2. 3D Character Animation

Movement results not only from physical structure of the character, but from its higher-level control as well. Animated characters are modeled after real-life animals or humans who are composed of both rigid interior skeletons and deformable exteriors. Articulated figure animation techniques are well-developed and can model the skeleton quite well, while deformable surface models for the skin are more complex and have not been as successful. Layered construction techniques attempt to combine these two approaches.

2.1. Introduction

Before we can start to animate characters on the computer, we must first examine the nature of movement itself and consider what aspects of a character we wish to move in order to bring about an appealing animation. Any real human or animal character moves in an extremely complex and subtle manner, resulting from a number of underlying causes. First, from a purely mechanical point of view, the character motion is a direct consequence of the geometrical and physical structure of its components and their various physical properties. This illustrates the strong interdependent relationship between modeling and animation. The structure of our character model will determine to a large degree its motion when animated.

In addition, this passive motion is under the control of a active muscles which themselves are controlled by a nervous system and brain. Not only does this active nervous system control use feedback based on multi-sensory input, but it also has "intelligence", making decisions and planning tasks based on some sort of internal model of the external world. When dealing with animated as opposed to real characters, we are confronted with another cause of animated movement: the artistic intent of the animator. In other words, animated characters don't just move according to physical or behavioral laws, they move the way the animator wants them to move. What the animator wants is very much dependent on his creative personality, artistic style and aesthetics, but it also has an objective relationship to the human visual system. For example, we humans are unusually good at recognizing and discriminating the motion of human and animal figures, and human facial expressions, so any imperfection or exaggeration of these aspects will naturally provoke a strong response in the viewer.

2.1.1 Model Structure

A central problem in 3D character animation is model structure: what kind of a computer model do we build? One extreme approach is to build as physically realistic a model as possible of a real human or animal character. Although this is an important avenue of research, for animation as well as for other fields such as medical graphics, it is not practical for a commercial, entertainment oriented animation system. Not only would such a system be too complicated and inefficient for interactive work, but it would also be too limiting for a creative animator. Animation, however realistic appearing, is, like any representative art form, an abstraction of reality. The nature of the abstraction is a combined result of the limits of the medium, the structure of the human visual system, and the stylistic intent of the animator. To determine a useful structure for 3D characters, we therefore must find an appropriate set of computer graphics modeling techniques which can be put together in a way that emulates real-life characters, but is efficient enough to be used interactively and is flexible enough to allow artistic control.

A way to approach this problem is to start with a simple structural model of a 3D character which can move in simple ways, and then refine it to incorporate more sophisticated kinds of motion. The simplest type of structure from an animation point of view is a rigid body. The mathematics of rigid bodies is well-defined and any type of solid or surface modeling technique that is affine invariant can be animated using rigid body motion simply by rotating or translating the coordinate system in which it is embedded. This forms a good starting point for animating characters, and allows us to control the large-scale motions of our characters independently of the details of their model or of their small-scale movements.

The next level of refinement is to view the character itself as a collection of connected rigid bodies, called an articulated figure. Articulated figures are one of the most successful and developed areas of 3D character animation not only because real-life character motion depends so much on the underlying skeletal motion, but because the mathematics of such structures is well-developed. Furthermore, our visual system is so attuned to the motion of articulated characters, that even a simple stick figure skeleton can be very convincing if its motion is well designed. So, the animation of articulated figures forms a well-defined area of research in itself.

Articulated figures of rigid bodies will always look mechanical and robot-like, regardless of the techniques used to model their links, so the next level of refinement is to take a layered approach, in which an interior skeleton, containing many joints, is used to move a single exterior skin surface such that there is a smooth deformation at the joints. This problem is an active area of research, and one approach to it will be presented in subsequent chapters.

A final step in the refinement of 3D character models is to recognize that surface deformations result not just from skeletal motion but also from muscle movements, environmental forces, collisions with external objects, and the whims of the animator. In order to implement these movements, we turn to global space deformation, local surface deformation and physically-based techniques.

2.1.2 Model Control

Selection of an appropriate model structure gives us a 3D character model which is constrained to move in a physically realistic (or at least acceptable) way. It also gives us a set of appropriate low-level numerical parameters (for example the joint angles) which can be varied to animate the character over time. What the structure does not contain is any representation of the nervous system of the character which makes it move. The passive model structure is therefore like a puppet without a puppeteer. To make the puppet move in an appealing and natural way, as though it had its own intelligence, it is necessary to build higher levels of motion control on top of the low-level parameters.

As we move from lower-level to higher level aspects of motion control, the problem becomes increasingly complex, not only because the task itself is complex, but also because we really don't yet understand the behavior of neural motor control. We also get more into the domain of what the animator would consider to be the creative and artistic aspects of character animation. Some animators would like to be able to issue a simple, task-level command to a character such as : "walk over there!", while others would prefer to work at a lower level in order to control the manner of the walk. An ideal animation system would allow the user to work at any level. Zeltzer, in fact, defines a continuous spectrum of control levels with "task-level" at the top and "machine level" at the bottom [Zeltzer 90]. An animator can therefore work at whatever level he is most comfortable.

If 3D character animation techniques are going to be successful and be accepted by professional animators, we need to identify those low-level areas of the motion control process which the animator considers to be the most tedious and least creative and attack them first, moving on progressively to higher levels of control. Two of these "lower-level" higher-level control techniques are inverse kinematics, and constrained dynamics.

2.2. 3D Modeling

The recent advances in interactive 3D graphics have been fueled by the increased hardware capabilities of graphics workstations. In fact, a new type of computer, the 3D Workstation, has been identified which specializes in being able to render complex objects in three dimensions, with lighting and texture mapping, at interactive speeds. These machines can be equipped with multidimensional input devices and can be made to drive virtual reality devices such as Data Gloves and Eyephones. An even newer type of workstation has been defined: the multi-media or digital-media workstation, which builds in such capabilities as digital audio, digital video and CD-ROM access. Such machines present the possibility of bringing together all forms of electronic media in one integrated environment.

It is important to distinguish between three-dimensional display devices, most of which use stereo viewing techniques, and 3D graphics, which can be rendered using stereo output devices but which is generally rendered on flat two-dimensional screens. What distinguishes 3D from 2D graphics is that 3D graphics uses three-dimensional models of the world, upon which all operations are performed except for the final rendering stage when it is mapped, using a perspective transformation, onto the two-dimensional screen.

The advantage of this is not only in rendering, where illumination models can be used to heighten a sense of realism, but also in modeling, because the world can be manipulated in its entire three dimensions. Since the real world in which we humans operate is a three-dimensional one, the user as well as the programmer can bring their real-world intuition to bear on the problem. With 3D input devices, this can be enhanced to use our three-dimensional motor skills to manipulate 3D objects interactively.

The addition of the third dimension, however, considerably complicates the subjects of both rendering and modeling. Interaction techniques, in particular, become more varied and the mathematics required becomes more sophisticated. The mathematics used in 3D computer graphics is generally based on the subjects of linear algebra, three-dimensional calculus and differential geometry. There are some particular techniques used by the computer graphics community, which will be outlined here.

2.2.1 Vector and Affine Spaces

Vector spaces can be defined very formally as a set containing elements called vectors with certain operations defined on it, such as addition and scalar multiplication, which must have certain properties, such as commutivity and associativity. Intuitively, we can think of three-dimensional vectors as arrows in 3D space with a magnitude and direction which can be decomposed into three elements. Given a coordinate origin, vectors can therefore be used to represent points in space by their offset from the zero coordinate origin. This can be misleading, however, because points and vectors are not the same thing. In particular, points can not be added together or multiplied by a scalar and there is no such concept as a zero point because there is no preferred location in geometric space.

Points in space can in fact be defined to belong to something called an affine space [Foley 90]. Most of the operations associated with a vector space are not defined in an affine space. However, it is possible to take the difference between two points of an affine space, which yields a vector, and to add an affine point to a vector which yields another affine point. This distinction between vectors, which belong to a vector space, and points, which belong to an affine space, is very important in 3D computer graphics.

2.2.2 Homogeneous Vectors

This distinction has been incorporated into the mathematical techniques of computer graphics through the use of homogeneous coordinates, which are now the standard way of representing points and vectors for computer graphics. Homogeneous coordinates represent 3D points and vectors as subspaces of a four-dimensional vector space. The fourth coordinate, usually called w, is set to the value 1 for points, and to the value 0 for vectors, so points form an affine 3D subspace of the 4D homogeneous space, while vectors form a vector subspace. It can easily be seen that the difference of two points will form a vector with $w = 0$, while adding two points is meaningless since w is no longer 1.

