Volume 13, (1994) number 4 pp. 229–241

Animation using NURBS

M. Preston † and W.T. Hewitt ‡

† Department of Computer Science, University of Manchester, Manchester, M139PL
m.preston@manchester.ac.uk

‡ Computer Graphics Unit, Manchester Computing Centre, University of Manchester, Manchester, M13 9PL
w.t.hewitt@mcc.ac.uk

Abstract
All animation, whether generated by pen or processor, is concerned with the changing behaviour of synthetic actors through time. Computer animation has allowed an animator to regard different actors as functionally equivalent entities for the purpose of generating their behaviours. This has opened the door to visually impressive effects, such as actors changing their appearance, thereby becoming other actors.
For NURBS to become a tool which is useful to the writer of animation packages, it is necessary to provide this ability to in-between objects of different appearances through time.
This paper aims to present a technique for creating higher dimensional NURBS to model object deformation, as well as to illustrate its advantages.

Keywords: *Animation; NURBS; Deformations*

1. Introduction

All computer generated animation relies upon the ability of the computer to work out how an object can deform through time into another object. This allows the computer animator to provide the machine with less information about the animation than would be required in conventional hand-drawn film making. Such an all-encompassing basic is as much required *in-betweening* a 2 dimensional tom-cat in 2 frames, as it is needed to allow a computer generated 3D milk bottle to skip across a pavement. Without this capability computers would not make the animators life any easier.

As most computer animation to date has used polylines and polygonal faceted representations of objects this ability has revolved around the necessity to interpolate points between positions in cells. Upon this foundation such techniques as key-framing have been devised, to allow an animator to specify extents of motion in a film, and let the computer fill in the 'inbetweens'.

In many existing systems [15] the following method is taken.

- All objects are modelled as a set of points and surfaces
- Through time all points interpolate via some method
- Consequently the movie representation is a list of 'curves'. When they are all evaluated at a particular time this gives the scene at that time.

This movie representation is heavily dependent on the rendering of the object at a particular time. If greater accuracy is needed in the rendering of one of the instances of the object through time, then greater accuracy is needed in all instances. Consequently a more complicated movie representation is required to model the way these new points behave.

Non-Uniform Rational B-Splines, or NURBS, are a powerful geometric tool [12, 11] and are gaining increased use and recognition in many CAD & Graphics systems. NURBS have a rich geometric domain, in that they can precisely represent conic sections, free form surfaces, as well as surfaces one would not normally think of as curved (e.g., cubes). Algorithms for the creation and manipulation of NURBS exist, as well as several rendering methods [1, 17].

This paper presents one approach for the creation and management of a NURBS hyper-surface. This technique centres around the creation of a 4 dimensional NURBS representation which will describe the behaviour of a NURBS surface through a period of time, in a manner which will not adversely affect the quality of the final movie. The paper also describes an extension of this technique to encompass the creation & manipulation of a NURBS hyper-volume, for free-form deformation modeling.

Manipulation of a hyper-surface directly can be difficult, so intuitive interfaces are required to hide the underlying complexity from a user. The method outlined builds upon a technique [16] to create 3 dimensional NURBS from 2 dimensional sections (skinning).

Following a discussion of related work Section 3 reviews the NURBS surface, and illustrates its suitability for animation. Section 4 outlines this technique, and its application to NURBS. Section 5 illustrates how this approach can be generally applied to produce a 4 dimensional object from several 3 dimensional sections. It then goes on to explain how the state at a particular time can be calculated, for rendering.

In section 6 this technique is extended to the production of a hyper-volume, to be used in the generation of Animated Free Form Deformations (AFFD). Section 7 discusses implementation issues arising from the use of the NURBS hyper-surface, and shows examples of its use in a real animation system. Finally section 8 reviews the technique and outlines areas for future work.

2. Related Work

Most computer animation systems represent the four dimensional nature of actors implicitly as a collection of trajectories and rotations. The wide variety of object definitions has precluded the use of a unifying common 4D representation.

Four dimensional deformations however have been developed [13, 6] upon techniques for geometric deformation control [5]. Animating Free Form Deformations (AFFD's) allow the object to be altered through time both by moving the object through a deformation lattice, and animating the lattice itself.

FFD based techniques have become extremely popular, but they still cause problems. Hsu, Hughes & Kaufman [9] note that exact control of shape is difficult to achieve, and complicated deformations require large numbers of control points, which confuses the interface. Whilst the effects of the second problem can be ameliorated by use of higher level lattice interaction techniques, such as the facility to move logical groups of control points, it is still difficult to achieve a desired shape, as the user is frequently unaware of the purpose of the control points, and the underlying parametric deformation.

They propose a *direct manipulation* technique as a solution to this problem. Here the user moves points on the surface. Once the user has specified a surface change, the system calculates the required changes to the enclosing lattice to achieve this effect. By using a least-squares solution the effect of the deformation appears to smoothly taper off. To the user this gives the impression of manipulating a malleable surface. This technique could be combined with the AFFD key-framing to allow controllable 4D deformation.

AFFD techniques however rely on all the lattices at the key frames having an equal number of control points. Consequently if one extreme of the deformation requires a complicated lattice, all the frames must have a cluttered lattice, which will be cumbersome to work with.

Bechmann & Dubreuil [2] proposed the use of a space-time deformation model, where the user can interactively control four dimensional deformations. This is specified by a set of 4D constraints, each of which is a displacement of one point in space time. A deformation relevant to animation usually consists of constant time constraints, which for each point $P_i(x_i, y_i, z_i, t_i)$, never have their t_i's altered.

An extrusion function controls how moving a particular 4D point affects the resulting space-time object, and the authors outline 3 sample functions, providing varying degrees of locality of deformation & efficiency of calculation.

A projection matrix is then calculated which satisfies each of these n constraints. By applying this matrix to the object we can step through the deformation.

