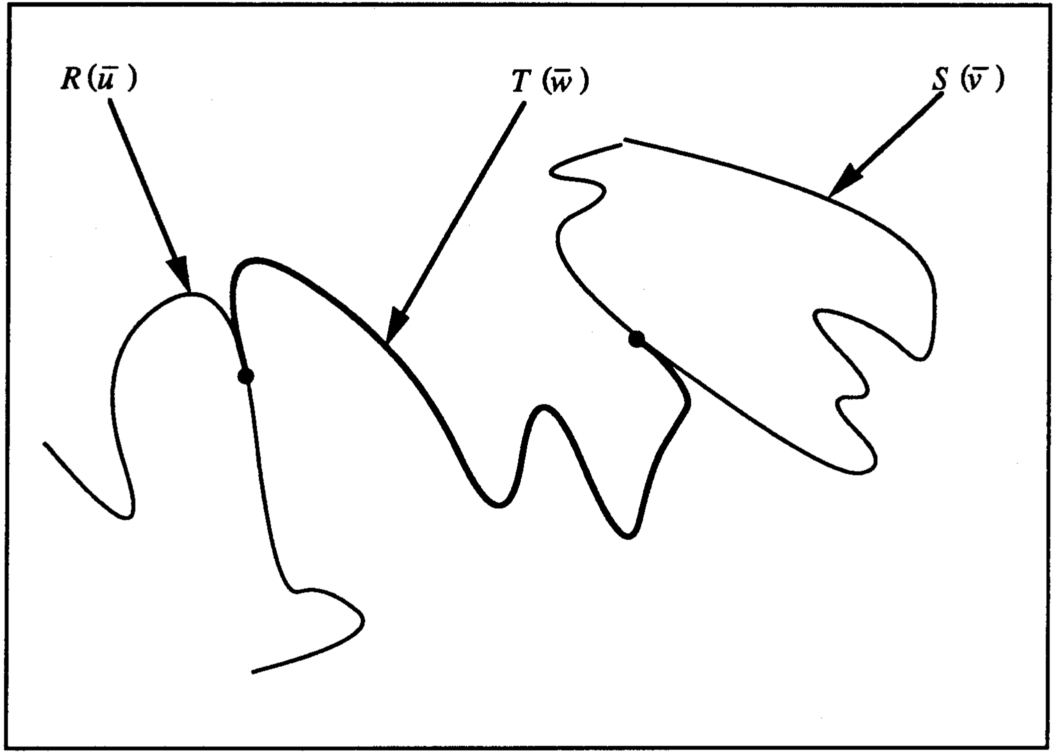

Figure 4.3. Illustration of (4.6); a fillet is shown (three splines).

4.6. Angular Associations

An angular association could be defined by the following equation:

(4.7) $\Theta\left(r_i^{(1)}(\hat{u}),\ s_j^{(1)}(\hat{v})\right) = \theta$,

where the operator $\Theta(\vec{a}, \vec{b})$ takes two vectors as parameters and returns the angle between them; θ is a constant. We will think of vectors as being position independent; that is, a vector has only a direction and a length. One can construct a tangent vector for spline R by a vector starting at the origin and ending at $\left(r_i^{(1)\,x}(\hat{u}), r_i^{(1)\,y}(\hat{u}), r_i^{(1)\,z}(\hat{u})\right)$

Equation (4.7) could be rewritten using the dot product as:

$$\arccos\left[\frac{r_i^{(1)}(\hat{u}) \bullet s_j^{(1)}(\hat{v})}{\left|r_i^{(1)}(\hat{u})\right|\left|s_j^{(1)}(\hat{v})\right|}\right] = \theta . \tag{4.8}$$

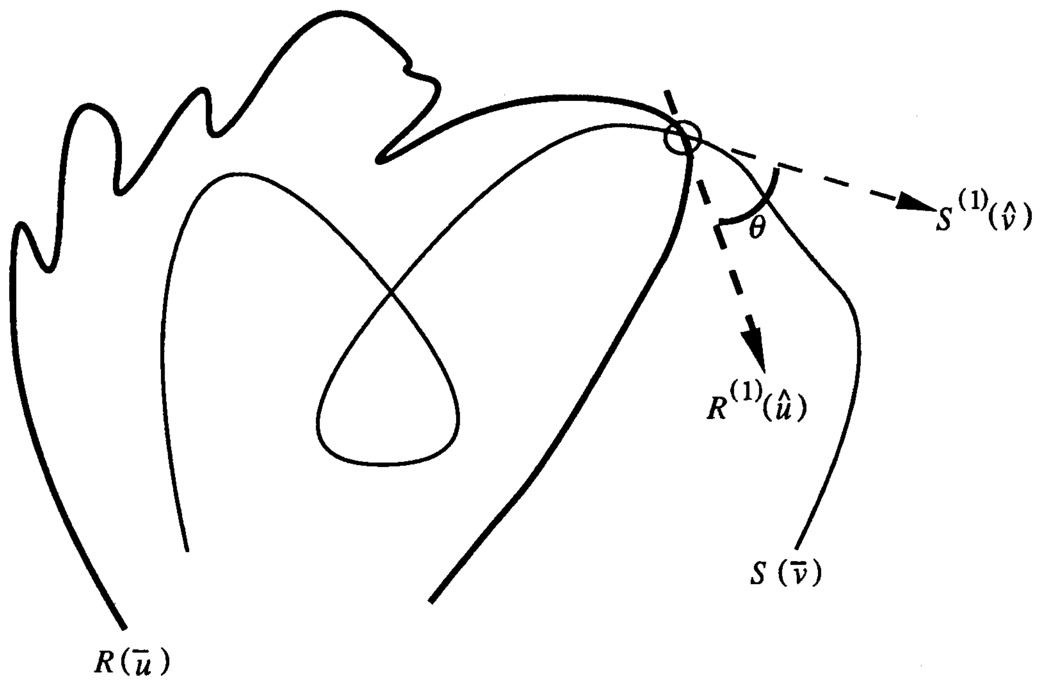

Figure 4.4 Angle Constraint (equation 4.8). The angle between two splines at a point is shown. The correct tangent magnitude is not shown. Also, it is not necessary for the two splines to intersect.

4.7. Parameter-varying Constraints

So far all the constraints have been defined in terms of constants. These constants have been the scale, translation, and rotational modelling transformations, and the constant angle θ in the previous section. One may wish to use a function instead of a constant.

For instance, with angular maintenance one may wish to maintain an equation of the form:

$$(4.9) \qquad r_i^{(1)}(\hat{u}) \bullet s_j^{(1)}(\hat{v}) = \left|r_i^{(1)}(\hat{u})\right| \left|s_j^{(1)}(\hat{v})\right| \cos(\theta(t)).$$

By using (4.9), we are asking for precise control over the angle between two curves for any position in time for an animation (or along a swept surface). One possible use of such an equation may be in modelling limb or joint movements.

4.8. Join Constraint Examples

In this section examples of point equality, tangent equality, angular maintenance, and combinations thereof will be illustrated. All the angle conditions presented in the equations in this section can be regarded as optional.

A *T-junction* (shown in Figure 4.5) has the end-point of one spline intersecting a second spline. Also, one may wish to maintain a particular angle between the two curves. The following system of equations expresses this:

$$(4.10) \qquad R(\hat{u}) = S(a_S)$$

$$\Theta\left(R^{(1)}(\hat{u}), S^{(1)}(a_S)\right) = \theta$$

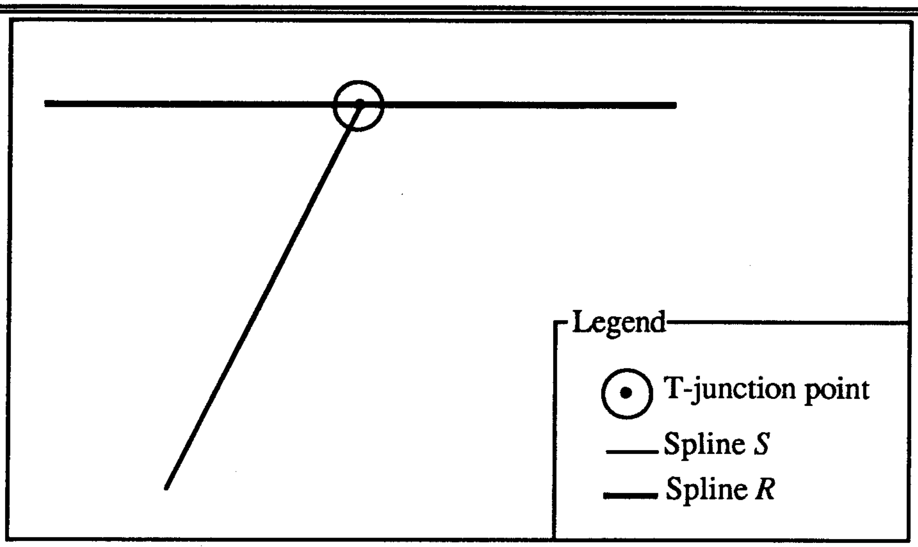

Figure 4.5 T-junction (equation 4.10)

A variation (or generalization) of the T-junction is a simple *intersection* (Figure 4.6), shown by:

(4.11) $$R(\hat{u}) = S(\hat{v})$$
$$\Theta\left(R^{(1)}(\hat{u}), S^{(1)}(\hat{v})\right) = \theta$$

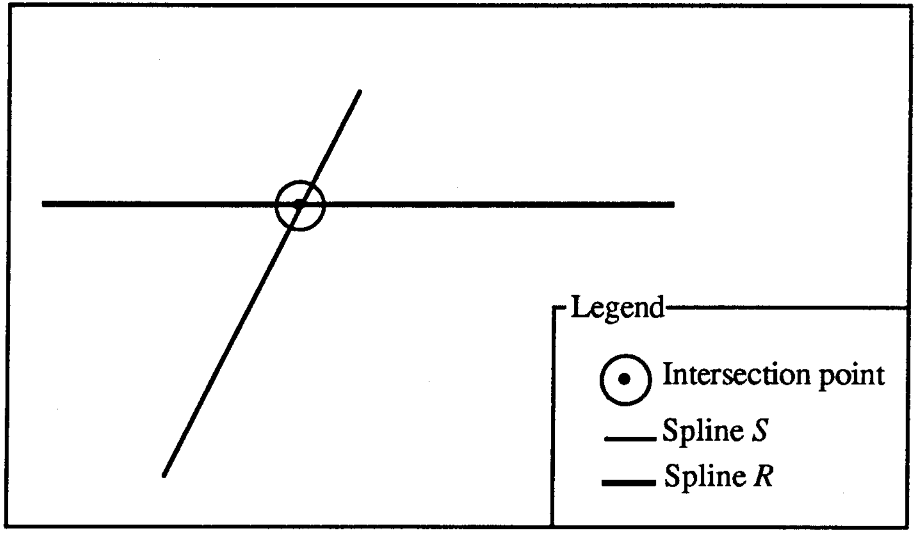

Figure 4.6 Intersection (equation 4.11)