2.2.3 Homogeneous Transforms

The real power of using homogeneous coordinates, however, comes when transforming coordinate systems, which is done frequently in computer graphics. These transformations usually involve both a translation and a rotation, as well as scaling and shear, and are called affine transformations. Although these operations can be maintained separately, by using 4x4 homogeneous transform matrices, it is possible to represent all of these operations in a single matrix. These operations can then be composited by using ordinary matrix multiplication, so that the result of many such transformation operations can be combined into a single matrix.

Unfortunately, much of the computer graphics community has adopted a standard different from almost all previous use of matrix transforms in the fields of mathematics, physics and robotics, by representing vectors as rows and post-multiplying matrices rather than using column vectors and premultiplying matrices. A point in a local coordinate system is therefore transformed to a global one by post-multiplying it with the local-to-global transformation matrix:

$$\mathbf{p}_{global} = \mathbf{p}\mathbf{M}_{transform}$$

For affine transformation this can be expressed as:

$$\mathbf{p}_{global} = \begin{bmatrix} p_1 & p_2 & p_3 & 1 \end{bmatrix} \begin{bmatrix} M_{11} & M_{12} & M_{13} & 0 \\ M_{21} & M_{22} & M_{23} & 0 \\ M_{31} & M_{32} & M_{33} & 0 \\ t_1 & t_2 & t_3 & 1 \end{bmatrix}$$

where t_1, t_2, t_3 are the translation component of the affine transformation. The 3x3 submatrix M_{ij} represents the concatenation of a scaling, a shear and a rotation operation. It is a pure rotation for that subset of affine transformations known as rigid body transformations. The M_{ij} matrix can be decomposed into its component operations as follows:

$$\mathbf{M} = \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{bmatrix} \begin{bmatrix} 1 & h_{12} & h_{13} \\ h_{21} & 1 & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

where s_i are the scaling factors, h_{ij} are the shearing components, and R_{ij} is the general rigid body rotation.

A non-affine transformation which can also be expressed as a homogeneous matrix is used to calculate the perspective transformation for an observer on the z axis viewing a scene projected on the x-y plane from a distance d:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1/d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This transforms points into a distorted "clipping" coordinate space where the z-axis now represents distance from the eye or depth.

2.2.4 Hierarchy

3D graphics models can become very complicated, often involving thousands or even millions of polygons. An important modeling issue, therefore, is how to manage this complexity effectively. One of the principal techniques for dealing with this problem is to organize models into hierarchies. The basic form of modeling hierarchies is a tree data structure, however, this can be augmented using multiple referencing to make a directed acyclic graph or other more complicated structures.

2.2.5 Transformation Hierarchies

The most important use of hierarchical representations is to organize transformations between coordinate systems. This type of transformational hierarchy can be described as an n-ary tree of nodes with a homogeneous transform matrix at each node. The matrix represents the node's local coordinate transformation with respect to its parent, that is, multiplying a point by this matrix will transform it from local coordinates into the coordinate system of its parent. Every node can therefore store its geometric information it its own local coordinate system, which does not change even if the node is moved to a different location in the hierarchy. The global coordinates of the geometric information can then be calculated by walking up the hierarchy to the root node, concatenating each local matrix to form the final local-to-global transformation matrix. Multiplying by this matrix will then transform a local coordinate point into a global coordinate one.

One way to visualize this process is to draw a coordinate system diagram as a graph with nodes representing coordinate systems and arcs representing transformations. Moving along an arc is equivalent to multiplying by that arc's transformation matrix and moving backwards along an arc is equivalent to multiplying by the inverse transformation. This provides a powerful technique for organizing large assemblies of rigid bodies. It is particularly useful for interactive 3D graphics because it allows us to manipulate the transformations at any point in the hierarchy and therefore reposition all of its descendants at the same time.

2.2.6 Directed Acyclic Graphs

An extension to the strict tree structure is to allow nodes to be multiply referenced, although not by their own descendants. This results in a directed acyclic graph structure or DAG. This can be useful for representing large assemblies with many identical components: each instance of the component is just a multiple reference from a different node. As a modeling technique, this can have its difficulties since there is no longer a one-to-one correspondence between visual object and node. It also can present some implementation problems and normally a DAG node structure must be "unrolled" into a tree structure before rendering.

A simpler use of the DAG structure is to limit multiple referencing to other, non-hierarchical data such as color, material, and shape. This allows sharing of certain frequently-used attributes without the complexities of a full DAG hierarchy.

2.2.7 Attribute Inheritance

The defining feature of hierarchical modeling which makes it so useful for computer graphics is inheritance. When a node is attached to a parent node, it "inherits" the coordinate system of its parent. Other attributes of a node can be inherited as well such as color, material, texture, visibility status, or other more domain-specific attributes of the model. A standard way to do this is to implement these attributes as pointers to a data structure. If the pointer has a null value, it is considered to be undefined and the attribute will therefore be inherited from the node's parent. In this way, for example, "colorless" nodes can be moved from one parent to another, taking on the color of each parent in succession.

2.2.8 Copying

The use of a DAG structure has an important relationship to the issue of copying. Multiple referencing a single data structure provides a powerful alternative to copying it, especially when it is necessary to go back and change the structure later. With attribute inheritance, it is possible to have each reference displayed with different attributes. However, if it is desired to change the internal (i.e. non-inherited) attributes of one "incarnation" of the referenced structure but not the others, it is necessary to make a copy.

2.3. Elementary 3D Motion Control

Much of the success of early three-dimensional animation of the flying logos variety is due to ease and simplicity of moving rigid bodies in three dimensions. There are some subtleties to the subject, however, which is referred to as three-dimensional motion control.

2.3.1 Rigid Body Motion

Since the positions of three-dimensional objects in 3D graphics are usually specified using 4x4 homogeneous transform matrices, the simplest way to animate them is just to make the components of the matrix functions of time, or of interactive input. This can work well for the translation components of the transformation matrix, because they are independent of the other components and of each other. The position of the rigid body can therefore be controlled quite easily simply by controlling these elements of the transform matrix.

For the rotation components, however, this is not quite so simple. Since a rotation in space requires no more than three components in its representation, the nine upper-left components of the transform matrix are interdependent, containing as they do scaling and shear components which are not used for rigid body transformations. It is therefore necessary to parameterize rotations in a more fundamental way so that they are independent of each other. The traditional way to do this is by Euler angles. Euler proved that any orientation in space could be represented by a single rotation about an arbitrary axis, and that this rotation is equivalent to three successive rotations about the x, y and z axes. An arbitrary rotation in space can therefore be parameterized by these three angles.

This parameterization of rotation space has at least two problems [Watt 92]. The first, known as gimbal lock, occurs when the second parameter angle (y roll) is set to 90 degrees. In this situation, which is a singularity in the parameterization, a degree of freedom is lost and not all rotations can be represented. The second problem with Euler angles concerns interpolation, which is an important property for motion control. Since the three Euler angles are not truly independent parameters in rotation space, there can be more than one way to represent a single orientation and more than one way to interpolate between successive orientations. Interpolating a motion between two successive orientations by simply linearly interpolating their Euler angles, as though they were Cartesian coordinates, will generally result in a contorted motion and not the smooth rotation about a single axis that one would expect from Euler's theorem. Furthermore, this motion will depend on the orientation of the coordinate system.

A solution to this problem, which was developed originally for spacecraft orientation control and later introduced to the computer graphics community by Shoemake [Shoemake 85] is to represent rotations by unit quaternions. Quaternions were developed by Hamilton in the 19th century as an extension of imaginary numbers into three dimensions. Just as imaginary numbers are well suited to represent certain kinds of one-dimensional cyclic phenomenon (such as waves), it turns out that quaternions are well suited to represent cyclic rotations in three-dimensions.