Whilst this technique avoids some of the problems inherent in the AFFD (there is no restriction on the complexity of the deformation from frame to frame), the technique is not particularly intuitive, and it is difficult to see animators, who may not understand B-spline extrusion functions, finding it comfortable to work with.

3. Using NURBS

The NURBS surface can be expressed as :

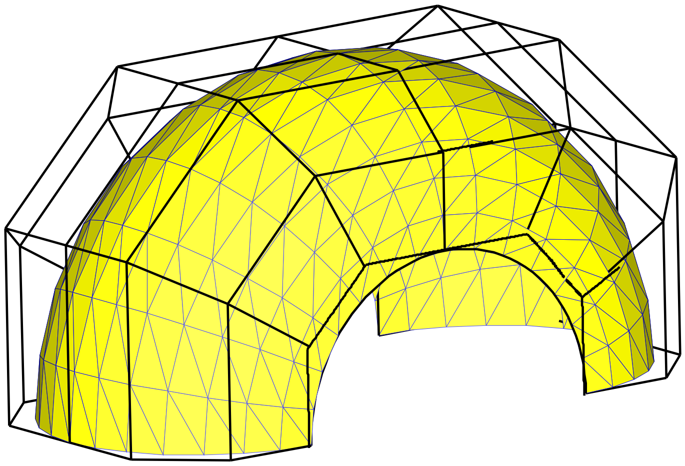

Figure 1: *A NURBS surface*

$$\mathbf{S}(u,v) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}\mathbf{P}_{i,j}}{\sum_{i=1}^{n}\sum_{j=1}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}}$$

Where $\mathbf{P}_{i,j}$ is a 2D array of 3D control points, $w_{i,j}$ are called the weights, $N_{i,p}(u)$ and $N_{j,q}(v)$ are normalized B-Splines of degree p & q respectively. Figure 1 shows a simple surface, with the control point net enclosing the shaded surface.

The NURBS exhibits local control, i.e., moving a single control point will only affect the appearance of a portion of the surface. This, coupled with the ability to repeat knots, and thereby force the NURBS to interpolate a control point, allows us to represent straight edges and sharp corners with a NURBS.

NURBS allow us to represent a single surface in more than one way. By increasing the order, and manipulating the knots and control point the same surface can be represented. Whilst this makes it difficult to compare surfaces for conformity it is ideal for animation purposes, as it is possible to manipulate two NURBS until they are stored in a compatible manner.

Unfortunately, whilst the NURBS is ideal for the animation systems' purposes, it is often not suitable as a generic object to interact with in higher level packages. Many users find more familiar representations, such as polygons, spheres and triangle strips more intuitive, so consequently it would be helpful for any animation system to take account of this, and provide users with an interface they are comfortable with.

As NURBS can represent a wide range of surfaces, it is possible to design a system whereby the user sees the 'familiar objects'. The user can then interact with them, whilst internally the system converts these objects to NURBS surfaces for the purposes of in-betweening [7] (See figure 2). This abstracted approach is perhaps the ideal setting for NURBS in computer animation, as it obviates the necessity for the user to be aware of the workings of the system.

The 3 dimensional NURBS surface is represented by two parameters, u & v, which describe the shape of the NURBS in parametric space. To represent a 4 dimensional NURBS (the fourth dimension here being time) in geometric space will require 3 parameters, which we will call u,v & w.

To represent a 4 dimensional NURBS, which we will henceforth refer to as the hyper-surface, we will need to store an order & knots sequence for u,v & w as well as a 3 dimensional array of control points.

A representation of the NURBS hyper-surface, would then be

$$\mathbf{S}(u,v,w) = \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{o}\mathbf{P}_{i,j,k}R_{i,j,k}(u,v,w) \qquad (1)$$

where

$\mathbf{P}_{i,j,k}$ are the rational control points arranged in a 3D array

$\mathbf{R}_{i,j,k}(u,v,w)$ are basis functions.

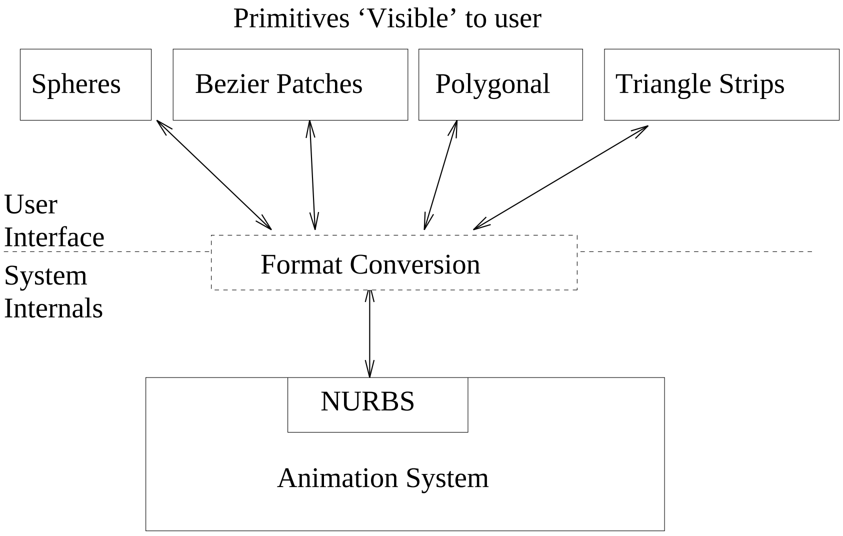

Figure 2: *Presenting the user with familiar objects*

4. Animation Theory

In creating a movie for the computer to animate, the user wishes to provide the machine with enough information for the system to fill in the rest of the movie. One of the most straightforward, for the programmer at least, but by no means the only way of doing this, is by key-framing. Here the user supplies several scenes. Each scene consists of several 3 dimensional objects, which we will refer to as actors. The user must then supply some information to outline the behaviour of these actors through their lifetime.