A T-junction can be created to wrap back in such a way that both end-points intersect the same spline. This is called a *two-point loop* (see Figure 4.7). The following system of equations define this algebraically:

$$\begin{aligned} R(\hat{u}_1) &= S(a_S) \\ R(\hat{u}_2) &= S(b_S) \\ \Theta\left(R^{(1)}(\hat{u}_1), S^{(1)}(a_S)\right) &= \theta_1 \\ \Theta\left(R^{(1)}(\hat{u}_2), S^{(1)}(b_S)\right) &= \theta_2 \end{aligned} \tag{4.12}$$

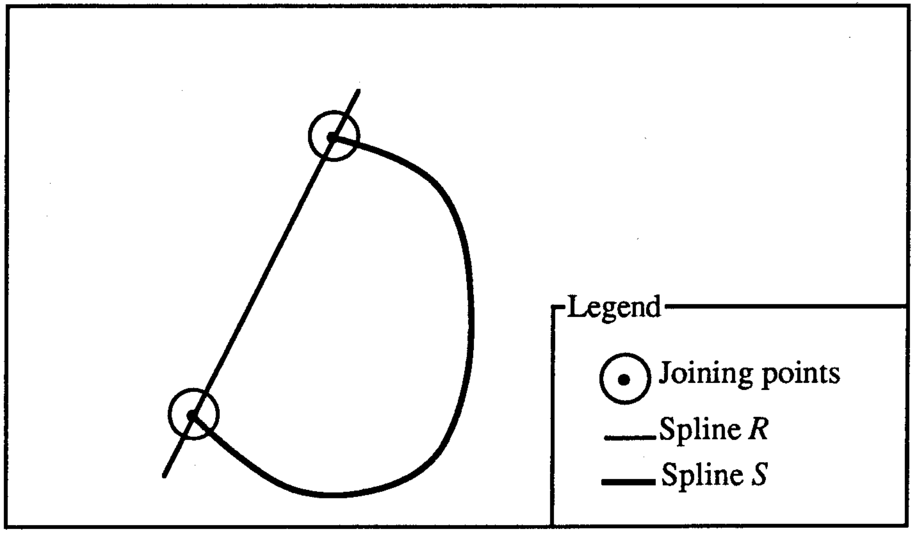

Figure 4.7 Two-point loop (equation 4.12)

An extension of the two-point loop is a *join*; in a join the two end-points of the joining spline meet two different splines (Figure 4.8). The following system of equations illustrates this:

$$R(\hat{u}) = T(a_T)$$
$$S(\hat{v}) = T(b_T) \qquad (4.13)$$
$$\Theta\left(R^{(1)}(\hat{u}), T^{(1)}(a_T)\right) = \theta_1$$
$$\Theta\left(S^{(1)}(\hat{v}), T^{(1)}(b_T)\right) = \theta_2$$

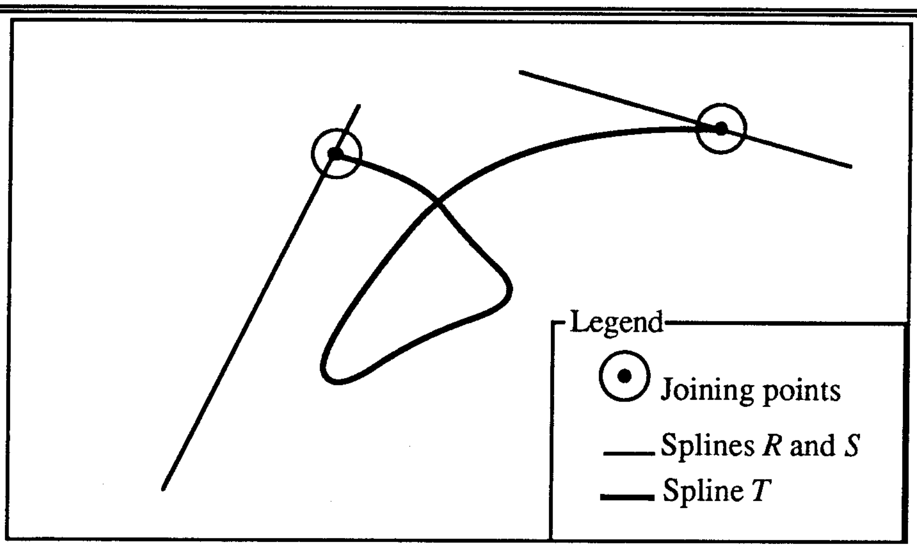

Figure 4.8 Generalized Join

A special kind of join is a *fillet*, which was shown in Section 4.5 (with Figure 4.3 and equation (4.6)).

4.9. Inclusion/Exclusion

All of the previous constraint examples involved an association (perhaps multiple associations) between one point on a spline (indicated by, say, $\hat{u}$) and one point on a second spline (say $\hat{v}$). We extend now to an association between all points on one spline and all points on a second spline. One such association is *inclusion* (Figure 4.9), where one curve is entirely contained within a second curve. Although Figure 4.9 does not show it, both curves can change shape from one key to the next. Examples of inclusion are a ball in a room, eyes on a face, and the modelling of an object inside a gas or liquid.

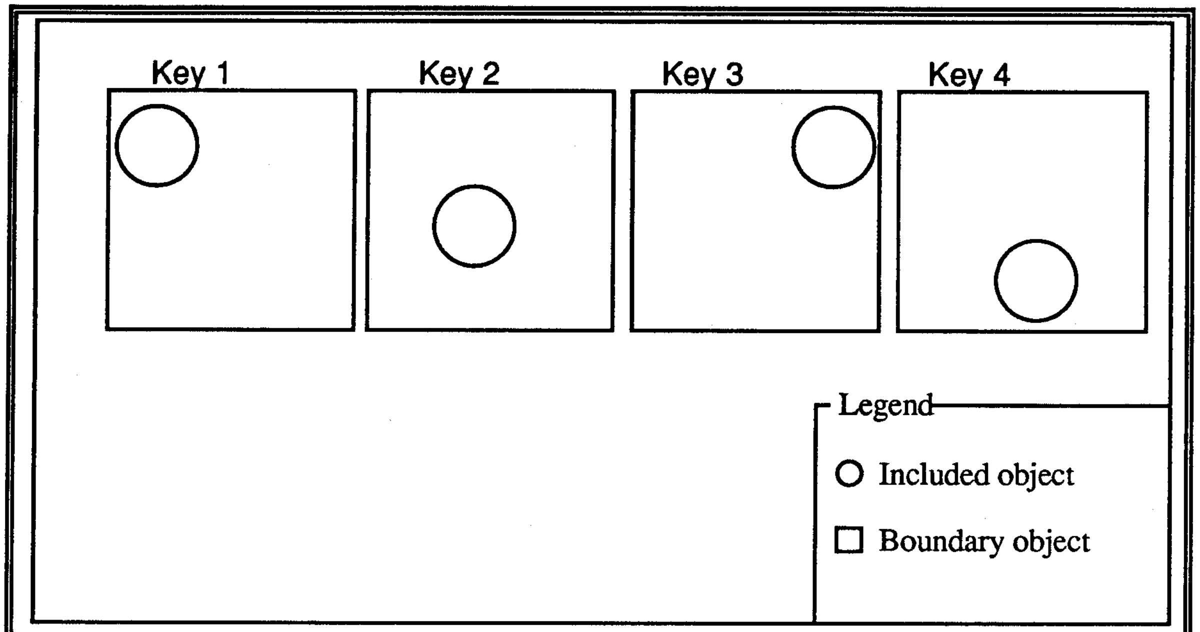

Figure 4.9 Inclusion.

The natural opposite of inclusion is *exclusion* (Figure 4.10) where one object is always entirely outside a second object. Examples of exclusion include a foot always being on or above the floor, two balls colliding, and the modelling of any two solid objects. As Figure 4.10 shows, in going from Key 3 to 4, the major problem with exclusion is what to do to avoid a collision when the paths of the two objects cross.

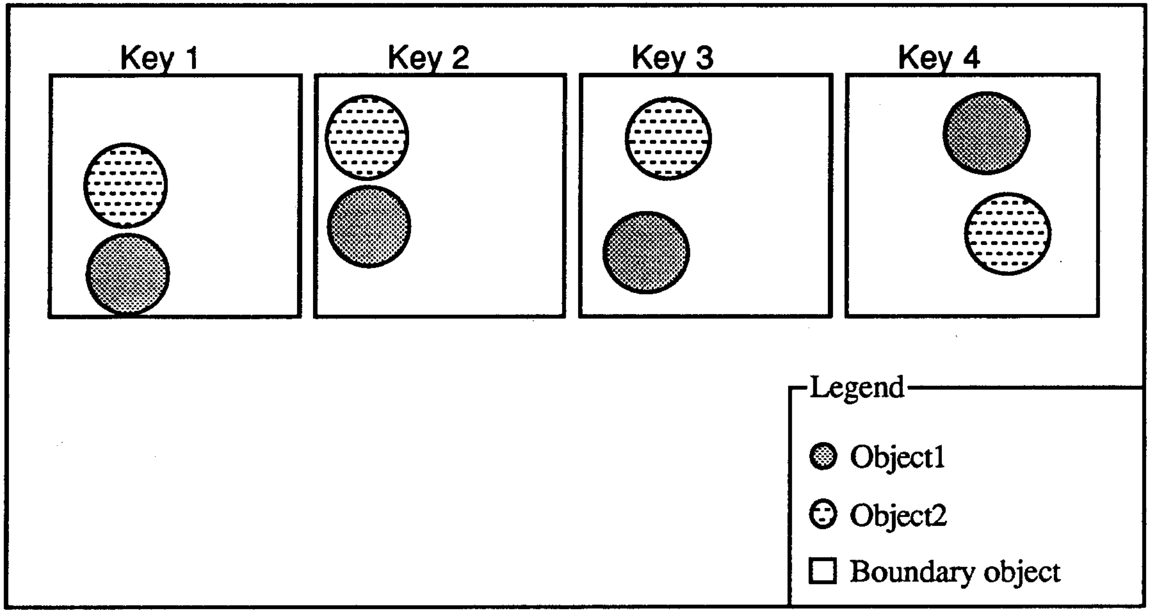

Figure 4.10 Exclusion (and inclusion)

4.10. Separation/Distance Maintenance

One can view separation/distance maintenance in one of two ways. Either one wishes to have a particular point on one spline kept at a fixed distance from a particular point on a second spline, or one wishes to have an entire spline kept at a fixed distance from a second spline. We will only discuss the first case, and leave the second case as a matter for future work (see infinite constraints in Chapter 7).