Mathematically, a quaternion is defined as a number which is the sum of four components:

$$q = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$$

where *a, b, c* and *d* are scalars and

$$\begin{aligned}&\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = -1\\&\boldsymbol{ij} = -\boldsymbol{j}i = \boldsymbol{k}\\&\boldsymbol{jk} = -\boldsymbol{k}j = \boldsymbol{i}\\&\boldsymbol{ki} = -\boldsymbol{ik} = \boldsymbol{j}\end{aligned}$$

A more useful notation for computer graphics is

$$q = (s, \mathbf{v})$$

where s is a scalar and v is a three dimensional vector

$$(s, \mathbf{v}) = s + v_x\boldsymbol{i} + v_y\boldsymbol{j} + v_z\boldsymbol{k}$$

The product of two quaternions is another quaternion and can be shown to be:

$$q_1 q_2 = (s_1 s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2)$$

And the magnitude of a quaternion is defined as:

$$\|q\|^2 = s^2 + \|\mathbf{v}\|^2$$

Since quaternions form a four-dimensional space, unit quaternions lie on the three-dimensional surface of a unit hypersphere which can represent the space of possible 3D rotations. It can be shown that multiplication of unit quaternions corresponds exactly to a rotations in three-dimensional space. Unlike rotation matrices, the four quaternion components can be interpolated, resulting in a smooth rotation from one orientation to the next. Since unit quaternions lie on the surface of a four-dimensional hypersphere, linearly interpolating the quaternion components will result in non-unit quaternions so spherical interpolation must be used.

2.3.2 Scripting vs. Interactive Systems

Once we have a set of parameters with which to control the rigid body, it becomes necessary to specify exactly how each of these parameters will change in time. Since early computer animation systems were notoriously non-interactive, it became common to design a scripting language with which to specify the motion parameters. The entire animation can then be created by writing an animation script. Parameters can be directly specified as functions of time, or as results of more complicated algorithms.

Examples of script based systems are ASAS (Actor Script Animation System) [Reynolds 82] which is a Lisp-like language, MIRANIM [Magnenat-Thalmann 84] which uses an extension of Pascal called CINEMIRA, the object-oriented Clockworks system [Breen 87] and SOLAR [Chua 88].

The advantages of script-based systems are ease of implementation, since a simple parser is just about all that is necessary for input handling, and if the scripting language is general enough, almost unlimited flexibility. The disadvantages are the lack of immediate feedback and sense of direct manipulation that are the hallmarks of interactive systems, and virtual inaccessibility to anyone who is not a computer programmer.

As the performance of graphics workstations has improved, scripting systems are giving way to interactive animation systems. Interactive animation systems are usually more intuitive to understand and more accessible to non-programmers, who are ultimately likely to be the end users of commercial animation systems. They can be very difficult to implement, however and, since they are non-symbolic, they are generally not as flexible as script-based systems. Practical systems should allow some combination of both approaches to animation, and Zeltzer has defined a continuous spectrum of interactivity ranging from purely interactive "guiding" to purely script-based "programming" [Zeltzer 90].

2.3.3 Key Frame Animation

Interactive systems therefore need some kind of model or technique for animation which can be parameterized in such a way that these parameters can be easily and intuitively manipulated in an interactive environment. The most common technique used for animating rigid bodies is key-frame animation. The term comes directly from the traditional animation process of drawing key frames which are then in-betweened. In this case, the user of a key-frame animation system is considered the key animator. Rather than drawing the key frames, he instead places the rigid body into key positions and orientations, and the computer calculates the inbetween positions automatically.

Interactive key frame animation requires at least three proper components to be effective: an appropriate set of parameters to key on (unlike, for example, Euler angles), an interpolation method for calculating the values of the inbetween frame parameters, and an interactive technique for setting the key parameter values, that is, placing the object into its key positions. For rigid body motions, the three Cartesian coordinates for position and the four quaternion coordinates for orientation form an appropriate parameter set. The various interactive techniques, such as the virtual trackball and spaceball input devices can be used by the animator to specify the key-frame positions. The remaining problem is to determine a method for interpolating the inbetween values of the parameters. The simplest and most intuitive way to do this is straightforward linear interpolation: the difference between two adjacent key parameters is divided by the number of inbetween frames to determine the parameter increment for each frame. Linear interpolation does not usually give satisfactory results, however, because of discontinuities at the key values, and usually it is necessary to compute a smoother curve between the key values.

2.3.4 Spline Interpolation

Fortunately, the problem of interpolating smooth motions in space is almost identical to the problem of interpolating smooth curves in space. In fact, if a space curve is represented parametrically, the curve parameter maps directly onto time in the motion curve. The mathematics of interpolating smooth curves, called splines, through points in space, called control points, is well developed, and can easily be generalized to any number of degrees of freedom, although for interpolating orientation, special quaternion splines must be used. In this section we will see how splines has been adapted for motion control.

There are an immense variety of spline curve techniques that have been developed by the computer graphics community for interpolating or approximating control points. It was shown by Duff that many of these, in particular local splines of degree n, can be characterized by a system of linear equations written as a matrix [Duff 86]:

$$S_M(t+i) = \sum_{j=0}^{n} \sum_{k=0}^{s-1} t^{n-j} M_{jk} P_{i+k} \qquad 0 \le t < 1, 0 \le i < m-3$$

where $S_M(t)$ represents the parametric curve as a function of the parameter t, P_i is a set of m control points, and M_{jk} is the spline matrix, the values of which represent the type of spline. In the case of cubic splines, which are normally the most useful type, this formula reduces to:

$$S_M(t+i) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \mathbf{M} \begin{bmatrix} P_i \\ P_{i+1} \\ P_{i+2} \\ P_{i+3} \end{bmatrix} \qquad 0 \le t < 1, 0 \le i < m-3$$

2.3.4.1. B-Splines

The most popular type of spline in use is probably the Cubic B-Spline, and a variation on it, the Non-Uniform Rational B-Spline (NURBS), is becoming an industry standard. B-Splines exhibit a number of desirable properties which justify their popularity such as affine invariance, local control, and C^2 continuity. However, they do not generally interpolate their control points, so it is necessary to find a second set of B-Spline control points that will generate the desired interpolating curve. There are several techniques for doing this.

2.3.4.2. Hermite Splines

There are other types of spline curves, however, that do interpolate their control points, and these have been used successfully for key-frame interpolation. Hermite splines, which use the Hermite basis functions, interpolate their control points and are controlled in addition by ingoing and outgoing tangents at each point. Hermite spline curves between any two control points can be represented in matrix form as follows:

$$S_{\text{Hermite}}(s) = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_i \\ P_{i+1} \\ D_i \\ D_{i+1} \end{bmatrix} \quad 0 \le s < 1$$

where s is the parametric coordinate, P_i and P_{i+1} are the endpoints of the curve segment and D_i and D_{i+1} are the tangent vectors of the curve at the endpoints.

2.3.4.3. Kochanek-Bartels

A variety of interpolating spline curves can be derived from the Hermite spline by using different values of the incoming and outgoing tangent vectors. One way of determining the tangents is to average the source chord with the destination chord , resulting in the Catmull-Rom spline which is defined as:

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

Another parameterization, called the Kochanek-Bartels spline, was developed specifically for animation purposes [Kochanek 84]. This spline specifies the curve shape in terms of three parameters: bias *(b)*, tension *(t)* and continuity *(c)*. These parameters control the ingoing and outgoing tangents of the Hermite basis spline segments at each point as follows:

$$D_i = \frac{1}{2}(1-t)(1-c)(1+b)(P_i - P_{i-1}) + \frac{1}{2}(1-t)(1+c)(1-b)(P_{i+1} - P_i)$$
$$D_{i+1} = \frac{1}{2}(1-t)(1+c)(1+b)(P_{i+1} - P_i) + \frac{1}{2}(1-t)(1-c)(1-b)(P_{i+2} - P_{i+1})$$

and can be controlled globally or locally. The tension parameter controls the straightness of the line segments between control points. Curves with increasing tension will approach their control polygon. The bias parameter controls the degree of undershoot or overshoot of the curve as it passes through the control point. The continuity parameter controls the degree of inward or outward "kink" of the curve at the control point. These three form an intuitive set of parameters which can be tuned by the animator to control the manner in which the curve interpolates the control points.

2.3.5 Time Parameterization

A serious problem with using splines for interpolating motion is that, although it is possible to have good control over the shape of the curve in space, it is very difficult to control the speed at which the curve is traversed in time. Kochenek-Bartels splines allow a certain amount of timing control by adjusting the tension and continuity parameters, but this is limited because the parametric coordinate which determines the spline values does not necessarily represent anything physical about the curve. To make this coordinate represent an actual distance in space, we must reparameterize it in terms of arc length. This is not a trivial problem, and usually has to be done numerically. Once we have a function relating spline parameter to arc length, it is possible to control the speed (i.e. magnitude of the velocity) of the motion along the curve, by creating a velocity curve.

Although arc length is a useful concept for Cartesian space, it is not necessarily an important metric for controlling animation of other parameters. Steketee and Badler developed a technique called the double interpolant method in which a one-dimensional spline, called a kinetic spline, provides a mapping from the time coordinate into the position spline coordinate, referred to as key-frame number [Steketee 85]. This allows the overall geometrical path to be controlled fairly independently of the velocity curve.