The system must then take these 3D actors, and create a 4 dimensional scene, which is the movie. When the user wishes to render this 4D scene the system must determine the 3D state, at a particular point in time. This is done for regular intervals of time, and the resulting image is presented to the user in a manner which provides the illusion of animation (e.g., by the use of video).

Techniques of animation other than key-framing tend to provide the user with a different interface to the process, but this jump from 3 to 4 dimensions remains the same.

Abstractly we need two functions internal to the animation system in order to provide this capability :

```
3d_to_4d (list of objects, list of behaviours)
  for each object in scene
    for object & its corresponding behaviour
        at time t
      add object & behaviour to 4D object
      increment t

4d_to_3d (time)
  for each object in scene
    calculate object at time
    display object
```

The first routine takes the definition of the movie and creates a 4D representation, and the second interrogates this representation at a particular time. More abstractly still, we are adding an extra dimension of information to the scene to create the movie , and extracting a more specific representation by removing a dimension of generality.

Reasoning about the behaviour of a four dimensional object is difficult, so in order to simplify the discussion we can first consider the animation of a 2 dimensional scene, which is represented using a 3D model. The techniques used in this simpler situation can then be extended to encompass the full four dimensional problem.

Therefore we need to develop a technique for beginning with the scenes described in 2 dimensions (with their attendant behaviours) and the generalized movie being stored in 3 dimensions.

This method involves taking

- Several 2D objects, which we will henceforth refer to as sections
- A behaviour.

As the technique we are going to use is widely used in object modelling we will call the behaviour a spine, as

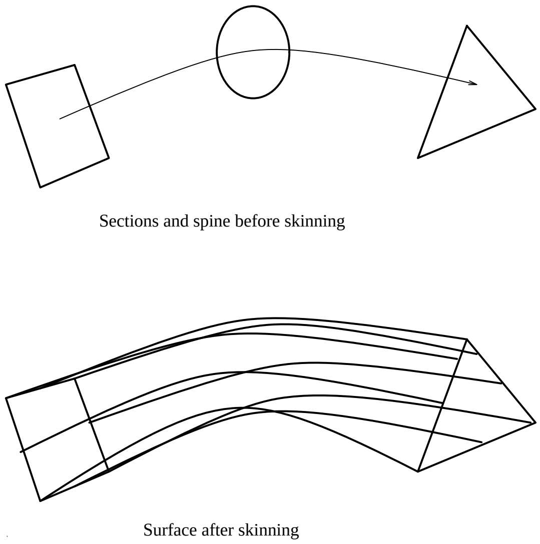

Figure 3: *3 dimensional skinning*

in surface modelling it corresponds to the 'backbone' of the object.

These 2D sections are placed on positions along the spine to give a 'wire' outline of a 3D object (as in figure 3. We have to then calculate how we would represent this 3D object as a solid model. This 3D model would then need to be simplified down to 2D again at a particular time in order to display the section at that moment.

For conveniences sake we will think of the distance along the spine as varying from 0 to 1, which will correspond to the beginning and end of the movie defined in this way.

With NURBS this is convenient as we can represent the spine by a NURBS curve, just like each of the sections. The technique for performing this 2-3D dimension increase, also known as skinning, is outlined in [16]. We can simplify this process by restricting the positioning of sections along the spine to places where there are knots in the underlying NURBS. Whereas this is insufficient in surface modelling, it is adequate for animation, for as we will create the spine ourselves from a trajectory supplied by the user, we can ensure there are knots at the points corresponding to the key-frames.

4.1. 3D Skinning

Three dimensional skinning, then, is a method of generating a surface from multiple cross sections (curves). If the user chooses to repeat the same cross section it will produce an extruded surface, but the algorithm remains the same.

The algorithm can then be expressed as :

```
/* Normalise in u */

raise all the sections to
  a common degree of complexity;
produce a common knot vector
  for each of the sections;
ensure all sections have the
  same number of control points;

/* Build up resulting 3D NURBS */

for the v direction
  copy the relevant data from the spine
    supplied to the algorithm;

for each section, 1..n
  Calculate the position in cartesian
    space on the spine which
    corresponds to the nth knot;

  Position the nth section at this point;
  Calculate direction vector of spine at
    nth knot;
  Choose a common normal to the spine at
    this point;
  Twist the section so a vertical in its
    2D plane is aligned to
    the normal of the spine at that point;
  Place the translated control points from
   the nth section into the 3D NURBS control
   points. They form the nth slice in the
   v direction;

output the 3D NURBS;
```

This provides us with a 3D representation of the NURBS. A few points in the algorithm warrant special discussion.

1. When we produce a common knot vector for the sections we need to
   - Union all of the knot vectors in turn.
   - Apply the Oslo algorithm [3] to each of the sections, with the unioned knot vector.

   This has the effect of ensuring that all of the sections will have a common number of control points.
2. A curve does not have a normal, so when we need to calculate the twist that we will apply to the sections, we have to ensure we always take a common orientation,otherwise the sections would appear to be twisted at random angles. Other, more advanced techniques could be employed, [14].

3. A suitable way of calculating the direction vector at the nth knot is

$$direction(\eta_n) = \frac{(\eta_{n-1} \rightarrow \eta_n) + (\eta_n \rightarrow \eta_{n+1})}{2}$$

where η_n is the control point corresponding to the nth knot. Note that this method will not work if the spine provided to the algorithm is a point. In this case the algorithm should perform no transformation.
Figure 4 shows the effect of translating a section onto a given spine.
4. The control point array is created in the manner outlined in the table. The translated control points of the section form slices of the resulting 3D NURBS. Each row is formed from the n control points of a particular transformed section.