Let us define the following notation for the distance between two three-dimensional points:

$$(4.14) \qquad \Delta(P, Q) = \sqrt{(P^x - Q^x)^2 + (P^y - Q^y)^2 + (P^z - Q^z)^2}\,.$$

A distance maintenance constraint can be stated in the following way:

$$(4.15) \qquad \Delta\left(R(\hat{u}), S(\hat{v})\right) = \delta \text{, where } \delta \text{ is any constant.}$$

As in Section 4.7, one may wish to increase the flexibility by using a t-varying distance function $\delta(t)$ instead of a constant δ:

(4.16) $\Delta(R(\hat{u}), S(\hat{v})) = \delta(t).$

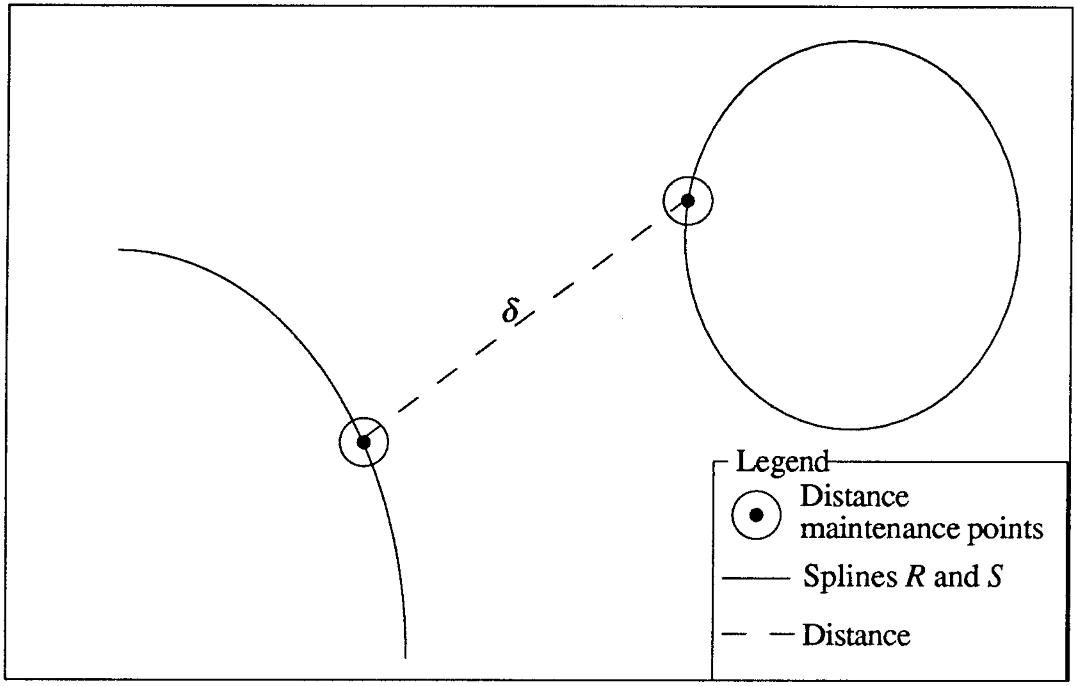

Figure 4.11 Point distance maintenance

4.11. Area/Volume Maintenance

Area maintenance is useful in modelling *squash* and *stretch* in animation. The idea of squash and stretch can most easily be demonstrated by a bouncing ball (Figure 4.12); squashing and stretching gives more life to the action. Only the most rigid of objects (such as chairs, dishes) will remain completely rigid throughout an animation. Many objects (anything living) will undergo a change in shape as they move [Thomas81].

In squashing and stretching an object, the animator has to be careful to not distort the apparent size of the object; that is, he needs to try to keep the area nearly constant. For further generality , one may wish to have the area be a function over time rather than a constant.

If we are animating solids (3D-animation), it is also beneficial to have control over the volume occupied by the object throughout the animation sequence ("volume maintenance").

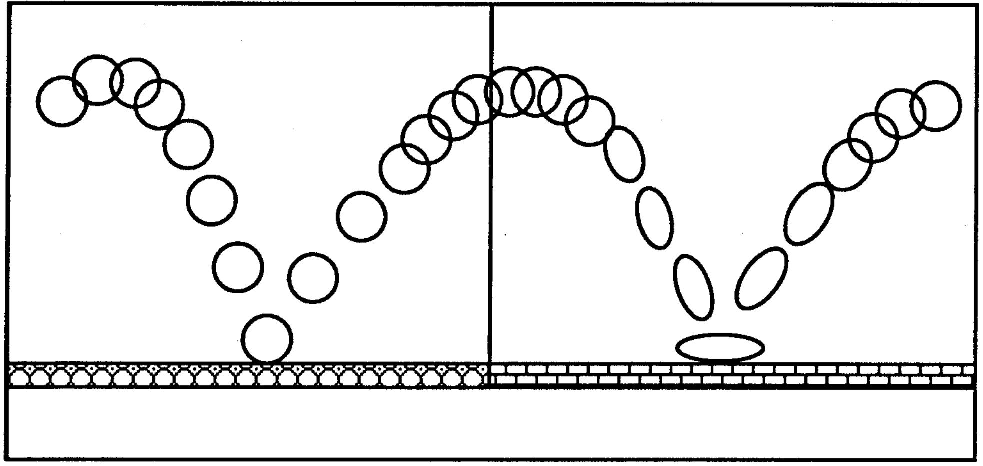

Figure 4.12 Bouncing ball shown without and with squash and stretch.

5. Results

The primary result of this thesis is, for a particular class of interpolation, a certain class of constraints will be maintained automatically throughout the interpolation process.

5.1. Development

We state our result in an evolutionary style. Conditions will evolve in the development that enable us to reach our goal. We have keyframes composed of splines, and some sort of associations (constraints) are true among the splines in any keyframe. We also know that we compute the inbetween frames by the interpolation of a spline's control vertices to produce a spline trajectory. Our goal is for the same associations to also hold for the inbetween frames.

Let (5.1a) $$R_k(u) = \sum_{i=0}^{n^r} U_{i,k} B_i(u)$$

and (5.1b) $$S_k(v) = \sum_{i=0}^{n^s} V_{i,k} C_i(v)$$

be two spline curves at the k^{th} keyframe.

The symbols n^r & n^s represent the number of control vertices, $U_{i,k}$ and $V_{i,k}$, (minus 1) for splines *R* and *S* respectively (recall that every keyframe has the same number of control vertices). The interpolation process converts these control vertices into functions of *t*. That is, one replaces the set of control vertices in (5.1a) and (5.1b) by functions $U_i(t)$ and $V_i(t)$ ($t_0 \leq t \leq t_m$). How one constructs the functions $U_i(t)$ and $V_i(t)$ depends on the interpolation method one is using.

Assume that one can write

$$(5.2)\qquad U_i(t) = \sum_{j=0}^{n^t} P_{i,j} D_j(t)$$

as the spline representing the trajectory of U_i. The symbol n^t is used to denote the number of control vertices (minus 1) in the trajectory $U_i(t)$.

Let $t \in \{t_0, t_1, \ldots, t_k, \ldots, t_m\}$ correspond to the keyframes, and notice that $m \le n^t$, as each addition of a new control vertex can add at most one segment. Then, $U_{i,k} = U_i(t_k)$.

Thus, with the following equation one can capture the spline *R* at any value of the parameter *t*:

$$(5.3a)\qquad \begin{aligned} R(u, t) &= \sum_{i=0}^{n^r} U_i(t) B_i(u) \\ &= \sum_{i=0}^{n^r} \sum_{j=0}^{n^t} P_{i,j} D_j(t) B_i(u) \end{aligned}$$

Notice that $R_k(u) = R(u, t_k)$.

S can be treated similarly:

$$(5.3b)\qquad \begin{aligned} S(v, t) &= \sum_{i=0}^{n^s} V_i(t) C_i(v) \\ &= \sum_{i=0}^{n^s} \sum_{j=0}^{n^t} Q_{i,j} D_j(t) C_i(v) \end{aligned}$$

5.2. Association Conditions

We need to be able to discuss the associations in a general sense. Hence, we'll restrict ourselves to the class of associations that are expressible as the following *mapping* *equality*:

$$F(\mathit{expression}_1) = G(\mathit{expression}_2).$$

An association that holds at the k^{th} key would be written as

(5.4) $$F(R(u,t_k)) = G(S(v,t_k)),$$

while an association that holds for all t is

$$F(R(u,t)) = G(S(v,t)).$$

Also, we require that the maps F and G be linear (why we require this condition will become evident later on in the proof). A map F is *linear* if,

$$F(R_k(u)) + F(R_l(u)) = F(R_k(u) + R_l(u)),$$

and, for any α,

$$\alpha F(R_k(u)) = F(\alpha R_k(u)),$$

These two statements are equivalent to:

(5.5) $$\sum_\mu \alpha_\mu F(R(u,t_\mu)) = F(\sum_\mu \alpha_\mu R(u,t_\mu))$$

for any set of α_μ.

Examples of associations that are expressible as a linear mapping equality are differentiation and evaluation at a point.

Our *goal* is to be able to say that:

if $F(R(u,t_k)) = G(S(v,t_k))$ for all $k = 0,1,..,m$,

then $F(R(u,t)) = G(S(v,t))$.

5.3. Proof Method Outline

From (5.3) we have

$$R(u, t) = \sum_{i=0}^{n^r} \sum_{j=0}^{n^t} P_{i,j} D_j(t) B_i(u).$$

We will try to obtain an equation of the form:

$$(5.6a) \qquad R(u,t) = \sum_{\mu} \alpha_{\mu}(t) R(u, t_{\mu}),$$

and similarly:

$$(5.6b) \qquad S(v, t) = \sum_{\mu} \alpha_{\mu}(t) S(v, t_{\mu}),$$

for then, from (5.5) and (5.6a), we can say that:

$$\begin{aligned} F(R(u,t)) &= F(\sum_{\mu} \alpha_{\mu}(t) R(u, t_{\mu})) \\ (5.7a) \qquad &= \sum_{\mu} \alpha_{\mu}(t) F(R(u, t_{\mu})), \end{aligned}$$

and, from (5.5) and (5.6b), we have:

$$\begin{aligned} G(S(v,t)) &= G(\sum_{\mu} \alpha_{\mu}(t) S(v, t_{\mu})) \\ (5.7b) \qquad &= \sum_{\mu} \alpha_{\mu}(t) G(S(v, t_{\mu})). \end{aligned}$$

Hence, if $F(R(u,t_k)) = G(S(v, t_k))$ is true for all k, then (5.7a) and (5.7b) can be combined to show that

$$F(R(u,t)) = G(S(v, t)).$$

Conditions will be placed on the interpolation method to make it possible to obtain (5.6).

5.4. Interpolation Conditions

To carry out the method of Section 5.3 it is necessary that both $R(u,t)$ (in 5.6a) and $S(v, t)$ (in 5.6b) have the same $\alpha_{\mu}(t)$ coefficient. We satisfy this constraint by imposing the condition that both splines R and S use the *same interpolation method.* Elaboration of what is meant by "same interpolation method" is now given, and results in the *conformity of trajectory* condition.