2.3.6 Applications

By key framing the three position and orientation parameters, and then interpolating these values using spline curves for the position parameters and quaternion splines [Pletincks 89] for the orientation parameters, it is possible to build an interactive system for animating rigid bodies: the object is positioned and oriented interactively using, for example, a spaceball input device, and a key frame is recorded. This process is repeated until all the key frames have been recorded at which point the motion sequence can be played back. Key frames can be edited, inserted and removed, and the kinetic curve can be specified using a two-dimensional spline editor. If we are using Kochenek-Bartels splines, the tension, bias and continuity parameters can be adjusted and fine tuned as the motion is being played back, and the entire process can be conducted in an interactive way.

This technique can be applied to any of the transformations in a general modeling hierarchy, resulting in hierarchical motion control. Rigid body motion control works very well for animating flying logos in space or for controlling the overall motion of more complicated characters. For example, if a bird character is required to fly through the air and flap his wings, the motion of the bird's center of gravity could be animated with a key frame technique, while another technique is used to animate the wings on a stationary bird.

It is perhaps most useful for animating the virtual camera, which can be thought of as a rigid body as well. So, key-frame animation can be used to animate walk-throughs of architectural or other static scenes as well. Several popular commercial animation systems such as *Wavefront* and *Alias* use this kind of technique.

2.3.7 Multi-Track Animation

The key frame interpolation technique can be extended beyond rigid bodies to animate any parameter of a graphical model. For example, the color, size, or shape parameters of an object can be key framed. Many animation systems allow virtually any floating point parameter of the model to be animated in this way, such as CINEMIRA's MUTAN system [Fortin 83], and the *Twixt* system [Gomez 84].

Just as transformations can be arranged in hierarchies, motion splines can be placed in hierarchies. An example of this type of motion spline hierarchy can be found in the *Animator* system, described in [Gobbetti 93]. Any node in the hierarchy can be interactively animated along a spline curve using a type of interpolating spline, C-splines [Duff 86], resulting in complex, hierarchically-controlled motions of the leaf nodes in the hierarchy. This type of hierarchical animation, however, is generally too flexible and difficult to control for use in character animation

2.4. Articulated Figures

One of the interesting things we notice, when contemplating the animation of anthropomorphic characters, is that characters display two opposing characteristics: they act, in some respects, as flexible and deformable objects, and in other respects, as collections of rigid bodies. This fact is a result of the endoskeletal construction of vertebrate animals. We vertebrates are made, in fact, of a soft exterior surrounding a rigid interior. Since rigid bodies are easier to animate than deformable ones, and since early display devices could only display simple line drawings at interactive rates, it is not surprising that the first attempts at 3D character animation have focused on rigid articulated figures.

Articulated figures are collections of rigid bodies which are connected together in hierarchies so that their motion is constrained. These rigid bodies are called links, and they are attached together in such a way that they can either rotate or translate with respect to each other. These points of attachment are called joints and they are categorized by whether they are rotational or translational and the number of degrees of freedom they possess. A ball joint, for example, possesses three degrees of rotational freedom, while a single bearing possesses only one. In practice, multi-DOF joints can be composed of several single-DOF joints, so we only have to consider two types of joint: rotational or revolute and translational or prismatic. A simple type of articulated figure is an articulated chain, which consists of a base connected to an end-effector by a single chain of links.

2.4.1 Robotics

The mathematics of articulated figures, particularly articulated chains, was developed initially for the study of robotics but has since been taken up by the computer animation community for representing skeletons. Denavit and Hartenberg developed the first matrix formulation for representing the kinematics of articulated chains and a notation, DH notation, for describing it. In DH notation, which can be thought of as link-based, each link is represented by four parameters, θ, d, a, and α, which represent the joint angle, distance from the origin, offset distance, and offset angle respectively [Fu 87]. The relations between links (i.e. the joints) can then be represented using homogeneous 4x4 matrices (unfortunately for computer graphicists, using column vectors rather than row vectors). Forward kinematics calculations can therefore be made simply by concatenating these matrices in the same manner as computer graphics transformation hierarchies.

2.4.2 Skeletons

The theory of articulated figures was eventually adapted for computer graphics in order to represent the skeletal motions of humans, animals and animated characters, because the human and animal skeleton can be represented very well by such a structure. Stick figures of humans were demonstrated early as 1970 by Withrow [Withrow 70] and several simple human models were developed for industrial ergonomic studies in the seventies [Fetter 82] [Dooley 82]. Two early researchers in animating skeletons are Norman Badler, who developed an articulated figure with spheres as links called Bubbleman [Badler 82], and David Zeltzer who literally animated a human skeleton (a real visualization of human bones) with 22 degrees of freedom [Zeltzer 82]. Most computer graphics work in human skeleton motion, however, has regarded a "skeleton" as a simple stick figure of connected joints, to which rigid body links can be attached to represent the various body parts. The focus of research here is on achieving natural-looking or appealing joint motion, leaving the problem of putting a flexible "skin" on the skeleton as a separate issue.

When applied to 3D character skeletons, the link-oriented DH notation of robotics is not necessarily appropriate, and cannot handle branching structures well. A better approach is therefore based on joints, in which the skeleton can be thought of as a hierarchy of joints, to which links (which are purely graphical objects) can be attached at each node. In this way, an articulated skeleton hierarchy can be thought of as an extension to the standard computer graphics transformation hierarchy, the extension being the addition of a single degree of freedom (rotational or translational) at each node in the hierarchy. This type of articulated figure representation is referred to by Sims and Zeltzer as axis-position joint representation [Sims 88].

2.4.3 Joints

We can therefore integrate articulated skeletons into a traditional computer graphics hierarchy by extending the hierarchical node so that it maintains an extra degree of freedom, thereby becoming a joint. This requires maintaining information about the type of joint (rotational or translational), its value (an angle for rotational joints), information about the joint limits, and a default or rest position for the joint. Joint limits are important to prevent the skeleton from being placed into unrealistic positions, while the rest position is useful for certain types of control algorithm. The joint also should contain a pointer to a graphical subhierarchy which represents the link attached to that joint.

If we think of the entire hierarchical structure as a fixed, rigid body, it can be considered to have six degrees of freedom for purposes of animation: three translational and three rotational. For each node that is made into a joint, an additional degree of freedom is added to the hierarchy. These degrees of freedom can be treated as motion parameters just as the rigid body ones, and can be animated using the same techniques such as key-frame animation.

2.4.4 Links

A consequence of this separation of joint hierarchy from the links is that a skeleton can be described independently of the links which populate it. This has advantages in that the same skeleton can be reused many times with different geometrical objects as links. In the simplest case, the skeleton can be rendered with simple lines between the joint coordinate system origins, resulting in a pure (in the computer graphics sense) stick figure skeleton. A disadvantage of this separation is that it can become difficult to place the links in the correct positions with respect to the joints. There is nothing is the data structure guaranteeing the ends of each link will line up with the appropriate joints, and this must normally be assured by the user of the system. There is no reason that the links themselves must be simple geometrical objects, and in general they are separate graphical hierarchies. So, the entire skeleton data structure can be thought of as a collection of link subhierarchies hanging off of a top level joint hierarchy.

2.4.5 Forward Kinematics

Although articulated skeletons are collections of rigid bodies, we are now dealing with a much larger number of degrees of freedom than with a simple rigid body, and the question naturally arises in animation as to how so many motion parameters can be controlled. It might seem at first approach that it would be fairly easy for an animator to interactively adjust the individual joint angles of a 3D character skeleton to achieve the desired posture or motion, after all, this is just what our own brains do when we move our body. This type of control is called forward kinematic control, and in practice it turns out to be quite difficult. The mapping between joint angle coordinate space of the skeleton and the Cartesian space coordinates of an end-effector, for one thing, is not at all trivial or obvious. Furthermore, the actual articulated motion of a real animal or human results from a complex combination of physics and muscle behavior. It also turns out that we as humans are extremely sensitive to the motion of articulated figures and, like our ability to recognize human faces, this seems to be a pattern recognition skill that our brains have evolved to do particularly well. Therefore, controlling and animating articulated skeletons to the level of accuracy necessary to suspend our sense of disbelief is a very difficult problem.

2.4.6 Rotoscoping

One approach to solving this problem is based on a technique that was used in the Disney studios called rotoscoping. The traditional rotoscoping technique was to project successive frames of a live action film onto paper and trace over the outlines of the human or animal figures as a guide for the animator. For purposes of animating 3D skeletons, this is modified to incorporate multiple camera views of the action, while markers are placed on the important joints of the actor. The resulting parallax views can then be digitized and the joint angle motion inferred from the motions of the markers in three dimensions. This joint angle motion data can then be used to drive a 3D skeleton of similar dimensions to the live actor.