$Sec1_1$	$Sec1_2$	$Sec1_3$	$Sec1_4$	$\ldots$	$Sec1_n$
$Sec2_1$	$Sec2_2$	$Sec2_3$	$Sec2_4$	$\ldots$	$Sec2_n$
$Sec3_1$	$Sec3_2$	$Sec3_3$	$Sec3_4$	$\ldots$	$Sec3_n$
$Sec4_1$	$Sec4_2$	$Sec4_3$	$Sec4_4$	$\ldots$	$Sec4_n$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

It is worth emphasizing again that this approach is not truly general, it only works where the number of sections corresponds to the number of knots in the spine. As we will probably capture the trajectory of these 2D objects by fitting a curve to a set of points, it is not difficult to ensure this is always the case.

5. 4D Skinning

Now we need to increase the generality of the above algorithm to allow for the creation of 4 dimensional states from 3 dimensional 'sections'. We will continue to use the same notation, so the trajectory is still referred to as the spine, despite the fact it no longer corresponds to the backbone of a simple solid object.

Note that this spine will often not appear (should it be rendered) as a trajectory through space, as it may be used to control the deformation of an object whilst it remains still in cartesian space. This does not affect the algorithm in any way.

To represent a NURBS hyper-surface we will need to store knot vectors for the u, v & w directions, and a volume of control points, which position the hyper-surface in space-time.

In order to express several NURBS surfaces as parts of the hyper-surface we need to ensure that each section has a like number of control points, and a common degree of complexity. Once this has been achieved the control points for each surface are copied into a 'slice' of the hyper-surfaces control volume.

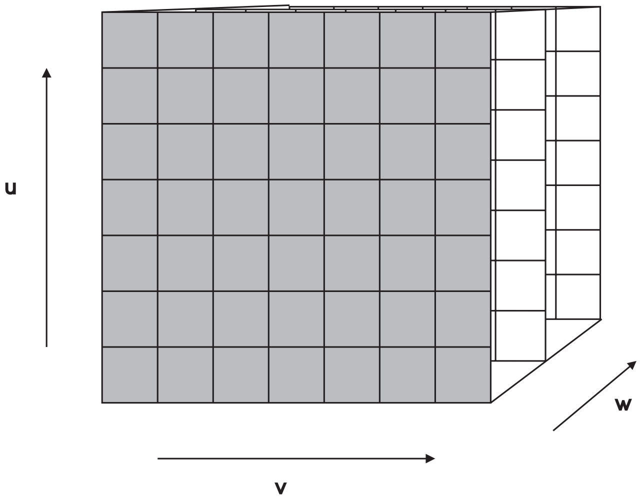

Figure 5: *Adding the control points of a section to the hyper-surface*

The process of ensuring a like storage format for each section is similar to the process of normalising each of the 2D sections for a skinning operation, i.e., each NURBS surface is elevated to a common degree of complexity and a common knot vector established in u & v.

In figure 5 we have created a NURBS hyper-surface from 3 NURBS surfaces. Each surfaces' control point grid is copied into the volume as a slice in u, v. Note that in this case we are using the hyper-surface to represent the behaviour of a surface through time, so w can be thought to correspond to time.

The w knot vector is obtained from the 'spine' provided to the algorithm. This encapsulates the rate of metamorphosis of the surface through time. Conventional animation systems provide acceleration & deceleration curves for the user to manipulate, and these would probably be used as an interface to manipulating the spine NURBS.

Once we have created our hyper-surface, here by creating a 4 dimensional skinning, we need to be able to 'evaluate' the surface at points in w in order to render different frames of an animation.

Recall that we can generate a surface using the 3D skinning algorithm when given several curves, and so we would like to determine, from the hyper-surface, the component curves of a surface at a particular time. This is achieved by taking a number of slices from the control volume in v, w, to generate several NURBS surfaces (see figure 6). Each of these surfaces represents the way a part of the hyper-surface behaves in time. By evaluating these surfaces we obtain a col-

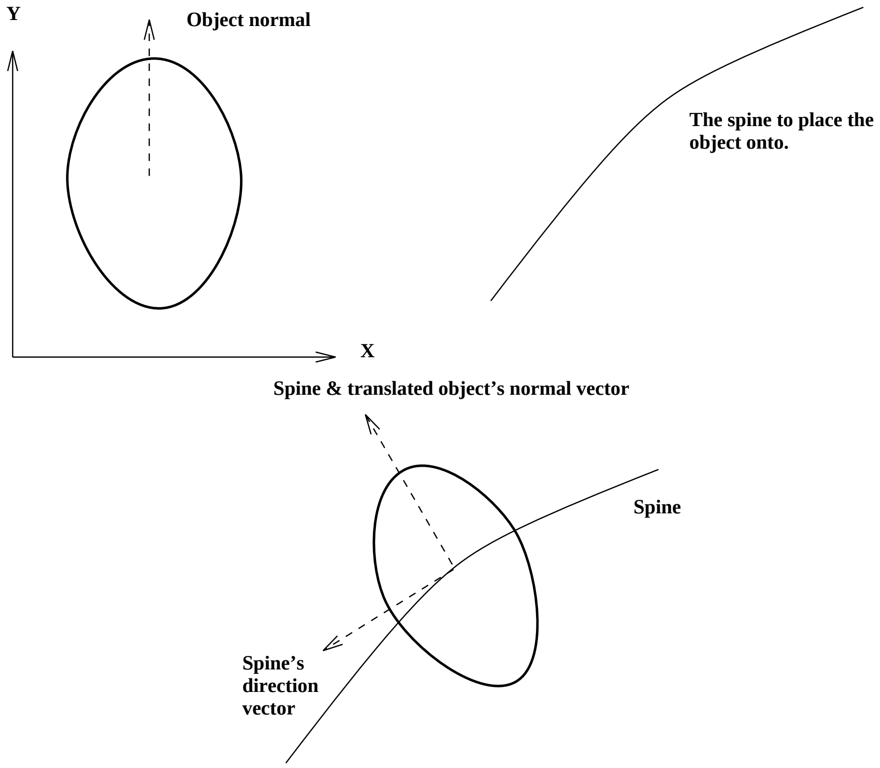

Figure 4: *Translating the sections onto the spine*

lection of NURBS curves, each of which encapsulates part of the appearance of the surface at that point in time. By using the 3D skinning algorithm we can generate the surface at the required time.