Equation (5.3) states that

$$R(u, t) = \sum_{i=0}^{n^r} \sum_{j=0}^{n^t} P_{i,j} D_j(t) B_i(u)$$

and

$$S(v,t) = \sum_{i=0}^{n^s} \sum_{j=0}^{n^t} Q_{i,j} D_j(t) C_i(v).$$

The interpolation basis $D_j(t)$, the number of control vertices for the trajectory $(n^t + 1)$, and the number of keys $(m + 1)$ should be common to both $R(u, t)$ and $S(v,t)$. It is required that the keyframes occur at the same t values for both splines R and S. We collectively call these conditions *conformity of trajectories.*

We wish to transform (5.3a) into (5.6a), and likewise (5.3b) into (5.6b). This will be done by making a substitution for $P_{i,j}$ in (5.3a), and for $Q_{i,j}$ in (5.3b). Recall from (5.2) that

$$U_i(t) = \sum_{j=0}^{n^t} P_{i,j} D_j(t).$$

Let us fix on a particular i^{th} control vertex trajectory $U_i(t)$ and try to solve for its control vertices $\{P_{i,0},\ldots, P_{i,n^t}\}$.

Let $$P_i = \begin{bmatrix} P_{i,0} \\ \vdots \\ P_{i,n^t} \end{bmatrix}, \text{ and similarly } Q_i = \begin{bmatrix} Q_{i,0} \\ \vdots \\ Q_{i,n^t} \end{bmatrix}.$$

Assume that a matrix E exists, such that $P_i = E\,U_i$, and $Q_i = E\,V_i$. The matrix E is the same for all i, as well the same for both trajectories $U_i(t)$ and $V_i(t)$; thus, we are assuming a *unique* matrix E exists. Let

$$(5.8) \qquad E = \begin{bmatrix} e_{0,0} & \cdots & e_{0,m} \\ \vdots & \vdots & \vdots \\ e_{n^t,0} & \cdots & e_{n^t,m} \end{bmatrix}.$$

Expanding $P_i = E\,U_i$ and $Q_i = E\,V_i$ gives

$$(5.9a) \quad P_{i,j} = \sum_{\lambda=0}^{m} e_{j,\lambda} U_i(t_\lambda),$$

and similarly,

$$(5.9b) \quad Q_{i,j} = \sum_{\lambda=0}^{m} e_{j,\lambda} V_i(t_\lambda).$$

Thus, we can word our assumption (requirement) to say that the interpolation process is *linearly dependent on the data.* As was stated, we require that the matrix E be *unique*; thus we will state this interpolation requirement as being *unique and linearly dependent on the data.* "Linearly dependent on the data" means that the interpolation control vertices can be written as in (5.9a) and (5.9b). That is, for fixed i, the interpolation control vertices $P_{i,j}$ and $Q_{i,j}$ are linear combinations of the keyframe positions of U_i and V_i, respectively. In other words, we are saying that $P_{i,j}$, the j^{th} control vertex of the trajectory of the i^{th} control vertex of spline *R*, is a linear combination of the keyframe positions of the i^{th} control vertex of *R*.

One way of constructing a matrix *E* is to solve a system of linear equations constructed from the interpolation basis $D_j(t)$. In the vector P_i we have $n^t + 1$ unknowns. If we substitute $n^t + 1$ values for t into (5.2), we will get $n^t + 1$ linear equations. In particular, if we use the values $t_0, t_1, .., t_m$ we get the following matrix form of this system of linear equations:

$$\text{(5.10)} \quad \begin{bmatrix} D_0(t_0) & \cdots & D_{n^t}(t_0) \\ \vdots & \vdots & \vdots \\ D_0(t_m) & \cdots & D_{n^t}(t_m) \end{bmatrix} \begin{bmatrix} P_{i,0} \\ \vdots \\ P_{i,n^t} \end{bmatrix} = \begin{bmatrix} U_i(t_0) \\ \vdots \\ U_i(t_m) \end{bmatrix}.$$

Let this system be represented by the following notation:

$$DP_i = U_i .$$

The unknowns P_i are given uniquely by:

$$\text{(5.11)} \quad P_i = D^{-1}U_i ,$$

assuming that the inverse matrix D^{-1} exists. For this to occur we at least need $n^t = m$. Thus, this presents one way of constructing the matrix *E*; that is, the matrix *D* is non-singular and $E = D^{-1}$. If *D* is non-singular, we will say that the interpolation process is *non-singular*.

In summary, the interpolation conditions are the following:

•(a) *Conformity of trajectories* is observed; this means that splines *R* and *S* use the same interpolation basis $D_j(t)$, have the same number of keyframes, and the keyframes occur at the same times. Conformity of trajectories also means that in equations (5.9a) and (5.9b) the same coefficients $e_{j,\lambda}$ are used.

and

•(b) The interpolation process is *unique and linearly dependent on the data.* This condition implies that equations (5.9a) and (5.9b) exist.

5.5. Finale

Recall our method outline in (5.7). Using this as a guide, and using equations (5.9a) and (5.9b) we can now finish the proof with the following reasoning:

$$
\begin{aligned}
F(R(u,t)) &\overset{1}{=} F(\sum_{i=0}^{n^r}\sum_{j=0}^{n^t} P_{i,j} D_j(t) B_i(u)) \\
&\overset{2}{=} F(\sum_{i=0}^{n^r}\sum_{j=0}^{n^t}\sum_{\lambda=0}^{m} e_{j,\lambda} U_i(t_\lambda) D_j(t) B_i(u)) \\
&\overset{3}{=} F(\sum_{\lambda=0}^{m}\sum_{j=0}^{n^t} e_{j,\lambda} D_j(t) \sum_{i=0}^{n^r} U_i(t_\lambda) B_i(u)) \\
&\overset{4}{=} F(\sum_{\lambda=0}^{m} \alpha_\lambda(t) \sum_{i=0}^{n^r} U_i(t_\lambda) B_i(u)) \\
&\overset{5}{=} F(\sum_{\lambda=0}^{m} \alpha_\lambda(t) R(u, t_\lambda)) \\
&\overset{6}{=} \sum_{\lambda=0}^{m} \alpha_\lambda(t) F(R(u, t_\lambda))) \\
&\overset{7}{=} \sum_{\lambda=0}^{m} \alpha_\lambda(t) G(S(v, t_\lambda)))
\end{aligned}
$$

Equality 1 is derived from (5.3). Equality 2 comes from (5.9a). Equality 4 makes the substitution of variables of

$$\alpha_\lambda(t) = \sum_{j=0}^{n^t} e_{j,\lambda} D_j(t)$$

Equality 5 is derived from (5.1a). Equality 6 is from (5.5). Equality 7 is from (5.4).

Similarly, the following reasoning can be applied to the second spline S:

$$\begin{aligned}
G(S(v,t)) &\overset{1}{=} G(\sum_{i=0}^{n^r} \sum_{j=0}^{n^t} Q_{i,j} D_j(t) C_i(v)) \\
&\overset{2}{=} G(\sum_{i=0}^{n^r} \sum_{j=0}^{n^t} \sum_{\lambda=0}^{m} e_{j,\lambda} V_i(t_\lambda) D_j(t) C_i(v)) \\
&\overset{3}{=} G(\sum_{\lambda=0}^{m} \sum_{j=0}^{n^t} e_{j,\lambda} D_j(t) \sum_{i=0}^{n^r} V_i(t_\lambda) C_i(v)) \\
&\overset{4}{=} G(\sum_{\lambda=0}^{m} \alpha_\lambda(t) \sum_{i=0}^{n^r} V_i(t_\lambda) C_i(v)) \\
&\overset{5}{=} G(\sum_{\lambda=0}^{m} \alpha_\lambda(t) S(v, t_\lambda)) \\
&\overset{6}{=} \sum_{\lambda=0}^{m} \alpha_\lambda(t) G(S(v, t_\lambda)) \\
&\overset{7}{=} \sum_{\lambda=0}^{m} \alpha_\lambda(t) F(R(u, t_\lambda)) .
\end{aligned}$$

Therefore, we have shown that, with the appropriate conditions, $F(R(u,t)) = G(S(v,t))$.

5.6. Summary

We have shown that *if*

- we are working with two splines R and S,

- and the association we are dealing with is expressible as a mapping equality as:

$$F(R(u, t)) = G(S(v, t)),$$

- and the mappings F and G are linear (5.4),
- and the interpolation process is representable as a spline and conformity of trajectories is obeyed by splines R and S,
- and the interpolation process is unique and linearly dependent on the data,
- and $F(R(u,t_k)) = G(S(v, t_k))$ for each keyframe t_k where $(0 \leq k \leq m)$,
- *then*

$F(R(u,t)) = G(S(v, t))$ is true for all $t \in [r_a, r_b]$, where $[r_a, r_b]$ is the domain for the trajectory splines $U_i(t)$ and $V_i(t)$.

5.7. Interpolation Condition Usefulness

One could argue that the conditions imposed in the previous section are too rigid to be of much use. This section will argue (by example) that the second interpolation condition is not difficult to satisfy. The first interpolation condition of conformity of trajectories is not discussed here; it is not a condition on the type of interpolation, but instead a condition that relates the interpolation method of one spline to the interpolation method of a second (associated) spline. The second interpolation condition is that "the interpolation process is unique and linearly dependent on the data"; this section will further discuss this condition.

A trivial example of an interpolation technique that obeys this interpolation condition is simple linear interpolation. The linear interpolation function between 2 points a and b is the following:

$U(t) = (1 - t)a + t\,b.$

As *t* goes from 0 to 1, the function $U(t)$ moves along the line joining a to b. The following equations express $U(t)$ as a spline (with control vertices and basis functions):

$$U(t) = \sum_{i=0}^{1} P_i D_i(t)$$
$$D_0(t) = 1 - t$$
$$D_1(t) = t$$
$$P_0 = a$$
$$P_1 = b$$

P_0 and P_1. the control vertices of the trajectory $U(t)$, are linear combinations of the two data points *a* and *b* (more precisely, they are the two data points). Hence, linear interpolation is linearly dependent on the data. By placing ourselves in the following form (similar to (5.11)) we show explicitly that the control vertices P_j are formed as a linear combination of the data:

$$P_j = \sum_{\lambda=0}^{1} e_{j,\lambda} U(t_\lambda)$$
$$E = \begin{bmatrix} e_{0,0} & e_{0,1} \\ e_{1,0} & e_{1,1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$t_0 = 0$$
$$t_1 = 1$$
$$U(0) = a$$
$$U(1) = b$$

Linear interpolation is unique since the matrix *E* is uniquely defined (that is, it is a constant that remains the same no matter what data is being interpolated). Hence, linear interpolation is an example of a process that is unique and linearly dependent on the data.