Rotoscoping has obvious limitations: every motion sequence must be filmed first, and the animated skeletons must conform fairly closely to the live-action ones, which does not leave very much room for creativity and caricature. However, techniques have been developed by which some types of periodic rotoscoped motion data, such as walk sequences, can be modified according to functional higher level parameters and can even be made to meet specified constraints [Boulic 92].

2.4.7 Inverse Kinematics

Rotoscoping is generally too inflexible to be used for all but the most realistic forms of character animation. In order to give animators as much creative freedom as possible, it is necessary to use a more general purpose control method. Although forward kinematic control of skeleton joint angles is usually not very intuitive, specification of end-effector motion can be a very natural way to control a skeleton. The analogy here is to a puppet or stop motion clay figure being manipulated by moving, for example, a hand, which causes the entire arm to move accordingly. This type of control is called inverse kinematic control, and is the opposite problem of forward motion control.

Inverse kinematics was also developed originally by the robotics community and normally is applied to articulated chains only. It is possible, however, to apply it to portions of a general articulated figure, and this was first done by Girard in the PODA system [Girard 85, 87] . We can think of the forward and inverse kinematics problem as one of conversion between two coordinate vector spaces. We have on the one hand, joint coordinate space which is represented by a joint angle vector, θ, containing the angles of each joint in the chain, and on the other hand the six-dimensional rigid body coordinate space of the chain's end-effector, $\mathbf{X}$, containing three translational and three rotational coordinates. The problem of forward geometry is, for a given articulated chain, to find $\mathbf{X}$ as a function of θ, . The inverse geometry problem is to find θ as a function of $\mathbf{X}$. Since an articulated chain very often has more degrees of freedom than the end-effector, there can be in general many possible joint vectors that result in the same end effector position. The inverse kinematics problem can therefore have an infinite number of solutions.

While the forward geometry problem is quite easy to solve, simply by multiplying together the local transformation matrices of each joint, the inverse geometric problem is a highly non-linear one, and therefore in general quite difficult to solve. One simplifying approach is to linearize the problem by considering only differential motion. If we move each joint slightly in turn, and observe how it affects the position of the end-effector, we can build up a linear mapping between differentials in joint space and differentials in end-effector space. This differential problem is called inverse kinematics.

2.4.7.1. Jacobian Matrix

This linear transformation can be expressed as a matrix and is called the Jacobian. It is defined as follows:

$$J_{ij}(\theta) = \frac{\partial X_i}{\partial \theta_j}$$

The Jacobian matrix is a multidimensional version of the derivative, the result of differentiating a vector variable with respect to another vector variable, representing in its rows and columns all the partial derivatives of the first vector variable with respect to the other. With it we can convert a small, differential motion in joint space into its resulting motion in end-effector space simply by multiplying the joint angle differential vector by the Jacobian.

$$d\mathbf{X} = \mathbf{J}(\theta)d\theta$$

or in velocity terms:

$$\dot{\mathbf{X}} = \mathbf{J}(\theta)\dot{\theta}$$

where $\mathbf{J}$ is a (non-linear) function of θ. This is simply a differential version of the formula for forward kinematics. What is important about this is that, now that we have a linear operation from joint angle space to end-effector space, albeit a differential one, we can invert the Jacobian matrix to obtain the inverse mapping.

$$\dot{\theta} = \mathbf{J}^{-1}(\theta)\dot{\mathbf{X}}$$

This means that, although we cannot solve the inverse geometric problem directly, we can solve the inverse kinematic problem, so that given an initial joint angle vector, we can calculate its end-effector position and then move incrementally towards the desired position through velocity control. This provides a local solution and eventually a local minimum. Since it is usually more desirable for robots and computer graphics skeletons to move continuously from one point to the next, this is not a significant limitation. This is also well-suited for interactive manipulation of skeletons because with a human in the loop, it is very intuitive to interactively push an end effector towards the desired position.

2.4.7.2. Pseudo-Inverse

This formula only works, however, when the dimensions of the joint angle vector are the same as the dimensions of the end-effector coordinates (i.e. when the Jacobian is a square, non-singular matrix and hence invertable). In general, the number of joint angles is usually larger than the end-effector dimensions, so the Jacobian will be wider than it is high. For example, for the simplified case of a two-dimensional articulated chain with four links, and an end effector whose orientation is not considered, we have four-dimensional joint-angle space and a two-dimensional end-effector space. The inverse kinematic equation can then be written as:

$$\begin{bmatrix} \dot{X}_1 \\ \dot{X}_2 \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} \\ J_{21} & J_{22} & J_{23} & J_{24} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \end{bmatrix}$$

Although the inverse of a non-square matrix is not defined, it is possible to take the pseudo-inverse of such a matrix. Since the problem is underspecified, many possible inverses could be correct, and the strategy is normally to minimize some criterion, such as the mean squared norm. The pseudo-inverse is designated as $\mathbf{J}^+$ so the inverse kinematic problem for non-square Jacobians becomes:

$$\dot{\theta} = \mathbf{J}^+(\theta)\dot{\mathbf{X}}$$

Numerically, the inverse kinematics problem using the pseudo-inverse Jacobian method has two components: calculating the Jacobian and calculating its pseudo-inverse. The problem of finding the Jacobian for a given value of θ is fairly straightforward and can be derived in several ways. [Watt 92], [Fu 87] and [Klein 83] describe some of them in detail. Calculating the pseudo-inverse is a standard linear algebra numerical problem, and an algorithm can be found for it in [Press 92]. One particular problem is handling the ill-conditioning of the solution near singularities. These regions in joint angle space correspond to chain configurations where links are colinear and a small change in end-effector space can cause discontinuous changes of joint angle values. This is commonly handled using a damped least square numerical method, details of which can be found in [Maciejewski 90].

A common extension to the pseudo-inverse control method is to add a secondary task which is optimized over the null-space of the inverse Jacobian. The space of all possible inverses to a non-square matrix is called its null-space. For an articulated chain, this corresponds to all of the chain configurations possible while maintaining the same end-effector position. A robot or animated character might prefer certain of these positions according to some criterion such as minimizing overall torque or avoiding joint angle limits. This criterion is called a secondary task, and the inverse kinematic formula can be extended to optimize this by modifying the equation to add an optimization criterion:

$$\dot{\theta} = \mathbf{J}^{+}\dot{\mathbf{x}} + a(\mathbf{J}^{+}\mathbf{J} - \mathbf{I})\nabla H(\theta)$$

Where a is a weighting function, $\mathbf{I}$ is the identity matrix and H is a scalar optimization criterion function to be minimized. This was first proposed by Liégois and was used in [Girard 85]. More details on this technique can be found in [Klein 83].

2.4.7.3. Skeleton Positioning

Inverse kinematic techniques generally only work on articulated chains. To extend them to manipulation of general skeletons for animation purposes, it is necessary to isolate certain paths of the hierarchy and consider them to be chains. In fact, we can think of the chain as an external inverse kinematic engine which can be attached at will between a joint and any of its descendants [Watt 92]. It can be attached to extremities such as arms, or to higher-level portions of the hierarchy such as the backbone, in which case the non-controlled branches simply follow along as rigid bodies attached to the controlled joints. Multiple inverse kinematic engines can be attached to separate portions of the same skeleton, for example two arms can be bound to the same end effector position so that they move in concert

One problem with this technique is that all skeleton motion must be made relative to the upper levels of the hierarchy, that is, the joint which is highest level in the hierarchy must remain fixed. This often works well because the highest levels in the skeleton hierarchy usually corresponds to the center of gravity of the figure. In certain situations, most notably when a foot is fixed to the floor, it becomes more useful to interchange the end-effector and base of the chain so that, for example, the pelvis is free to move with respect to a fixed foot. It is therefore necessary for the inverse kinematic engine to be able to reverse direction or for the skeleton hierarchy to be reconfigurable so that any joint can temporarily be made the root.

In general, the inverse kinematic technique must be carefully integrated into a general interactive skeleton positioning system to make it practical, and there are many situations where other more domain-specific techniques are necessary, such as simple or higher-level forward kinematic control.