In figure 7 we have created a simple hyper-surface by supplying two NURBS surfaces, the first a flat surface, the second an extruded curve. The spine used was a simple linear, order 2 NURBS, which gives an even metamorphosis between the 2 extremes.

Though the volume of control points is a general solution of the storage of the hyper-surface it may not be the ideal format for all uses. During an evaluation of the hyper-surface the first step involves extracting several NURBS surfaces in v, w. If we know that we will be evaluating a hyper-surface a large number of times, e.g., during an animation, it may be useful to store the hyper-surface as a list of NURBS surfaces, i.e., the v, w surfaces. This will simplify the evaluation process, but slightly complicates the storage, as it is necessary to repeat the knot vectors for each 'slice'.

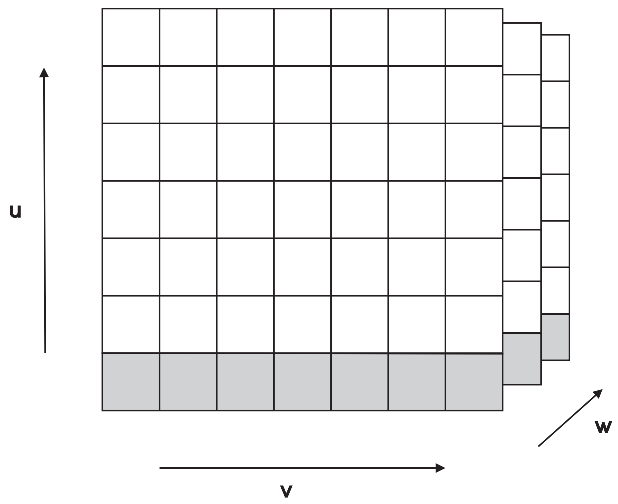

Figure 6: *Extracting a v, w surface from the control volume*

6. Animating Free Form Deformations

AFFD's are normally created by the user specifying a Bézier volume at several key frames. These frames

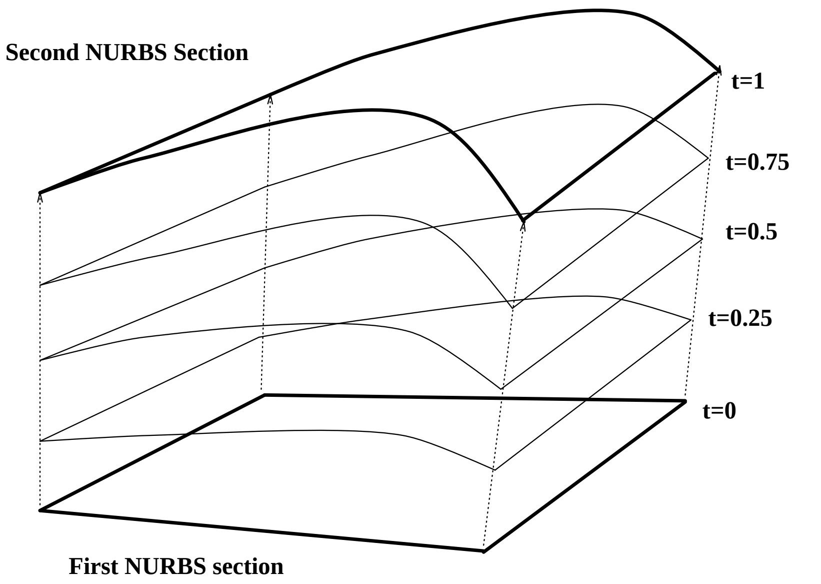

Figure 7: *Calculating the NURBS surface at various points in time*

are then used to create an in-betweened AFFD lattice, which can be evaluated at different points in the animation to determine which deformation to apply to an object.

Whilst this technique allows a wide variety of deformation effects the user is constrained to produce a sequence of lattices, all of which have the same number of control points. This means that we may only need the complexity at one key frame, but we must manipulate all the excess control points in all the other key frames.

As we add control points to a lattice this means that in order to achieve a certain affect we must manipulate more points. The beauty of FFD techniques lies in their ability to create interesting effects in an intuitive manner. Any forced increase in the number of control points then inhibits the FFD's effectiveness. Consequently we would like to be able to model an AFFD created using key frames of different complexity.

The NURBS hyper-surface, although presented as a technique for modelling the behaviour of a surface through time, can be thought of as a NURBS volume. If we add an extra degree to the storage of the NURBS we can store a hyper-volume, or a model of the behaviour of a volume through time. If we can use a NURBS volume then as the lattice in an FFD, we can create different size lattices as different key frames, and employ skinning techniques to generate an AFFD hyper-volume.

Free form deformation techniques work by

- Determining, from the 3 space position of an object point (which may be the vertex of a polygonal object, or the control point of a parametric surface), its position in lattice space, i.e., the u, v, w triple.
- Deform the lattice, which usually involves the user dragging points, though this could be the result of a physical model deformation.
- Determine the deformed position of the object point

Using Bézier volumes the first step, thanks to the uniformity of the Bézier functions, is trivial. However, using NURBS the step from 3-space to parametric space can only be solved by numerical methods. A similar problem occurs when attempting to render NURBS surfaces, and techniques exist to determine the solution [17]. Using a NURBS technique then we must determine the u, v, w triple for each object point at the start of the animation.

Consequently a NURBS volume based technique is slightly more expensive to use than its Bézier equivalent. Hardware implementation of the NURBS blending functions however help to reduce this cost.

Although we wish to store the lattice as a NURBS volume the principle advantage of FFD techniques are their intuitive nature. This can be retained by always presenting the user with a Bézier volume. When the user has edited this it can always be converted to an equivalent NURBS volume for storage and inbetweening purposes.