The paper [Kochanek84] discusses the use of cubic interpolating splines with local tension, continuity, and bias using the Hermite interpolation basis. This is another example of an interpolation technique that satisfies our interpolation condition. Let us concentrate on the k^{th} segment of the interpolating spline. Let $u_k(t)$ define this segment. The reason for digressing to a segment-wise view of the trajectory is two-fold. The Hermite interpolation basis has already been defined for the parameter range $[0, 1]$; this basis would need to be transformed appropriately if the segment is not on this range. Secondly, each segment has its own four control vertices; that is, no sharing of control vertices (as with B-splines) is done. Thus, we concentrate on a single segment of the trajectory for simplicity. The interpolating spline is defined as:

$$u_k(t) = \rho_k h_1(t) + \rho_{k+1} h_2(t) + DD_k h_3(t) + DS_{k+1} h_4(t)$$

where

$$DD_k = \frac{(1-\tau)(1-\chi)(1+\beta)}{2}(\rho_k - \rho_{k-1}) + \frac{(1-\tau)(1+\chi)(1-\beta)}{2}(\rho_{k+1} - \rho_k)$$

$$DS_k = \frac{(1-\tau)(1+\chi)(1+\beta)}{2}(\rho_k - \rho_{k-1}) + \frac{(1-\tau)(1-\chi)(1-\beta)}{2}(\rho_{k+1} - \rho_k)$$

ρ_k are the keyframe data points being interpolated ($u_k(t)$ interpolates between the points ρ_k and ρ_{k+1}),

$t \in [0, 1]$,

$k \in \{1, 2, \ldots, m-1\}$,

$h_i(t)$ are the Hermite cubic interpolation basis functions (Chapter 2),

and $\{\tau, \chi, \beta\}$ are tension, continuity, and bias control variables respectively. They give a control mechanism to tune the interpolating spline to one's needs.

The controls $\{\tau, \chi, \beta\}$ can be global, so that every ρ_k has the same $\{\tau, \chi, \beta\}$, or local, where each ρ_k can have a different $\{\tau_k, \chi_k, \beta_k\}$, in which case one should use the following equations:

$$DD_k = \frac{(1-\tau_k)(1-\chi_k)(1+\beta_k)}{2}(\rho_k - \rho_{k-1}) + \frac{(1-\tau_k)(1+\chi_k)(1-\beta_k)}{2}(\rho_{k+1} - \rho_k)$$

$$DS_k = \frac{(1-\tau_k)(1+\chi_k)(1+\beta_k)}{2}(\rho_k - \rho_{k-1}) + \frac{(1-\tau_k)(1-\chi_k)(1-\beta_k)}{2}(\rho_{k+1} - \rho_k)$$

From this definition it is obvious that the interpolating spline $u_k(t)$ is formed as a linear combination of the data points $\{\rho_{k-1}, \rho_k, \rho_{k+1}, \rho_{k+2}\}$. To ensure that the interpolation process is unique, we need to have the same linear combination coefficient matrix E for all the control vertex trajectories, independent of which keyframe spline control vertex we are interpolating. We can ensure this if all control vertices in any keyframe have the same $\{\tau_k, \chi_k, \beta_k\}$; that is, $\{\tau_k, \chi_k, \beta_k\}$ is independent of which control vertex trajectory we are computing. Hence, this interpolation technique can be unique and linearly dependent on the data.

One must still be careful to satisfy the interpolation condition that both splines involved in the association use the same interpolation process (conformity of trajectories). In order to satisfy this condition one requires that both splines R and S use the same interpolation basis ($h_i(t)$ in this case), and the same linear combination coefficients ($e_{j,\lambda}$ in 5.9). Recall that uniqueness means that the $e_{j,\lambda}$ coefficients are independent of which control vertex of R or S we are dealing with. Thus, for the theory of this thesis to be directly applicable, if we fix on a particular keyframe then all control vertices of both splines in this keyframe should have the same $\{\tau, \chi, \beta\}$ values. Different keyframes can, however, have different $\{\tau, \chi, \beta\}$ values (giving us local $\{\tau, \chi, \beta\}$ control).

6. Constraints Part II

6.1. Restatement of our Problem

We know that something is true, say, every day at 5 pm. Can we say that this event is true all the time? Interpolation makes a guess at what our world looks like at any time between two keys, where a key is a place where we know the state of the world. For our world of keyframes composed of splines, in the previous chapter we have determined that, under some interpolation methods, some constraints that are true at every keyframe will be be true all the time between any two keys.

6.2. Introduction

In Chapter 4, various types of constraints were shown. We will now revisit these constraints to see which ones meet the association criteria imposed in Chapter 5. A constraint (as distinct from an interpolation method) need only obey the following two preconditions:

- the constraint is expressible as a mapping equality:

(6.1a) $F(R(u, t)) = G(S(v, t))$,

- the mappings F and G are linear:

(6.1b) $\sum_{\mu} \alpha_{\mu} F(R(u,t_{\mu})) = F(\sum_{\mu} \alpha_{\mu} R(u, t_{\mu}))$.

We will express our mapping equalities such that they are applicable for all parametric t values (as equation (6.1a)). However, in showing that they are (or are not) linear, we will just consider the keyframe parametric t values (6.1b).

6.3. Constraints Obeying Criteria

6.3.1. Point Equality

Let us first consider the simple form of the point equality given in equation (4.1). The following is the mapping equality representation:

$$F(R(u,t)) = R(\hat{u},t)$$
$$G(S(v,t)) = S(\hat{v},t)$$

F is linear since:

$$\sum_{\mu}\alpha_{\mu}F(R(u,t_{\mu})) = \sum_{\mu}\alpha_{\mu}R(\hat{u},t_{\mu})$$

and $F(\sum_{\mu}\alpha_{\mu}R(u,t_{\mu})) = \sum_{\mu}\alpha_{\mu}R(\hat{u},t_{\mu})$.

Similarly, G is linear. Hence, the simple point equality obeys our criteria.

Let us now consider the generalized form of point equality in equation (4.2). We will use the following as the mapping representation:

$$F(R(u,t)) = R(\hat{u},t)$$
$$G(S(v,t)) = \sigma S(\hat{v},t) + \tau$$

F is as above, hence it is linear. G, however, is <u>not</u> linear because of the following:

$$\sum_{\mu}\alpha_{\mu}G(S(v,t_{\mu})) = \sum_{\mu}\alpha_{\mu}(\sigma S(\hat{v},t_{\mu}) + \tau) = \sigma\sum_{\mu}\alpha_{\mu}S(\hat{v},t_{\mu}) + \tau\sum_{\mu}\alpha_{\mu}$$

but $G(\sum_{\mu}\alpha_{\mu}S(v,t_{\mu})) = \sigma\sum_{\mu}\alpha_{\mu}S(\hat{v},t_{\mu}) + \tau$

However, having a mapping G of the following would be linear:

$$G(S(v,t)) = \sigma S(\hat{v},t).$$

If we instead use a modelling transformation in homogeneous coordinates for G, we can obtain a linear map. A modelling transformation can combine a scale, a translation, and a rotation about an arbitrary axis into a single 4x4 matrix (see [Foley83] for more details). Suppose G is the following map:

$$G(S(v,\ t)) = \left[S^x(\hat{v},\ t),\ S^y(\hat{v},\ t),\ S^z(\hat{v},\ t),\ S^w(\hat{v},\ t)\right] M\ ,$$

where M is a 4x4 matrix storing the modelling transformation,
and $\left[S^x(\hat{v},\ t),\ S^y(\hat{v},\ t),\ S^z(\hat{v},\ t),\ S^w(\hat{v},\ t)\right]$ represents the parametric homogeneous (x, y, z, w) functions.

This modelling transformation form of G is linear as:

$$\sum_{\mu} \alpha_{\mu} G(S(v, t_{\mu})) = \sum_{\mu} \alpha_{\mu} \left[S^x(\hat{v}, t_{\mu}), S^y(\hat{v}, t_{\mu}), S^z(\hat{v}, t_{\mu}), S^w(\hat{v}, t_{\mu})\right] M$$

$$= \left[\sum_{\mu} \alpha_{\mu} S^x(\hat{v}, t_{\mu}), \sum_{\mu} \alpha_{\mu} S^y(\hat{v}, t_{\mu}), \sum_{\mu} \alpha_{\mu} S^z(\hat{v}, t_{\mu}), \sum_{\mu} \alpha_{\mu} S^w(\hat{v}, t_{\mu})\right] M$$

and,

$$G(\sum_{\mu} \alpha_{\mu} S(v, t_{\mu}))$$

$$= \left[\sum_{\mu} \alpha_{\mu} S^x(\hat{v}, t_{\mu}), \sum_{\mu} \alpha_{\mu} S^y(\hat{v}, t_{\mu}), \sum_{\mu} \alpha_{\mu} S^z(\hat{v}, t_{\mu}), \sum_{\mu} \alpha_{\mu} S^w(\hat{v}, t_{\mu})\right] M$$

Thus, simple point equality and the general homogeneous modelling transformation form of point equality are expressible as linear maps.

6.3.2. Derivative Equality

For the simple form of derivative equality, the mapping equality representation is the following:

$$F\,(R\,(u,t)) = R^{(\gamma)}(\hat{u},t)$$
$$G\,(S\,(v,t)) = S^{(\gamma)}(\hat{v},t)$$

For the modelling transformation form, the mapping equality representation is the following:

$$F\,(R\,(u,t)) = R^{(\gamma)}(\hat{u},t)$$
$$G\,(S\,(v,t)) = \left[{S^{x}}^{(\gamma)}(\hat{v},t),\ {S^{y}}^{(\gamma)}(\hat{v},t),\ {S^{z}}^{(\gamma)}(\hat{v},t),\ {S^{w}}^{(\gamma)}(\hat{v},t)\right]M$$

Since the form of derivative equality is very similar to point equality, the details of the proof of linearity are not included here (just replace all of the spline functions of the previous point equality section with their γ^{th} derivatives). Notice that point equality is a special case of derivative equality, where $\gamma = 0$.

6.3.3. Multiple Constraints

As each constraint is independent of any other constraints, the proof of Chapter 5 is directly applicable to multiple constraints. More precisely, if constraint *i*, for $i = 1 \ldots n$, is expressible as a mapping equality involving linear maps F_i, G_i, then these *n* constraints meet the association criteria both independently as well as combined together. Thus, provided that the remaining conditions of the solution (Section 5.6) are met, all *n* constraints will be obeyed simultaneously throughout the interpolation.

6.3.4. Angular Associations

Let us now rephrase our angle maintenance constraint of Section 4.6 (equation (4.8)) to get a form that we can represent as a mapping equality. First, we require that the angle θ keeps a fixed orientation throughout all the keyframes. *Fixed orientation* means that either

- in every keyframe the angle from $R^{(1)}(\hat{u}, t_k)$ to $S^{(1)}(\hat{v}, t_k)$ in a <u>clockwise</u> sense sweeps out an angle of θ, or
- in every keyframe the angle from $R^{(1)}(\hat{u}, t_k)$ to $S^{(1)}(\hat{v}, t_k)$ in a <u>counter-clockwise</u> sense sweeps out an angle of θ.