2.4.8 Key Joint Angle Interpolation

Inverse kinematics can be integrated with key-frame animation in two possible ways. The first is to specify the key positions in end-effector space, interpolate a curve through these points and then use inverse kinematics to move the end effector to the interpolated points. This method has several problems, suffering in particular from the fact that the differential inverse kinematics solution is a result of past history, so that to be repeatable, the entire motion sequence must be recalculated from a known initial configuration. This can be a very CPU-intensive task and may not be possible at interactive rates. Another problem is that this technique can be hard to integrate together with other skeleton control techniques, although it has been used for Cartesian-based correction of predefined joint motions [Boulic 92].

A much simpler and more practical way to animate skeletons, which was used in the PODA system [Girard 85] is to use inverse kinematics to interactively position the skeleton in its key positions, and then simply to interpolate the joint angle values using interpolating splines to obtain the animated motion. The interpolated motion can be thought of as a single multi-dimensional spline curve in joint-angle space. Obviously, this space is not flat and, like interpolating Euler angles, will not necessarily result in straight end-effector motion. If the key frame positions are not too far apart in joint-angle space, however, these deficiencies are not very obvious and in practice the technique works quite well. The most notable problems with it are when the end effector must track a moving target, such as a foot on the floor, if the foot is not the base of the skeleton hierarchy.

2.4.9 Other Techniques

A number of other techniques exist, many of them very specialized, for controlling articulated skeletons. Fortunately, almost all of them use the same type of skeleton data structure as a basis, and therefore can usually be easily interchanged and combined. Badler used a technique for skeleton positioning called multiple constraint positioning [Badler 91]. The PODA system [Girard 85] makes use of a variety of kinematic input techniques together with an analysis of legged animal motion and walk cycles.

A natural enhancement to kinematic techniques for animation is to add dynamics to make a more physically realistic simulation. For skeleton motion this necessitates enhancing the skeleton data structure to maintain mass and moment of inertia data for each link, and force and torque information at the joints. Dynamic or physically-based techniques generally fall into two categories: forward dynamics, in which the skeletal motion is determined from specified input torques and forces, and inverse dynamics, in which a skeletal motion is specified as input, and the required forces and torques necessary to achieve that motion are determined. Generally, physically based methods can give very realistic results for skeletons which are moving in a relatively uncontrolled manner (e.g. bodies in free fall). Making the skeleton perform a desired task and still exhibit natural motion is a difficult problem, requiring constraint techniques. Space-time constraint techniques, perhaps the most sophisticated of the physically-based methods, attempt to meet certain specified constraints while minimizing certain global functions such as energy expended. Physically-based skeleton techniques will be discussed in more detail in Chapter 3.

2.5. Deformable Character Techniques

Unlike rigid body techniques, for which the division between modeling and animation is fairly clear-cut, the animation of deformable bodies is more closely intertwined with the type of model used, and the line between animation and modeling becomes more blurred. Unlike hierarchical transformation and skeleton representations, which are fairly standard and accepted as adequate for animating rigid bodies, there exists no such universal technique for representing deformable objects.

2.5.1 Rectangular Meshes

Perhaps the simplest conceptual way to model a deformable surface is to represent it as a three-dimensional function of the surface's local two-dimensional coordinate space (uv-space). The surface can then be discretized in u-v coordinates and each 3D point stored in a rectangular array. This type of representation, called a rectangular mesh, can either be a sampling of a pure surface function, or the actual primary data representation resulting from digitized or interactively sculpted surface data. This is the standard way, for example, that data from 3D laser digitizers is stored. This type of surface representation has several advantages: it has a simple data structure, the connectivity of each point with its neighbors is well-defined, the u-v coordinate system makes it easy to map textures onto the surface, it is mathematically straightforward to work with, it can easily be resized, and is simple to render. Also, it is easy to calculate partial derivatives and other differential geometric quantities.

Unfortunately it also has some disadvantages. Chief among these is that the resolution in *u-v* space is fixed, and it is not possible to locally increase the resolution at regions of high curvature. Secondly, its topology is limited to representing roughly flat or cylindrical objects. It works very well for representing a head, for instance, but not well at all for an entire human body with protruding arms and legs.

2.5.2 Triangular Meshes

Triangular meshes solve both these problems. Arbitrary topologies are possible and triangle size can be reduced as much as desired around regions of high surface curvature. For this reason, triangular meshes remain the most flexible low-level way of representing arbitrary surfaces. However, triangular meshes are more complicated conceptually than rectangular ones and do not have all of their other nice properties. The data structures are more complex, connectivity is more difficult to keep track of, changing surface resolution is more difficult, and the mathematical properties of the surface are much more complicated, since there is no longer a clear u-v coordinate system, which can make it more difficult to apply such techniques as texture mapping.

2.5.3 Polygonal Shape Interpolation

Once we have a surface representation in either a rectangular or a triangular mesh form, animating its shape can be done simply by varying the *x, y* and *z* coordinates of each point as a function of time. In fact, any two surfaces which have the same topology and connectivity can be interpolated simply by interpolating its mesh points in three-dimensional space, so the same sorts of key-framing techniques developed for rigid bodies can be used for mesh surfaces. This type of key-frame shape interpolation of pure polygonal surfaces was used to great effect by Chris Wedge in the *Tuber's Two Step* video [Wedge 87]. This technique works particularly well for interpolating between similar mesh shapes that have been digitized using a 3D digitizer (for example a laser scanner) since these digitized surfaces all have the same topology as a result of the scanning process. This allows a kind of three-dimensional morphing to be performed, where one 3D character transforms smoothly into another, as in the film *Galaxy Sweetheart.*

A variation on this method is to take a series of digitized surfaces and consider them to be set of basis functions from which linear combinations can be made. This works particularly well for facial animation, as in the films *Toney de Peltrie* [Bergeron 86], where a series of facial expressions were digitized from a real person and then combined dynamically to create animated facial expressions. This kind of shape interpolation has recently become a popular cinematic special effects technique used, for example in the film *Terminator II*.

2.5.4 Spline Surfaces

Both rectangular and triangular meshes suffer from a common problem: they have a fixed surface resolution. In other words, the closer you zoom in on a mesh surface, the more it resembles what it really is--a discrete polygonal object--and the less it resembles a smooth surface. In order to have a truly resolution independent representation of a curved surface, it is necessary to use some kind of continuous function, which can be polygonalized to whatever resolution is necessary for display purposes. The shape of the surface is then controlled by a relatively small set of parameters which do not depend on the resolution. Commonly, there are two kinds of mathematical surface representation techniques used: implicit representations and parametric representations.

Parametric surface representations are usually based on extending ordinary splines, which are functions of a single parameter, into spline surfaces, which are functions of two parameters, much like the rectangular mesh. These are also called tensor product surfaces because they can be defined using a double tensor product:

$$\overset{\vee}{S}_M(u+i, v+j) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \mathbf{M} \begin{bmatrix} P_{i,j} & P_{i,j+1} & P_{i,j+2} & P_{i,j+3} \\ P_{i+1,j} & P_{i+1,j+1} & P_{i+1,j+2} & P_{i+1,j+3} \\ P_{i+2,j} & P_{i+2,j+1} & P_{i+2,j+2} & P_{i+2,j+3} \\ P_{i+3,j} & P_{i+3,j+1} & P_{i+3,j+2} & P_{i+3,j+3} \end{bmatrix} \mathbf{M}^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

where u and v are the two parametric surface coordinates, $P_{i,j}$ are the rectangular mesh of control points, and M is the spline matrix. Common types of tensor product surfaces are Bézier surface patches, which interpolate their (alternating) control polygons but do not guarantee continuity, Catmull-Rom patches which do guarantee continuity, and B-Spline surfaces, which guarantee at least C^2 continuity but do not interpolate their control points.

Although spline surfaces are resolution-independent in the sense that they are guaranteed to generate a smooth curve, they are not necessarily resolution-independent from the point of view of deformation or animation. Animating a spline surface normally is accomplished by animating the control points. Since moving a single control point will change the shape of the surface within the entire region surrounding it, we can see that our ability to control and animate a spline surface is limited by the resolution of the control point mesh. In fact, for animation purposes ordinary spline surfaces suffer from many of the same problems as rectangular meshes, including inability to control resolution locally and limitations on surface topology.

In the worst case, therefore, spline surface animation techniques can be considered little more than a way of interpolating smoother surfaces through a rectangular mesh surface representation. Nonetheless, it can be a very effective technique. The baby in Pixar's *Tin Toy* film was animated using several thousand Catmull-Rom spline patches, the control of which required a hierarchical structure.