Once we have generated the different NURBS deformation volumes we need to skin these to determine the hyper-volume(**HV**).

$$\mathbf{HV}(u,v,w,t) = \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{o}\sum_{l=1}^{p} \mathbf{P}_{i,j,k,l} R_{i,j,k,l}(u,v,w,t) \quad (2)$$

where

$P_{i,j,k,l}$ are the control points arranged in a 4D array
$R_{i,j,k,l}(u,v,w,t)$ are the basis functions.

This hyper-volume is then evaluated at successive points in t, to produce the deforming volume at that time. Once the object to which this deformation has been updated (to take into account any movement), we determine where each object point is in the undeformed lattice, and then deform the object, in the same manner as we would using normal FFD techniques.

7. Implementation Issues

The NURBS hyper-surface, whilst a powerful technique, is not particularly suitable for presentation to the user as an interface to an animation system. The hyper-surfaces principal use is in a storage format internal to the system. Ideally no user should be aware that this representation is being used.

In order to validate the hyper-surface as a storage format a simple animation system was constructed, using the Manchester NURBS library [8], and the InterViews C++ class library [10]. The system will be used to demonstrate the effects a higher level interface will need to encapsulate.

In implementing an animation system using this internal representation, the designers must provide the ability for the user to control:

1. The change of position, both linear and angular, or trajectory, of an actor through time.
2. Acceleration & Deceleration.
3. The deformation of the object through time.

Modelling of trajectories is essentially a separate implementation issue. Whilst the hyper-surface can model this, by transforming the surface sections the hyper-surface will represent the trajectory implicitly, it is just as easy to represent the trajectories, both through cartesian and rotational space, separately.

The second area, the acceleration and deceleration of the object, is represented in the hyper-surface in the w knot vectors. By manipulating the uniformity of the knots the system can control the speed of metamorphosis throughout the defined time. However, for ease of use, any animation system would provide conventional acceleration & deceleration graphs for the user to manipulate, which are then converted to a knot vector internally.

In the sample animation system the primitive approach has been implemented, in order to illustrate some of the problems. In figure 8 the central box represents the behaviour of different actors through time, with time increasing to the right, through the metaphor of a script. Each button in the script denotes the position of a surface in each objects hyper-surface. By bunching the sections together (by dragging the buttons across the script), the user can speed up the metamorphosis in certain areas.

Note that in this very simple animation, with only 3 surface sections, the level of control over the acceleration & deceleration given using this basic interface is sufficient.

Finally the deformation of the object must be represented by the hyper-surface. As mentioned previously there are two ways of using NURBS techniques to represent deformation.

Firstly,by deforming the NURBS hyper-surfaces' sections, the deformation of the surface through time can be controlled in a relatively coarse manner. By manipulating the control points of a NURBS surface directly the user can create a wide variety of effects, but without intimate knowledge of the nature of the NURBS, and the part played by knot vectors & basis functions in the appearance of the surface, it is difficult to achieve many effects.

The NURBS editor shown in figure 11 demonstrates a simple point-and-drag interface for the manipulation of NURBS control points. As the user moves the points the system gives real time feedback of the shape of the surface.

This problem can be reduced by presenting the user with a non-NURBS interface, so he or she will manipulate a quad-mesh, which is then converted internally to a NURBS surface for animation.

For even better control over the deformation of the object through time, without needing to know how the object is stored or presented, the system can present a NURBS hyper-volume based AFFD interface. Here the user manipulates the lattice in a similar manner to the way the NURBS control points are moved in figure 11, to generate a deformation volume for each control frame. (Note that the lattices may have a different sequence of key frames to the actors, as these are only used to generate the space-time hyper-volume and hyper-surfaces respectively).

As has been previously noted the lattice is normally

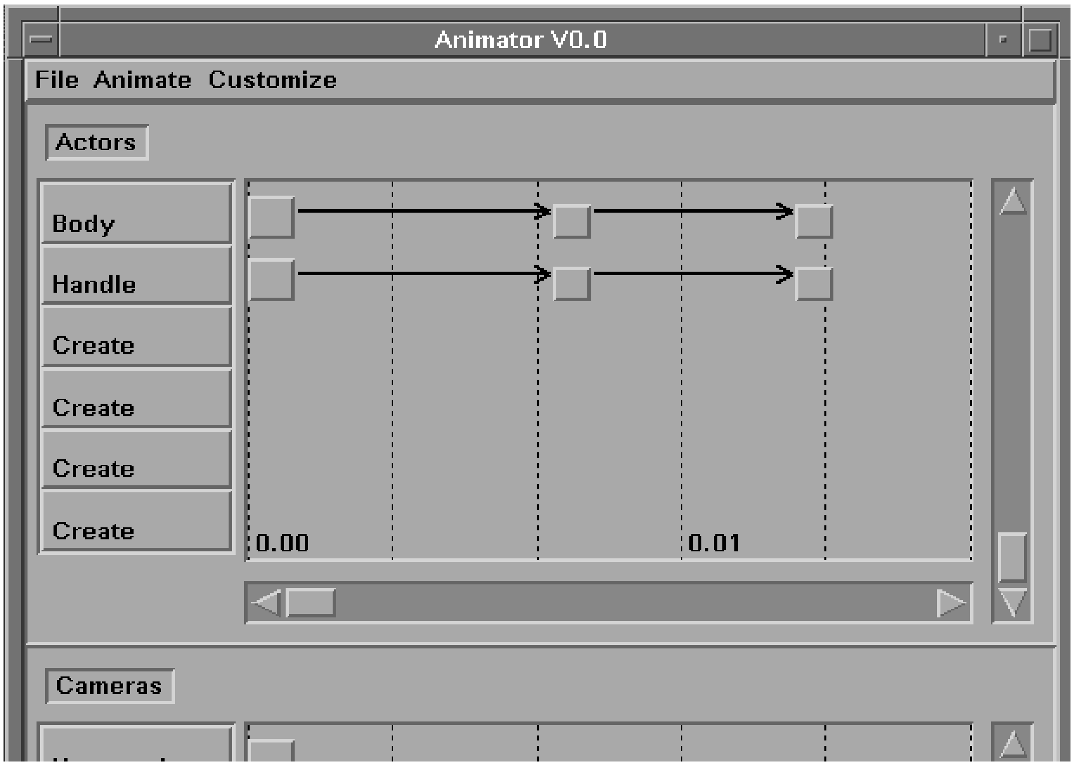

Figure 8: *Control of rate of metamorphosis by manipulating the position of the 3D sections in the time defined by the hyper-surface*

presented as a Bézier volume, and there is no reason why that shouldn't remain so. Internally the volumes are converted to NURBS volumes, for skinning into a hyper-volume. In this manner the user is presented with greater flexibility, i.e., different size lattices at different key frames, without the onus of learning a new tool.