Thus, this condition means that there exists a rotation of θ which will make $R'(\hat{u}, t_k)$ parallel to $S'(\hat{v}, t_k)$, and that this rotation is the same for all keyframes.

Secondly, we require that the following equation is true for every keyframe, where σ is any scalar,

$$\left|R'(\hat{u}, t_k)\right| = \sigma^2 \left|S'(\hat{v}, t_k)\right|.$$

If these two conditions are met, we are able to state a restricted angle maintenance constraint in the following mapping notation form:

$$F(R(u, t)) = \left[R^x(\hat{u}, t)^{(1)}, R^y(\hat{u}, t)^{(1)}, R^z(\hat{u}, t)^{(1)}\right]$$
$$G(S(v, t)) = \sigma\left[S^x(\hat{v}, t)^{(1)}, S^y(\hat{v}, t)^{(1)}, S^z(\hat{v}, t)^{(1)}\right]M_{rot}$$

M_{rot} is the 3 by 3 rotation matrix which rotates spline *S* by an angle θ to make the tangent vector $\left[S^x(\hat{v}, t)^{(1)}, S^y(\hat{v}, t)^{(1)}, S^z(\hat{v}, t)^{(1)}\right]$ parallel to $\left[R^x(\hat{u}, t)^{(1)}, R^y(\hat{u}, t)^{(1)}, R^z(\hat{u}, t)^{(1)}\right]$. The scale factor σ is then applied to make the two tangents equal in magnitude; the result of maps *F* and *G* are vectors, and two vectors are equal if they have the same magnitude and direction (that is, each $\{x, y, z\}$ component of the two vectors is equal). Previously (Sections 6.3.1 and 6.3.2) we showed that the maps F and G are linear.

Thus, we have reduced angle maintenance to a modelling transformation; however, it has produced a restricted form of angle maintenance. Going to this restricted form of angle maintenance was necessary to be able to represent angle maintenance as a mapping equality involving linear maps.

6.3.5. Join Examples

As T-junction, intersection, 2-point loop, join, and fillet are all examples of multiple constraints, they will satisfy the association criteria as long as each individual constraint satisfies the criteria.

Recall that in equations (4.10)-(4.13) the angle maintenance constraint was optional. If one removes this constraint, the remaining constraints are simple forms of point and tangent equality constraints. From Section 6.3.1 and 6.3.2 point and tangent equality constraints satisfy our association criteria. Hence, these five types of joins satisfy our criteria provided that angle maintenance is not used.

If one wishes to use the angle maintenance constraint, as long as the restricted form of angle maintenance discussed in the previous section is used, then these joins will also satisfy the association criteria.

6.4. Constraints Not Obeying the Association Criteria

The primary problem with the constraints mentioned in this section is that they are not expressible as a linear mapping equality (referred to as the association conditions); some general forms of the constraints in Section 6.3 are also of this class. However, the constraints mentioned in this section still might have a variation or restricted form which does obey the association conditions. Recall that the general form of point equality as well as angle maintenance did not seem to obey the association criteria. But a variation or restricted form was possible that did obey the criteria; the same might be possible with the constraints mentioned here. Thus, figuring out how the following constraints can be obeyed throughout the inbetweening is left for future work. The motivation one would have to solve the following constraint maintenance problems depends very much on one's needs. Suggestions on how one might try to obey the constraints are given in the following sections.

6.4.1. Parameter-Varying Constraints

To understand the problem of maintenance of parameter-varying constraints, we can gain an insight from the remark at the beginning of this chapter (Section 6.1). If we only know the keyframes, we cannot possibly know how the constraint behaves between the keys. The whole approach to keyframe inbetweening taken by this thesis is that we use a generic interpolation method and are given a set of keyframes which we interpolate to generate the inbetweens. Thus, only placing the *t*-varying constraint information into the keyframe spline positions is not enough to be able to preserve the *t*-varying constraint function. Somehow the interpolation method must make use of knowledge about the *t*-varying function.

Whether or not an interpolation method exists that both obeys the interpolation process criteria of the solution and allows itself to be driven by a *t*-varying constraint function is left as an open problem. In addition, one might require that such an interpolation method allows for simultaneous multiple constraint functions.

6.4.2. Inclusion/Exclusion

The first step is to try to get an algebraic representation for this property. As such a representation does not exist yet, not much can be said about inclusion/exclusion. One suggestion is to consider inequality-based maps such as the following:

$$F(expression_1) \geq G(expression_2).$$

Note also that inclusion/exclusion is an infinite constraint problem (we wish the constraint to hold between all points on one spline and all points on a second spline). This could be handled by the following notation:

$$F(R(u,t)) \geq G(S(u,t)).$$

The parameters *u* and *t* are common to both *R* and *S*. By using *u* rather than $\hat{u}$ we show the infinite nature of the constraint; *u* is a variable whereas $\hat{u}$, specified by the user, fixes our attention on a particular *u* value.

6.4.3. Separation/Distance Maintenance

A very restrictive form of distance maintenance can be achieved if it is represented as a modelling transformation. Suppose we wish to maintain the distance between two points. The following equation shows this constraint in the notation of Section 4.10:

$$\Delta(R(\hat{u},t),S(\hat{v},t)) = \delta \; .$$

If we concentrate on the keyframes, the following equation must hold:

$$R(\hat{u}, t_\mu) = S(\hat{v}, t_\mu) + \tau_\mu .$$

If $\tau_0 = \tau_1 = \ldots = \tau_{m-1} = \tau_m$, then let $\tau = \tau_\mu$.

This gives the following translational form of point equality:

$$R(\hat{u},t) = S(\hat{v},t) + \tau .$$

Notice that if this equation is true, then it must also be true that:

$$\Delta(R(\hat{u},t), S(\hat{v},t)) = \delta \ .$$

That is, if the distance between two points can be modelled as a translation that is the same for all keyframes, then this restricted form of distance maintenance becomes the same as point equality with translation. Hence, (from Section 6.3.1) this restricted form of distance maintenance obeys the association criteria.

Recall that in Figure 1.2 tangency and point equality constraints were satisfied but size was not. If we could maintain distance, we would then be able to maintain, for instance, the radius of a circle (and hence size) throughout the interpolation.

6.4.4. Area/Volume Maintenance

Based on the following statement from [Spivak80] one would think that area maintenance might be possible through the use of integrals: "The integral $\int_a^b f$ is also called the *area* of $R(f, a, b)$ when $f(x) \geq 0$ for all x in $[a, b]$." $R(f, a, b)$ refers to the region between $f(x)$ and the horizontal axis. Integration should behave as a linear map in much the same way as differentiation did.

However, in Figure 1.2 it was shown that the size (hence area) of a semi-circle is clearly not maintained. In this figure, point and tangency are maintained; note that one key is created from the previous key by applying a rotation. Under this rotation, even though each key maintains the area of the closed splines, the area of the closed splines shrinks and expands throughout the inbetweening process.

Applying integration as a means of obtaining a restricted form of area maintenance is left as an open problem. Insight gained from an area maintenance solution should also be applicable to volume maintenance.

6.5. Summary

This chapter has given examples, building on Chapter 4, of associations that obey the association conditions established in Chapter 5. We have also given examples of associations that, at least on the surface, do not appear to obey the association conditions. These associations not obeying the criteria have brought to the surface many open problems to be solved.

7. Future Work

The material presented in this thesis "solves" the specific Mickey's Nose problem, which uses tangency and point equality constraints, but in trying to solve the generalized MN problem numerous new problems are created. Presented in the following sections are areas that work can be done to further extend the boundaries of our results.

By way of example, Chapter 6 already illustrated some desirable constraints that are not maintained as a result of our work. These included *t*-varying constraints, inclusion/exclusion, distance, and area/volume maintenance.

7.1. Avoidance versus Detection

One has two choices in trying to maintain a constraint throughout the interpolation. One choice is *avoidance*, which is what we have done for this thesis. Avoidance means that we have a set of conditions which will guarantee that the constraint will be met automatically. That is, we proved that we automatically maintain the constraint; no checking is needed for the individual inbetween frames to see that they actually maintain the constraint, because we know that they do. We avoid any chance that the inbetween frames could not obey the constraint.

The second choice one has to maintain a constraint is *detection*. Detection means that we check the inbetween frames to see if the constraint is maintained. If the constraint is not maintained, then we do something to the frame to force the constraint. This detection technique could be used to solve the types of constraints not maintained by our avoidance strategy. However, we require algorithms to detect loss of the constraint and to do something to the frame to force the constraint. This "something" could involve solving equations (perhaps non-linear) and repositioning some of the control vertices. In simulations done for this thesis, such a technique was used in ensuring that the keyframes maintain a constraint.

7.2. Non-exact (Approximate) Constraints

In animation, for example, exact matching of constraints may not be necessary. What is necessary is that a constraint is approximately obeyed. With squash and stretch, for example, one just needs to maintain area approximately to aid in realism. Thus, it seems that approximate constraints could be useful. For example, if in all the keyframes the area within a closed spline is a constant A, is there an ε such that the area within the spline in all the inbetween frames is $A \pm \varepsilon$? If there is such a bound on the area "error" ε, one may wish to know how large ε is. A study of such error-bounded constraints could prove very useful.

7.3. Infinite Constraints

The infinite constraint problem is to say that "every point between $[a_R, b_R]$ on spline R is associated with every point between $[a_S, b_S]$ on spline S in the following way ...", or that "each point between $[a_R, b_R]$ on spline R is associated with a point between $[a_S, b_S]$ on spline S in the following way ..., and vice versa". Inclusion/exclusion (Section 6.4.2) is one infinite constraint type problem. Generalized distance maintenance (where we wish to keep one object at a "fixed distance" from another object) is another example of such a problem. Infinite constraints could even prove useful for more general point and tangency maintenance problems. For example, suppose one wishes not only to keep the nose tangent at one point to Mickey's snout, but to keep it attached over a range of points, which would appear more realistic. When we build a larger object out of smaller pieces, to aid in realism, the places where the smaller pieces join each other will usually be over a range of points.

The infinite constraint problem seems like a natural extension (or a special case) of multiple constraints, and hence one would hope that this thesis work solves this type of association. One problem with infinite constraints, however, is that we have said that the number of degrees of freedom present in the spline limit the number of simultaneous constraints that can be met. Hence, further work needs to be done to either prove or disprove that this thesis work can be applied to the infinite constraint problem.

7.4. The “Real” Mickey's Nose Problem

The Mickey Nose problem defined in Chapter 1 is rather a misnomer. If one looks at some of the drawings of Mickey and other Walt Disney characters in [Thomas81] one will rarely (if ever) see a drawing of Mickey as in Figure 4.2. One has to be looking at an exact side profile of the character to have a nose appear tangent in the two-dimensional drawing. More often, the character will appear as in Figure 9.1. Thus, further work should be done to solve the real Mickey nose problem which is to ensure that the nose always maintains some contact with the snout outline. In other words, the intelligence of the animator should be taken into account to come up with some set of rules as to how a frame should look. Avoidance or detection can then be utilized to maintain this set of rules.