One way to attack the resolution problems of the spline control mesh is to subdivide certain portions of the mesh which require higher resolution into finer meshes. This type of technique was used by Forsey and Bartels to create a type of refinable spline called the hierarchical B-Spline [Forsey 88]. This allows any portion of the rectangular mesh of control points to be replaced with a mesh at twice the resolution, thus allowing regions of high curvature to be represented at a higher resolution without subdividing the entire mesh. This technique shows promise for animating deformable surfaces, although it does not overcome the topological restrictions of the rectangular mesh.

There are several promising research directions being explored currently to solve the topological problems of parametric surface representations. A B-Spline surface of arbitrary topology, based on S-patches, was proposed by Loop and deRose [Loop 90], and the Cao En Surface allows for the creation of surfaces with spherical topology [Wyvill 92].

2.5.5 Implicit Surfaces

Implicit surfaces are an alternative to parametric methods for representing surfaces mathematically. Rather than being functions of an independent set of parameters, implicit surfaces are defined by a single scalar implicit inside/outside function of the spatial coordinates x, y, and z:

$$f_{io}(x, y, z)$$

Points for which this function is equal to one (or some other predefined value) are said to be on the surface. For example, the implicit function for a sphere is simply:

$$f_{sphere}(x, y, z) = x^2 + y^2 + z^2$$

This type of representation has certain advantages over parametric ones. Since there is no parametric coordinate system, topological restrictions are not a problem. The surface always represents a geometrically well-defined solid, since it divides space into three regions: inside, outside and on the surface, and it is very easy to test if a point is in one of these regions. This can be useful for collision detection. It is also very easy to determine the normal to the surface at any point because this is proportional to the gradient of the inside/outside function. Most interestingly, implicit surfaces can be summed together to form combinations, which are themselves implicit surfaces.

Implicit surfaces have several disadvantages, however. There is no straightforward way to render them, since there is no parametric coordinate system to serve as a basis for sampling. They can usually be ray-traced directly, but polygonalization generally must be done using a three-dimensional space sampling technique such as the marching cubes algorithm which can be very time consuming.

2.5.6 Superquadrics

One family of implicit surfaces is particularly interesting because, among other things, it has a parametric representation as well as an implicit one. These are the superquadrics which were developed by Alan Barr [Barr 81]. Superquadrics, of which the subset called superellipses are probably the most useful, and can be expressed parametrically as functions of the spherical parameters η and ω (latitude and longitude, respectively). The formula for a superellipse is:

$$\overset{\vee}{S}(\eta,\omega) = \begin{bmatrix} a_1 \cos^{\varepsilon_1} \eta \cos^{\varepsilon_2} \omega \\ a_2 \cos^{\varepsilon_1} \eta \sin^{\varepsilon_2} \omega \\ a_3 \sin^{\varepsilon_1} \eta \end{bmatrix}$$

where a_1, a_2 and a_3 are the scaling parameters in the x, y and z dimensions respectively and the ε_1 and ε_2 exponents are the two shape parameters which control the degree of squareness of the superellipse in the lateral and longitudinal directions respectively. When these are set to one, the superquadric is identical to an ellipsoid with dimensions controlled by the a parameters. As the ε_1 and ε_2 parameters increase, the surface becomes flatter and then more beveled. As they decrease below one, it becomes more square. Superellipsoids have the following inside-outside function.

$$f_{\text{supere}}(x,y,z) = \left[\left(\frac{x}{a_1}\right)^{2/\varepsilon_2} + \left(\frac{y}{a_2}\right)^{2/\varepsilon_2}\right]^{\varepsilon_2/\varepsilon_1} + \left(\frac{z}{a_3}\right)^{2/\varepsilon_1}$$

When this is equal to one, the point (x,y,z) is on the surface of the superellipse. Superellipses provide a useful set of solid primitives which can be varied continuously from ellipsoids through cylindroids to nearly cubical shapes. They are particularly useful when used in conjunction with space deformation techniques. Barr also provides formulas for the normal to the surface, which is itself a superellipse in normal space.

2.5.7 Soft Objects

Since they are defined by a scalar function of three-dimensional space, implicit surfaces can therefore be thought of, via the line integral theorem, as equipotential surface contours of some kind of conservative force field, much like the equipotential surfaces of an electrostatic field resulting from a collection of charges. Jim Blinn was the first to recognize the computer graphics application of this type of surface and used it to represent molecules as implicit surfaces formed from summing the individual surface functions of the molecule's constituent atoms [Blinn 82]. If we think of these atoms as control points, and their individual force field potentials as blending functions, then we can construct a general class of surfaces which can be manipulated in a manner similar to parametric ones.

This technique was developed by Geoff Wyvill, who introduced the term *soft objects* [Wyvill 86, 89], and by Nishimura et al [Nishimura 85], for animation and CSG modeling purposes. This can be made more powerful by allowing for negative potentials, which allow control points to subtract as well as add to the overall surface potential function. Unfortunately, while parametric surfaces tend to interpolate or at least approximate their control points, soft object surfaces tend to avoid their control points altogether and modeling surfaces with them can be quite difficult and non-obvious. However, their ability to interpolate smoothly between surfaces with varying topologies and guaranteed solid surface properties make them quite attractive for modeling certain types of shape.

2.5.8 Global Deformations

An alternative approach to animating surface models directly is to animate global deformations of space instead. Global deformations are mappings of three-dimensional space into three-dimensional space, in other words, a distortion of space, and can be thought of as three-dimensional extensions to two-dimensional image warping techniques. Any surface embedded within this space will also appear to deform along with it. By animating the deformation of the space around a surface, the surface itself can be animated. This has some attractive advantages over animating the surface directly. It can be applied to any type of surface model (at least in its final polygonal form), and it provides a clear separation between the modeling and animation operations, as exists for rigid bodies. In fact, global deformations can be thought of as an extension to the affine transformations that can be represented with 4x4 homogeneous transform matrices.

Global deformations were developed for computer graphics by Alan Barr [Barr 84]. He defined a collection of simple, intuitive non-linear deformations which could be applied to space in a hierarchical manner. A global deformation can be defined as a mapping from undeformed space, expressed in *(x, y, z)* coordinates, to deformed space, expressed in *(X, Y, Z)* coordinates. Using this notation, a simple scaling operation can be expressed as:

$$X = a_1 x$$
$$Y = a_1 y$$
$$Z = a_1 z$$

Barr defined a tapering operation along the z axis as:

$$r = f_{taper}(z)$$
$$X = rx$$
$$Y = ry$$
$$Z = z$$

A twist along the z axis results from:

$$\theta = f_{twist}(z)$$
$$X = x\cos\theta - y\sin\theta$$
$$Y = x\sin\theta + y\cos\theta$$
$$Z = z$$

While a global bend around the y axis is defined as:

$$\theta = \begin{cases} k(y_{\min} - y_0), & \text{if } y \le y_{\min} \\ k(y - y_0), & \text{if } y_{\min} < y < y_{\max} \\ k(y_{\max} - y_0), & \text{if } y \ge y_{\max} \end{cases}$$

$$X = x$$

$$Y = \begin{cases} -\sin\theta(z - \frac{1}{k}) + y_0, & \text{if } y \le y_{\min} \\ -\sin\theta(z - \frac{1}{k}) + y_0 + \cos\theta(y - y_{\min}), & \text{if } y_{\min} < y < y_{\max} \\ -\sin\theta(z - \frac{1}{k}) + y_0 + \cos\theta(y - y_{\max}), & \text{if } y \ge y_{\max} \end{cases}$$

$$Z = \begin{cases} \cos\theta(z - \frac{1}{k}) + \frac{1}{k}, & \text{if } y \le y_{\min} \\ \cos\theta(z - \frac{1}{k}) + \frac{1}{k} + \sin\theta(y - y_{\min}), & \text{if } y_{\min} < y < y_{\max} \\ \cos\theta(z - \frac{1}{k}) + \frac{1}{k} + \sin\theta(y - y_{\max}), & \text{if } y \ge y_{\max} \end{cases}$$

Barr also provides formulas for calculating the normal vector transformation as well, which can be useful for lighting calculations, for example. Global deformations provide a small set of intuitive deformation parameters which can be animated just like any other parameter in a key frame animation system. They also can be arranged in hierarchies, which can be thought of as subjecting the space to a series of deformations, resulting in operations such as a tapered twist, or a twisted taper.

2.5.9 FFDs

Although Barr's global deformations can generate a large variety of deformation effects, they are normally not general-purpose enough to animate all of the subtleties of a three-dimensional character. In particular, they lack the sort of local control associated with parametric or polygonal mesh surfaces. A more general-purpose technique based on control points was developed by Sederberg which he called free-form deformations or FFDs [Sederberg 86]. Conceptually, we can think of free-form deformations as being analogous to placing an object inside a cubical piece of jello, in which is embedded a cubical lattice of control points. The control points are then moved, deforming the shape of the jello and correspondingly the shape of the object. This process is directly analogous to control-point-based 2D image warping techniques.