The Bézier volume deforms points uniformly, though the authors believe there is no reason why direct manipulation of FFD techniques can't be employed with NURBS volumes, to generate *non-uniform* lattices. The least squares calculation employed by direct manipulation [9] determines how to manipulate the lattice to achieve a certain effect. If the 'cost' of achieving a certain change is too high, it is possible to add knots & control points in the volume, to give increased freedom of manipulation around the area of interest. This 'cost' would need to be set carefully, to prevent the lattice quickly becoming too complicated.

In implementing such a technique the system should attempt to divorce the users perception of the animation control from the technique used. If a naïve approach is used, the system may generate surprising animations, as the way the system represents a particular surface has a direct effect on the motion produced. For example, if the user chooses to animate a sphere changing into a face, the orientation of the NURBS surface used to represent a sphere may not match that used for the face. This would cause the sphere to appear to spin as the system generated the animation, which may not be what the user desired. If a simple approach is taken, the user should be allowed to manipulate the NURBS sections directly to help alleviate such difficulties.

To demonstrate a simple animation created using the hyper-surface 3 key points were defined, in order to create an animation of a deforming coffee cup. Each frame consists of 2 NURBS, one for the body, and on for the handle. Figure 8 shows how, in our simple animation system, the position of the key frame in the time of the hyper-surface, are moved.

In figure 9 the animation has been evaluated at varying values of t to indicate the deformation. After changing the second cup-body to add a bulge (as in figure 11), the animation is shown again (fig 10) to show the effect. Figure 12 shows a sample frame from the completed animation.

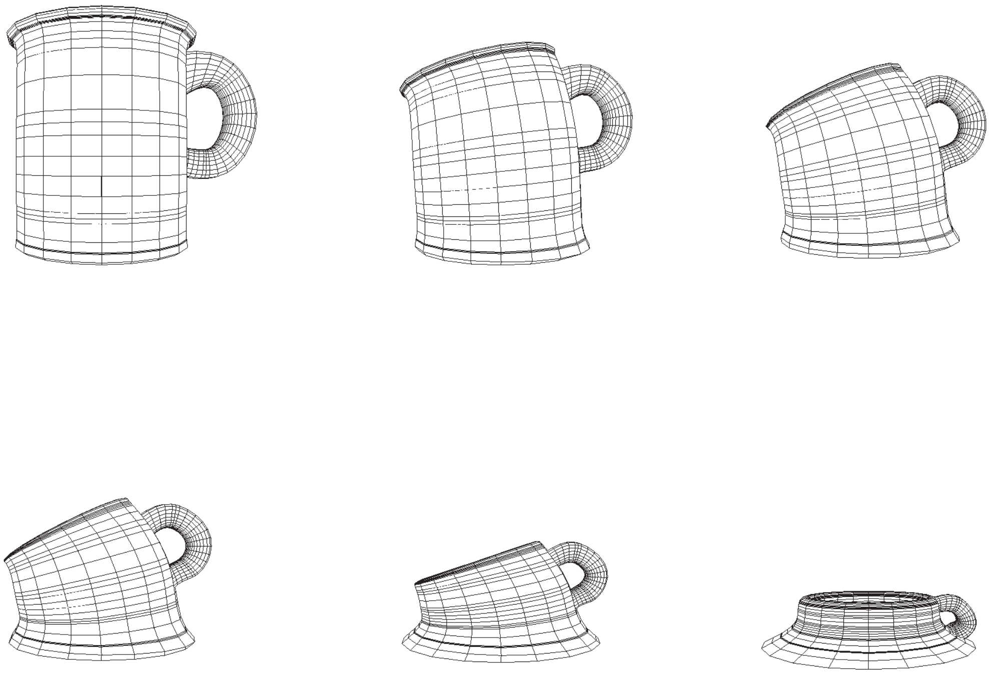

Figure 9: *A simple animation*

8. Conclusions

To reiterate.

1. Given that a NURBS can represent a large range of surfaces, then the behaviour of a large variety of surfaces through time can be modelled with this method.
2. By changing both the positioning and appearance of the trajectory, and the uniformity of t along it, we can easily model both different shape deformations and acceleration and deceleration through time.
3. By extending the theory to encompass a NURBS hyper-volume we can model an increased variety of deformations using AFFD techniques, without forcing the user to be aware of the NURBS.
4. The non-uniformity of a NURBS volume lends itself to uses in direct manipulation techniques.

Future research will centre around two areas.

Firstly the application of parallel techniques to improve the efficiency of this in-betweening method. By storing the hyper-surface as a list of NURBS surfaces, each of which holds the v, w portion of the control volume, we can split the evaluation stage of the hyper-surface into several surface evaluations, which can proceed in parallel. By use of a master-slave algorithm we can place the sections on slave processors, and employ the master processor to send out evaluate messages, and collate the returned NURBS curves into the final surface. This technique may help to alleviate the time penalty of using parametric techniques in real time animation.