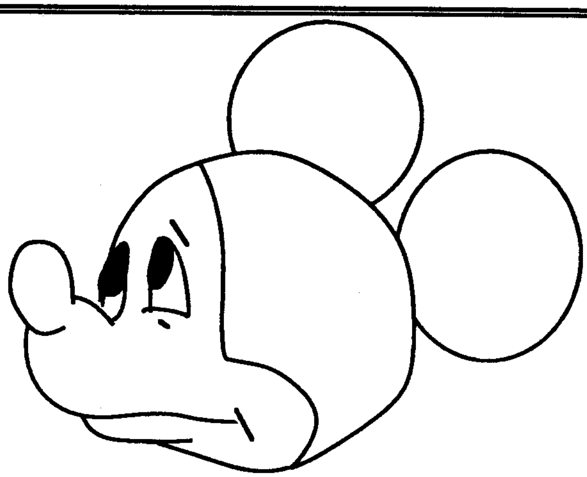

Figure 7.1 A more realistic Mickey.(from [Thomas81])

7.5. Dimensionality

A line is one-dimensional. A plane is two-dimensional. A real-life object is three-dimensional. A spline curve $S(u)$, like a line, has a single dimension. The parameter u takes one along the curve; we can either go forward, by increasing u, or backward,by decreasing u, along the curve. A spline surface $S(u, t)$, like a plane, is two-dimensional; it has two parameters (u and t) to control where on the surface one will be. A spline solid $S(u, v, t)$ is three-dimensional. A spline solid animation, like life, $S(u,v , s, t)$ is four-dimensional.

The work of this thesis has taken a collection of spline curves $S_k(u)$, and, through interpolation, created a spline surface $S(u, t)$. Thus, we have worked at the lowest level on the dimension ladder, which we call working in the *(1-2)-dimensional realm.* Left for future work is the idea of going to higher dimensions. That is, suppose we start with a collection of spline surfaces $S_k(u,v)$, and use interpolation to create a spline solid $S(u, v, t)$ (we call this working in the *(2-3)-dimensional realm*). Can we say that our results of Chapter 5 are still valid? If yes, then suppose we start with a collection of spline solids $S_k(u,v\ , s)$, and use interpolation to create a moving spline solid $S(u,\ v,\ s,t\ )$ (that is, the *(3-4)-dimensional realm*); are our results still applicable?

Consider Section 5.5 which contains the essence of our proof. Moving the proof up one dimension would require a reasoning like the following:

$$
\begin{aligned}
F(R(u, v, t)) &\overset{1}{=} F(\sum_{i=0}^{n^r} \sum_{j=0}^{n^t} P_{i,j} D_j(t) B_i(u,v\ )) \\
&\overset{2}{=} F(\sum_{i=0}^{n^r} \sum_{j=0}^{n^t} \sum_{\lambda=0}^{m} e_{j,\lambda} U_i(t_\lambda) D_j(t) B_i(u, v)) \\
&\overset{3}{=} F(\sum_{\lambda=0}^{m} \sum_{j=0}^{n^t} e_{j,\lambda} D_j(t) \sum_{i=0}^{n^r} U_i(t_\lambda) B_i(u, v)) \\
&\overset{4}{=} F(\sum_{\lambda=0}^{m} \alpha_\lambda(t) \sum_{i=0}^{n^r} U_i(t_\lambda) B_i(u, v)) \\
&\overset{5}{=} F(\sum_{\lambda=0}^{m} \alpha_\lambda(t)\, R(u, v, t_\lambda)) \\
&\overset{6}{=} \sum_{\lambda=0}^{m} \alpha_\lambda(t) F\ (R(u,v\ , t_\lambda))) \\
&\overset{7}{=} \sum_{\lambda=0}^{m} \alpha_\lambda(t) G(S(\upsilon,\varpi, t_\lambda)))
\end{aligned}
$$

Hence, we have reason to believe that the results of Chapter 5 should be directly applicable to higher dimensions. Formally proving this hypothesis is left as an exercise for future work.

A main part of such future work would require finding constraints that are well-defined for these higher dimensions. Recall the area maintenance constraint we mentioned in Chapters 4 and 6; this constraint really belongs in the (2-3)-dimensional realm , as a surface has an area, whereas a curve does not. Likewise inclusion/exclusion, which dealt with closed curves, could be considered a candidate for the (2-3)-dimensional realm. Maintaining a volume, which is an attribute of a solid, belongs to the (3-4)-dimensional realm. Normal vector maintenance (not previously mentioned) is also an ideal candidate for maintenance in the (2-3)-dimensional realm.

References

[Bartels87]

R.H. Bartels, J.C. Beatty, and B.A. Barsky, *An Introduction to Splines for use in Computer Graphics and Geometric Modelling*, Morgan Kaufmann Publishers, California, 1987.

[Bartels89]

Richard H. Bartels and Ronald T. Hardock, "Curve-to-Curve Associations in Spline-Based Inbetweening and Sweeping", *Computer Graphics* (SIGGRAPH '89 proceedings), Volume 23, number 3, July 1989.

[Burtnyk76]

N. Burtnyk and M. Wein, "Interactive Skeleton Techniques for Enhancing Motion Dynamics in Key Frame Animation", *Communications of the ACM*, Volume 19, number 10, October 1976.

[Coleman73]

A.J. Coleman et al., *Algebra (Elements of Modern Mathematics)*, Gage Educational Publishing, Toronto, 1973.

[Coquillart87]

Sabine Coquillart, "A Control-Point-Based Sweeping Technique", *IEEE Computer Graphics and Applications*, November 1987.

[Farin88]

Gerald Farin, *Curves and Surfaces for Computer Aided Geometric Design: A practical guide*, Academic Press, 1988.

[Foley83]

James D. Foley and Andries Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Welsey Publishing, 1983.

[Hardtke87]

Ines Hardtke, *Kinetics For Key Frame Interpolation*, M.Math Thesis, University of Waterloo, 1987.

[Jasmin89]

Pierre Jasmin, "Character Animation and Related Topics: A Survey of Ideas and Techniques", SIGGRAPH '89 Tutorial Notes: *;Introduction to Computer Animation*, August 1989.

[Kochanek82]

D.H.U. Kochanek, R. Bartels, and K.S. Booth, *A Computer System for Smooth Keyframe Animation*, CS-82-42, University of Waterloo, December 1982.

[Kochanek84]

D.H.U. Kochanek and R.H. Bartels, "Interpolating Splines with Local Tension, Continuity, and Bias Control", *Computer Graphics* (SIGGRAPH '84 proceedings), Volume 18, number 3, 1984.

[Lozano-Pérez79]

Tomás Lozano-Pérez and Michael A. Wesley, "An Algorithm for Planning Collision-

Free Paths Among Polyhedral Obstacles", *Communications of the ACM*, Volume 22, number 10, October 1979.

[Reeves81]

William T. Reeves, "Inbetweening for Computer Animation Utilizing Moving Point Constraints", *Computer Graphics* (SIGGRAPH '81 proceedings), Volume 15, number 3, August 1981.

[Rosebush87]

Judson Rosebush, Editor, SIGGRAPH '87 Tutorial Notes: *Advanced Computer Animation*, August 1989.

[Rosebush89]

Judson Rosebush, Editor, SIGGRAPH '89 Tutorial Notes: *Introduction to Computer Animation*, August 1989.

[Spivak80]

Michael Spivak, *Calculus*, second edition, Publish or Perish Inc., Berkeley, California, 1980.

[Steketee85]

Scott N. Steketee and Norman I. Badler, "Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control", *Computer Graphics* (SIGGRAPH '85 proceedings), Volume 19, number 3, 1985.

[Strang80]

Gilbert Strang, *Linear Algebra and Its Applications,* Academic Press, New York, 1980.

[Thalmann88]

Daniel Thalmann and Nadia Magnenat-Thalmann, *Tutorial on Computer Animation*, Graphics Interface '88, Edmonton, 1988.

[Thomas81]

Frank Thomas and Ollie Johnston, *Disney Animation: The Illusion of Life*, Abbeville Press, New York, 1981.

[Thomas86]

S.W. Thomas, "Scattered Thoughts on B-Splines", SIGGRAPH '87 Tutorial Notes: *Advanced Topics in Solid Modelling*, July, 1987.

Appendix A: Variable and Notation Usage

In our equations we have attempted to remain consistent in our choice of variable names; however, some inconsistencies will occur among some of the more locally used symbols. Presented here in chronological order is an outline of the variable and notation usage. The purpose of this appendix is to give further insight and understanding for the many equations and symbols appearing in this thesis. Cross referencing with (see *Chapter.Section*) notation is done for Chapters 2 and 4, but not for Chapters 5 and 6 as it would be too cumbersome.

Section	Variable Name	Usage Description
2.1	$f(x)$, $g(x)$	polynomials,
	a_i , x^i	coefficients and basis used in polynomial definition,
	d	degree,
	x,y , z	cartesian coordinates,
	t	parameter in defining a parametric function,
	C^k	used in defining continuity at a joint,
	r.........................	number of knots at a joint.
2.2	d	(see 2.1).
2.3	P_i	a point in key i,
	i	keyframe index,
	d	(see 2.1),
	n	number of segments in a spline,

	$P(t)$	interpolating function between P_i and P_{i+1},
	t	(see 2.1); in the range $[0,1]$,
	$h_j(t)$	the cubic hermite basis where $j = \{0, 1, 2, 3\}$,
	D_i	coefficient of the Catmull-Rom spline; also the
		derivative of $P(t)$ at the point P_i,
	$P'(t)$	the derivative of $P(t)$,
	m	we have $m + 1$ keys; the interpolating spline goes
		from key 0 to key *m*.
2.4	x, y, z	(see 2.1),
	$f(x)$	(see 2.1),
	u, t	(as 2.1), parameters for a parametric spline curve or
		surface,
	$[a, b]$	range of a parameter of a function,
	r	radius of a circle,
	$\{f_x(t), f_y(t), f_z(t)\}$	cartesian coordinate piecewise polynomials,
	$S(t)$	a spline curve in parametric form,
	$S_i(t)$	the i^{th} spline segment,
	t_i	the parameter value of the i^{th} knot of $S(t)$,
	i	spline segment index,
	n	(see 2.3),
	$S(u,t)$	a surface in parametric form,
	$\{f_x(u, t), \ldots\}$	two parameter cartesian coordinate polynomials.