Mathematically, the FFD deformation mapping can be expressed as a vector-valued trivariate Bernstein polynomial:

$$\vec{X}_{\mathrm{FFD}} = \sum_{i=0}^{l} \binom{l}{i} (1-s)^{l-s} s^{i} \left[\sum_{j=0}^{m} \binom{m}{j} (1-t)^{l-s} t^{i} \left[\sum_{k=0}^{n} \binom{n}{k} (1-u)^{l-s} u^{i} \vec{P}_{ijk} \right] \right]$$

where $\check{X}_{\mathrm{FFD}}$ is the deformed coordinates in Cartesian space, *(s, t, u)* are the undeformed coordinates of $\check{X}$ in the local lattice frame, *(l, m, n)* are the number of control points in the lattice for each dimension, and $\check{P}_{ijk}$ are the control points in Cartesian space.

Very successful 3D character animation of a certain style can be achieved using FFDs to deform a 3D character shape and animating it using key frame interpolation of the control point positions. Traditional effects such as squash-and-stretch, anticipation and follow-through, as well as certain types of emotional expression can be accomplished this way. It is particularly appropriate for animating "inanimate" objects such as cars, toasters, etc. which are not expected to move in as sophisticated a way as, say, an animal would. FFD-based animation has been implemented commercially in TDI's *Explore* Software.

Two extensions to FFDs have been developed by Coquillart. The first, called extended free-form deformations or FFDs extend the cubical control point mesh to a variety of different shapes, for example, a cylindrical arrangement [Coquillart 90]. This permits a wider range of possible deformations. The second, called animated free-form deformations, or AFFDs, allow the control point lattice to be moved freely about the space which it is deforming, causing the object to deform in an animated way as it passes through the space deformation [Coquillart 91].

2.6. Layered Construction

Both surface deformation and global space deformation techniques suffer from two common deficiencies when applied to character animation: they do not take into account the internal structure of the character, and they do not model the dynamic physical behavior of moving objects. They therefore tend to work well for highly stylized types of animation, where the viewer doesn't particularly expect the character to have any internal structure, or for very realistic sorts of animation based on digitized shapes, where the implied internal structure comes from the real world.

Most commercial animated characters, however, are based on humans or anthropomorphic versions of animals, whose overall appearance is very much influenced by their internal skeletal and muscle structure. For designing such animated characters, a common approach taken by artists and traditional animators is to work in layers. First a stick figure is drawn, representing the skeleton, followed by rounded forms to represent the flesh, followed by the finished outline, representing the skin [Culhane 88]. This same sort of approach is taken in clay animation, where plasticene is wrapped around a metal armature. It is therefore natural to consider that computer animated characters should also be constructed in layers.

2.6.1 Skeleton and Envelope

The first attempts at designing layered animated characters focused on trying to reconcile the two very different ways of modeling a character: as a rigid skeleton and as a skin surface envelope. Since a real (vertebrate) animal skeleton lies entirely within its skin, this conceptually leads to the simplest form of layered construction which consists of an inner skeleton layer and an outer skin layer. As with a real animal or clay animation figure, moving the skeleton should cause the skin layer to move along with it and to deform appropriately.

There are two basic approaches to building this kind of a layered model. In the first, which can be thought of as fleshing out the skeleton, a skin surface is sculpted around an existing skeleton model. This is a natural extension to the process of adding rigid links, and is perhaps most appropriate for an artistic person interactively building up a more stylized character from scratch. In the second approach, which might be termed skeletonizing the skin, a skeleton is created to fit inside an existing skin surface model. This is appropriate for more realistic types of human character modeling, where the skin surface is obtained from digitized data. In either case, we are presented with a difficult problem from a surface modeling as well as data representation point of view. Whereas before we had, on the one hand a strictly hierarchical arrangement of joints, and on the other a mesh of points with local connectivity, we are now presented with a more complicated thing: a kinematic hierarchy terminating in a set of leaf node points which are themselves connected to each other locally.

2.6.2 Construction vs. Animation

One of the major advantages of layered character models is that it allows the animation process to be divided into two stages: character construction, in which the behavior of the layers and attachment to the skeleton is defined, and character animation, in which only the skeleton motion is specified. The outer layers then derive all their input from the skeleton motion alone, greatly simplifying the animation process. Such an approach to character animation works because so much of a character's skin shape and behavior is determined solely by the position of its skeleton, but it does have its limitations. There is obviously a great deal of expressivness visible on the surface of an animated character which is conveyed by muscles not involved in moving the skeleton, or which results directly from the stylization of the character itself. However, these animation problems can be attacked using other techniques (e.g. FFDs).

2.6.3 Deformation at Joints

The first and most obvious problem to be solved with two-layered character models is how to maintain smooth deformation of the surface at the articulated figure's joints. For polygonal mesh surface models, any point on the skin surface can be bound to a link in the skeleton hierarchy simply by placing it in the link's local coordinate system. However, this is no different from attaching rigid links to the skeleton and where the skin surface makes a transition from one link to another, there will be a discontinuity, resulting in a robot-like effect when animated.

One solution to this problem is to use specific local deformation routines, called joint-dependent local deformation operators or JLDs. These were used to implement joint deformation in the film *Rendez-vous à Montréal*, [Magnenat-Thalmann 87, 88]. This technique, which uses a polygonal mesh skin envelope digitized from a sculpture, assigns each vertex point to a specific joint on the skeleton. Deformation is then implemented by writing specialized procedures for each joint which deform the surface as a function of joint angle. Similar techniques are used in the *Softimage* commercial animation system.

Higher-level control schemes have been used for polygonal mesh skin surfaces, in particular for animating human faces. Parke did some of the earliest work in facial animation using parameterized models of facial expressions [Park 75]. Later work in facial animation has been done by Platt and Badler [Platt 81], the Thalmanns [Magnenat-Thalmann 88] and Keith Waters [Waters 87], among others.

2.6.4 Spline Surface Animation

Using spline surfaces to represent the skin envelope provides resolution independence and can help to smooth out the larger discontinuities at the joints, but control points still need to be moved in the regions around the joints and to implement facial expressions. Pixar used Catmull-Rom splines to interpolate a rectangular mesh digitized from a clay model of baby for the film *Tin Toy* [Reeves 90]. They then used a variety of higher-level control schemes to control the deformation of the skin, including a geometric "shrink wrap" algorithm to warp the skin around an underlying cylindrical link structure at the joints, and an abstract muscle model similar to the one developed by Waters [Waters 87] to animate the face.

2.6.5 Hierarchical B-Splines

Spline techniques are still limited by the topological restrictions of the rectangular mesh, which make it difficult to represent an animated character with thin arms and legs. Forsey addressed this problem, and the joint deformation problem, by using hierarchical B-splines with control points attached to the various links of a skeleton for modeling animals and human joints. By attaching the control points in a hierarchical manner, he was able to use the interpolating properties of the spline surface itself to implement smooth joint deformations [Forsey 91]

2.6.6 Blobbies

A rather different approach to attaching a skin surface to a skeleton is to use soft objects, sometimes called blobbies or metaballs. If we attach multiple soft object control points to the skeleton and adjust their strengths appropriately, a sculpted implicit surface envelope can be created around the skeleton. Since the skin envelope is an isosurface of a continuous function of space, it will smoothly deform automatically as the joints are moved [Wyvill 86],[Bloomenthal 90]. This technique was used in the film *The Great Train Rubbery*, and is used in the *Softimage* animation system. Pacific Data Images used a blobbie technique to animate the singing globs of toothpaste in the Crest Sparkle Singers television advertisement [Gould 89].

2.7. Physically-Based Deformation

An major limitation of all the deformation models described so far is that they are purely kinematic or geometric models. However, real characters exist in the real world and must obey the laws of physics. Much of the appeal and lifelike quality of traditional animation comes from a proper or exaggerated use of physical behavior. For example, heavy objects will accelerate more slowly than light ones, and flexible objects will deform in volume-conserving ways when they collide. In traditional animation, the burden of making the motion appear physical is placed on the animator, but since physical laws are well-understood and can be represented mathematically, it would seem reasonable that the computer should be able to take over this task and automatically generate physically realistic motion. This technique is known as physically-based modeling, which will be covered in Chapter 3. In Chapter 4 we will discuss how physically-based models can be applied to character animation in the form of layered elastic models.