Whilst the hyper-surface allows great flexibility in the representation of the conceptual four dimensional object, it means that some rendering techniques become extremely inefficient. In particular motion blur techniques [4] become extremely expensive, as the repeated evaluation of the hyper-surface, in order to adequately sample the exposure time, takes a large amount of time. Recent work has concentrated on improving the efficiency of this operation by employing four dimensional ray tracing techniques.

9. Acknowledgements

Martin Preston wishes to thank his parents for supporting him while not doing anything particularly worthwhile at University, and SERC, under whose funding some of this work was completed. Both of us

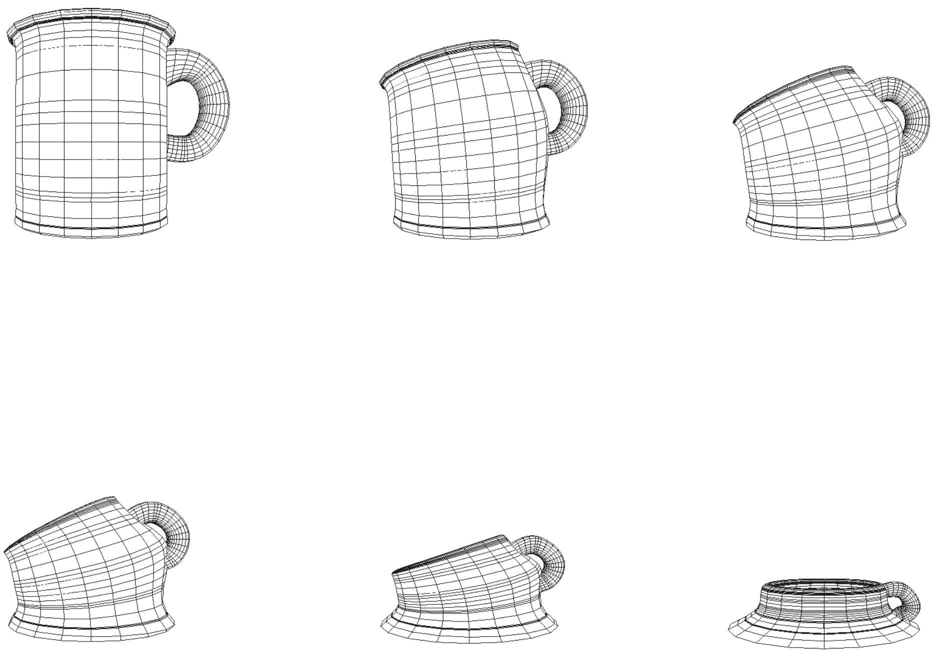

Figure 10: *The animation re-rendered after a bulge has been added to the cup in the middle surface.*

Figure 11: *Deforming the NURBS by manipulating the control points (Adding a bulge to a coffee cup)*

Figure 12: *A sample frame showing the cup deforming. One NURBS is used for the cup and another for the handle.*

wish to thank Mary for making the occasional cup of coffee, and the rest of the CGU staff for their help.

A particular vote of thanks goes to Fenqiang Lin and Chris Whitehead for their respective NURBS editors, without which this work would have taken much longer.

References

1. Salim Abi-Ezzi and Leon A. Shirman. Tesselation of Curved Surfaces under highly varying transformations. In *Proceedings of Eurographics '91*. Eurographics, 1991.

2. Dominique Bechmann and Nicholas Dubreuil. Animation through space and time based on a space deformation model. In *3^{rd} Eurographics workshop on Animation & Simulation*, 1992.

3. E Coehn, T Lyche, and L L Schumaker. Algorithms for Degree Raising of Splines. *ACM Transactions on Graphics*, pp. 171–181, July 1985.

4. Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed Ray Tracing. *ACM SIGGRAPH Computer Graphics*, **18**(3), pp. 137–144, Jul 1984.

5. Sabine Coquillart. Extended free-form deformation: A sculpturing tool for 3D geometric modeling. *Computer Graphics*, **24**(4), pp. 187–196, August 1990.

6. Sabine Coquillart and Pierre Jancène. Animated free-form deformation: An interactive animation technique. *Computer Graphics*, **25**(4), pp. 23–26, July 1991.

7. W. T. Hewitt, Fenqiang Lin, and M Preston. NURBS : A Tool or an Ornament for Graphics. In *State of the Art Reports, Eurographics '92*, 1992. Eurographics Technical Report Series EG92 SA.

8. W. T. Hewitt and D. Yip. The NURBS Procedure Library. Technical report, Computer Graphics Unit, University of Manchester, 1992. CGU 76.

9. William M Hsu, John F Hughes, and Henry Kaufman. Direct manipulation of free-form deformations. *Computer Graphics*, **26**(2), pp. 177–184, July 1992.

10. Mark A. Linton et al. InterViews Reference Manual. Technical report, Stanford University, 1992.

11. Les Piegl. On NURBS: A Survey. *IEEE Computer Graphics & Applications*, pp. 55–71, Jan 1991.

12. Leslie Piegl and Wayne Tiller. Curve and Surface Constructions using Rational B-Splines. *Computer Aided Design*, **19**(9), pp. 485–498, Nov 1987.

13. Thomas W. Sederberg and Scott R. Parry. Free Form Deformation of Solid Geometric Models.

ACM SIGGRAPH Computer Graphics, **20**(4), pp. 151–160, Aug 1986.

14. Pekka Siltanen and Charles Woodward. Normal orientation methods for 3D offset curves, sweep surfaces and skinning. In *Proceedings of Eurographics '92*. Eurographics, 1992.

15. Nadia Magnenat Thalmann and Daniel Thalmann. *Computer Animation, Theory and Practice*. Springer-Verlag, Second edition, 1990.

16. Wayne Tiller. Rational B-splines for curve and surface representation. *IEEE Computer Graphics & Applications*, pp. 61–69, September 1983.

17. Alle-Jan Von Der Veen. Intersection test for NURBS. In *IEEE Symposium on Computer Architecture & Real time Graphics*, pp. 101–114, Jun 1989.