2.5	$B_j(t)$	a basis for polynomials of degree *d*,
	d	(see 2.1),
	t	(see 2.1),
	j	basis function index,
	$f(t)$	general parametric polynomial,
	a_j	polynomial coefficient,
	P^d	set of all polynomials of degree *d*,
	i	(see 2.4),
	$(x_i(t),\ y_i(t),\ z_i(t))$	the cartesian polynomials of $S_i(t)$,
	t_i	(see 2.4),
	d_i	the degree of $S_i(t)$,
	$\{V^x_{i,j}, V^y_{i,j}, V^z_{i,j}\}$	the j^{th} control vertex of $S_i(t)$,
	$B_{i,j}(t)$	for fixed *i*, $B_{i,j}(t)$ is the basis for $S_i(t)$,
	$\bar{t}$	normalized *t*, so that $\bar{t} \in [0,1)$,
	$\binom{d}{j}$	the binomial coefficient $= \frac{d!}{(d-j)!\,j!}$,
	$B_i(t)$	a multiple segment basis.
2.6	$S_i(t)$	(see 2.4),
	i	(see 2.4),
	d	(see 2.1),
	t	(see 2.1),
	B_i	basis vector for $S_i(t)$,
	V_i	matrix of all the control vertices for $S_i(t)$,
	$\{V^x_{i,j}, V^y_{i,j}, V^z_{i,j}\}$	(see 2.5),

	$B_{i,j}(t)$	(see 2.5),
	j	(see 2.5),
	$b_{i,j,k}$	for fixed *i,j*, $b_{i,j,k}$ are the coefficients to represent $B_{i,j}(t)$ using the power basis,
	k	index for $b_{i,j,k}$,
	$x_i(t)$	(see 2.5),
	T	power basis vector,
	$\bar{B}_i$	matrix of $b_{i,j,k}$.
2.7	$S(t)$	(see 2.4),
	t	(see 2.1),
	n	(see 2.3),
	i	(see 2.4),
	V_i	control vertices for a multiple segment spline,
	$B_i(t)$	(see 2.5),
	t_i	(see 2.4),
	$\hat{t}$	a particular *t* value,
	$[a_S,b_S]$	domain of a spline *S*.
4.1	$R(\bar{u}), S(\bar{v}), T(\bar{w})$	spline curves,
	$\bar{u}, \bar{v}, \bar{w}$	parameters that traverse their entire splines,
	$[a_R,b_R], [a_S,b_S]$, & $[a_T,b_T]$	(see 2.7), the domains of their respective splines,
	u, v, w	parameters that traverse a single spline segment,

	$r_i(u)$	the i^{th} segment of $R(\bar{u})$,
	$s_j(v)$	the j^{th} segment of $S(\bar{v})$,
	$t_k(w)$	the k^{th} segment of $T(\bar{w})$,
	i, j, k	segment indices for their respective splines,
	$\hat{u}$, $\hat{v}$, and $\hat{w}$	particular u,v , w or $\bar{u}$, $\bar{v}$, $\bar{w}$ values.
4.2	$r_i(\hat{u})$ and $s_j(\hat{v})$	(see 4.1),
	σ	a scale transformation constant (or vector),
	τ	a translation transformation constant (or vector).
4.3	λ	which derivative we are taking,
	$r_i^{(\lambda)}(\hat{u})$, $s_j^{(\lambda)}(\hat{v})$	the λ^{th} derivative of $r_i(\hat{u})$ or $s_j(\hat{v})$,
	σ, τ	(see 4.2).
4.4	M_{rot}	3x3 rotation matrix,
	$\left[r_i^x(\hat{u}), r_i^y(\hat{u}), r_i^z(\hat{u})\right]$	a vector of the x, y, and z components of $r_i(\hat{u})$,
	$\left[s_j^x(\hat{v}), s_j^y(\hat{v}), s_j^z(\hat{v})\right]$	a vector of the x, y, and z components of $s_j(\hat{v})$,
	$[\sigma^x, \sigma^y, \sigma^z]$	a vector showing the x, y, and z components of σ,
	$[\tau^x, \tau^y, \tau^z]$	a vector showing the x, y, and z components of τ.
4.5	$r_i(\hat{u})$ and $s_j(\hat{v})$	(see 4.1),
	$r_i^{(1)}(\hat{u})$ and $s_j^{(1)}(\hat{v})$	(see 4.3),
	$R(\bar{u}), S(\bar{v}), T(\bar{w})$	(see 4.1),
	$\hat{u}$ and $\hat{v}$	(see 4.1),
	a_T and b_T	(see 4.1),
	$R^{(1)}(\hat{u})$	the first derivative of a spline $R(\bar{u})$ evaluated at $\bar{u} = \hat{u}$.

4.6	$r_i^{(\lambda)}(\hat{u})$, $s_j^{(\lambda)}(\hat{v})$	(see 4.3),
	$\Theta(\vec{a}, \vec{b})$	a function that returns the angle between two vectors,
	θ	a constant angle,
	$(\vec{a}^x, \vec{a}^y, \vec{a}^z)$	a vector in cartesian coordinate form,
	$\vec{a} \bullet \vec{b}$	the dot product of two vectors $= \vec{a}^x \vec{b}^x + \vec{a}^y \vec{b}^y + \vec{a}^z \vec{b}^z = \lvert\vec{a}\rvert\lvert\vec{b}\rvert\cos\theta$,
	$\lvert\vec{a}\rvert$	the magnitude of a vector.
4.7	$r_i^{(\lambda)}(\hat{u})$, $s_j^{(\lambda)}(\hat{v})$	(see 4.3),
	$\theta(t)$	an angle function
	t	(see 2.1).
4.8	$[a_S, b_S], [a_T, b_T]$	(see 2.7)
	$\hat{u}$ and $\hat{v}$	(see 4.1),
	$\Theta(\vec{a}, \vec{b})$, θ	(see 4.6),
	$R(\bar{u}), S(\bar{v}), T(\bar{w})$	(see 4.1),
	θ_1, θ_2	constant angles.
4.10	P , Q	3-dimensional points,
	$\Delta(P, Q)$	a function that gives the distance between two points,
	$R(\bar{u}), S(\bar{v})$	(see 4.1),
	$\hat{u}$ and $\hat{v}$	(see 4.1),
	δ	a constant representing the distance we wish to maintain between two points,
	$\delta(t)$	a function representing the distance we wish to maintain between two points.

5.1	$R_k(u)$ and $S_k(v)$	two spline curves in the k^{th} keyframe,
	u and v	parameters of the keyframe splines $R_k(u)$ & $S_k(v)$,
	k	a particular keyframe,
	n^r	$n^r + 1$ is the number of control vertices in spline $R_k(u)$,
	n^s	$n^s + 1$ is the number of control vertices in spline $S_k(v)$,
	$U_{i,k}$	the control vertices for spline $R_k(u)$,
	$V_{i,k}$	the control vertices for spline $S_k(v)$,
	i	(keyframe splines') control vertex index,
	$B_i(u)$	basis for spline $R_k(u)$,
	$C_i(v)$	basis for spline $S_k(v)$,
	$U_i(t)$	trajectory spline of $U_{i,k}$,
	$V_i(t)$	trajectory spline of $V_{i,k}$,
	t_k	the parameter value of t where the k^{th} keyframe is,
	t	parameter of the trajectories $U_i(t)$ and $V_i(t)$,
	j	(trajectory splines') control vertex index,
	m	(see 2.3), we have $m + 1$ keys,
	n^t	$n^t + 1$ is the number of control vertices in the trajectory $U_i(t)$ (as well as $V_i(t)$),
	$P_{i,j}$	control vertices for $U_i(t)$,
	$Q_{i,j}$	control vertices for $V_i(t)$,

	$D_j(t)$	basis for splines $U_i(t)$ and $V_i(t)$,
	$R(u, t)$	spline surface (combining keys with trajectory) from $R_k(u)$ and $U_i(t)$,
	$S(v, t)$	spline surface (combining keys with trajectory) from $S_k(v)$ and $V_i(t)$.
5.2	$F(expression)$	an operator (or map) that acts on *expression* and yields a new expression (perhaps a constant),
	$G(expression)$	another operator acting on *expression*,
	α	a constant,
	μ	a keyframe index.
5.4	P_i	vector of $P_{i,j}$ for a particular *i*,
	Q_i	vector of $Q_{i,j}$ for a particular *i*,
	E	a matrix of $m + 1$ columns and $n^t + 1$ rows,
	$e_{j,\lambda}$	the elements of *E*,
	λ	keyframe index,
	D	a matrix,
	U_i	vector of $U_{i,k}$ for a particular *i*.
5.7	$U(t)$	an interpolating function,
	a,b	two points being interpolated,
	P_j	control vertices,
	$D_i(t)$	a basis,
	$u_k(t)$	the k^{th} segment of a trajectory spline,
	ρ_k	keyframe data points being interpolated,

	t	parameter on the range [0,1],
	$h_i(t)$	the cubic hermite basis,
	$\{\tau, \chi, \beta\}$	(global) tension, continuity, and bias constants,
	$\{\tau_k, \chi_k, \beta_k\}$	tension, continuity, and bias constants for a
	....	particular keyframe,
	DD_k	the *destination* derivative (going out of ρ_k),
	....	$DD_k = u'_k(0)^+)$,
	DS_k	the *source* derivative (going into ρ_k),
	....	$DS_k = u'_k(0)^-)$.
6.3.1	$[a^x, a^y, a^z, a^w]$	a homogeneous coordinate vector of a point *a*
	$[a^x(t), a^y(t), a^z(t),. a^w(t)]$	a homogeneous coordinate vector of function $a(t)$ /
6.3.2	γ	which derivative we are taking.
6.3.3	n	number of constraints we have.

Appendix B: An Historical Note

Questions to be answered in this section are "What work was done previously by the University of Waterloo Computer Graphics Laboratory which encountered the MN problem?" and "What was done differently in that endeavor from what we are proposing in this thesis?".

The computer graphics lab had developed a two-dimensional spline curve editor, which worked with B-splines and Beta-splines. The editor was modified to do simple animation of spline curves. The modified editor would allow someone to draw a keyframe composed of several spline curves. Several keyframes could then be stored and inbetweened, using Catmull-Rom interpolation of the control vertices.

The editor did not allow the user to specify how the spline behaved at the parametric level; thus, control of the spline at the parametric level is what we are doing differently. The editor could be used to have two splines appear to the eye to behave a constraint such as tangency, and similarly the next keyframe could also behave the same constraint. But, the parametric value that the constraint is obeyed is not the same in the two keys (equation B.1). Having an equation like B.1 for the keyframes is precisely how Figure 1.3 was created. Figure 1.2, which obeys the constraint in the inbetween frame, used an equation like B.2. One would never expect equation B.1 to obey the constraint in the inbetween frame. For example, if Mickey's nose is attached the front of his head in one keyframe, and to the top of his head in the next keyframe, one could not expect the inbetween frame to have the nose still tangent to the head.

The following equation shows the constraint used which showed the existence of the MN problem:

B.1
$$\begin{aligned} R(\hat{u}_1, t_1) &= S(\hat{v}_1, t_1) \\ R(\hat{u}_2, t_2) &= S(\hat{v}_2, t_2) \end{aligned}$$

The following equation shows the constraint used for our solution to the MN problem:

B.2
$$\begin{aligned} R(\hat{u}, t_1) &= S(\hat{v}, t_1) \\ R(\hat{u}, t_2) &= S(\hat{v}, t_2) \end{aligned}$$