Navigating the Animation Jungle

Brian Wyvill

Department of Computer Science, University of Calgary.
2500 University Drive N.W.
Calgary, Alberta, Canada, T2N 1N4

Abstract

This tutorial paper presents an overview of 3D computer animation with an emphasis on motion control. A taxonomy of various motion control techniques is presented and examples are given. Also presented is the use of implicit surfaces (known as Soft Objects), for modelling and animation. The usefulness of this technique as applied to animation is shown along with a number of examples.

Key words: Computer Graphics, animation, motion control, implicit surfaces, soft objects.

Figure 1: *Graphicsland* made from polygons, particles and fractal mountains

Part I
Computer Animation

1 Introduction

In the last ten years, computers have become sufficiently fast and inexpensive to allow a computer animation industry to grow from almost zero to 150 million dollars in 1987. During this time the technology has advanced to permit the making of landmark animated shorts, from Peter Foldes' "La Faim" (1974), with software by Nestor Burtnyk and Marcelli Wein [Burtnyk *et al* 76], to John Lasseter's "Luxo Junior" (1986) [Lasseter 87], with software by Bill Reeves *et al* from Pixar. The skeleton techniques used in "La Faim" were essentially 2D. The method employed extends the traditional animation concept of "key framing". When using this technique, an animator draws a character posed in two or more "key" positions, and uses the computer to generate interpolated drawings in the frames spanning between them. By contrast, "Luxo Junior" uses 3D computer animation, in which digital models of the characters and backgrounds are described in three dimensional space and are moved using a variety of techniques, such as by simulating the dynamics of each motion. The common thread binding these two animated pieces is not that a computer was used to produce the pictures, but that they both succeed as animated shorts because the artists who made them share the same skills that have always distinguished good animators. The computer has allowed them to produce their art in an exciting new form, and has opened the door to many other enthusiasts who may have the ideas and the ability to breathe life into a character, but lack traditional draughting skills. Although there are several commercially available animation systems, the best computer animated films are generated by a team consisting of at least one animator and one computer scientist. Regardless of how sophisticated and general-purpose the tools are that an animation package provides, animators often want to go beyond the capabilities offered, and require a programming solution. The technology is advancing extremely quickly: hardware becomes faster and cheaper, and the volume of literature describing new software techniques grows alarmingly each year. An animator buying a system today could be put out of business by his competitor who has tomorrow's system.

There are many applications of computer animation in industries other than entertainment and advertising. Flight, and other, simulators have attained a fair degree of sophistication, responding with real-time graphics to a user's inputs. Scientific visualisation uses the techniques of computer animation to present complex data in a more pleasing and intelligible form. Such applications are intimately linked with simulation, and techniques from that discipline are generally useful in others.

In this tutorial, the fundamentals of 3D animation are presented with particular emphasis on motion control, or how to make models move. Of course, the work of an animator will always be judged by the final result, but a working knowledge of existing tools is a means to help him to convey the vision in his mind to the big (or little) screen, with less tedious effort.

1.1 The elements of 3D Computer Animation

In traditional animation, the end product is a series of cells which are overlaid and photographed for each film frame, often with added optical effects, such as pans and zooms, added at the camera stage. Reaching this final production requires many steps, including designing characters and backgrounds, and prototyping movements with flip books or by filming rough

sketches (known as "pencil tests"). Key frame drawings are made; the inbetween segments are drawn and tested; this leaves a series of pencil drawings which must be inked and painted, each character possibly being done in isolation on sheets of clear acetate; backgrounds are painted; and the results are composited and photographed.

Model animation involves the building of a miniature world. High-tech cameras with periscope lenses are used to photograph the models, giving them a sense of scale. Every movement of each character must be carefully calculated and the model changed with skill and care. Like model animation, 3D animation also requires that the artist create a 3D world. Each character and background object must be crafted using one of a variety of digital techniques.

As with model animation, specifying the motions of these computer models is the most complex operation to be carried out, and the one for which the least amount of packaged software is available. The computer animator has the advantage that the digital equivalent of a camera is weightless, sizeless, and can be moved along any desired path. Each frame of the 3D scene must be rendered: a view is chosen and the 3D model is projected as a 2D shaded image. This operation has been the subject of much research and there are many techniques available for it. Usually, rendering algorithms present a trade-off between picture quality and computation time. There are steps beyond rendering: transferring the finished images to film or video, and subsequent editing. Since complex animation frames at high resolution cannot, in general, be manufactured in real time, this last step requires some special hardware and a talented film or video editor.

To manufacture the finished animation frames requires the following major steps:

- Modelling
- Motion Control
- Rendering

Figures 2, 3 and 5 summarise these three areas, showing a taxonomy of each. The following sections present brief descriptions of trends in modelling and rendering, and a somewhat more detailed review of motion control. In the second part of this tutorial, a modelling method which uses implicit surfaces is described, and the way in which this technique is particularly appropriate for certain kinds of animation is shown.

2 Modelling

Building models for computers to display is a natural application and was accomplished in the 1950's on vector displays and graph plotters. Any mathematical function which produces points on a surface in 3-space could potentially form the basis of a modelling method. However, with many such techniques it is difficult to achieve the desired shape and also difficult to render the resultant surfaces. Since this tutorial is mainly concerned with motion control, only a brief explanation has been given for the major areas of modelling. Figure 2 shows various different classes of modelling techniques. There are three main categories:

- Surface Modelling
- Solid Modelling
- Generative Processes

2.1 Surface Modelling

As the name implies, in surface modelling, information is stored about the surface of the model. A point in space is either on the surface or it is not. A polygon mesh is an example of such a

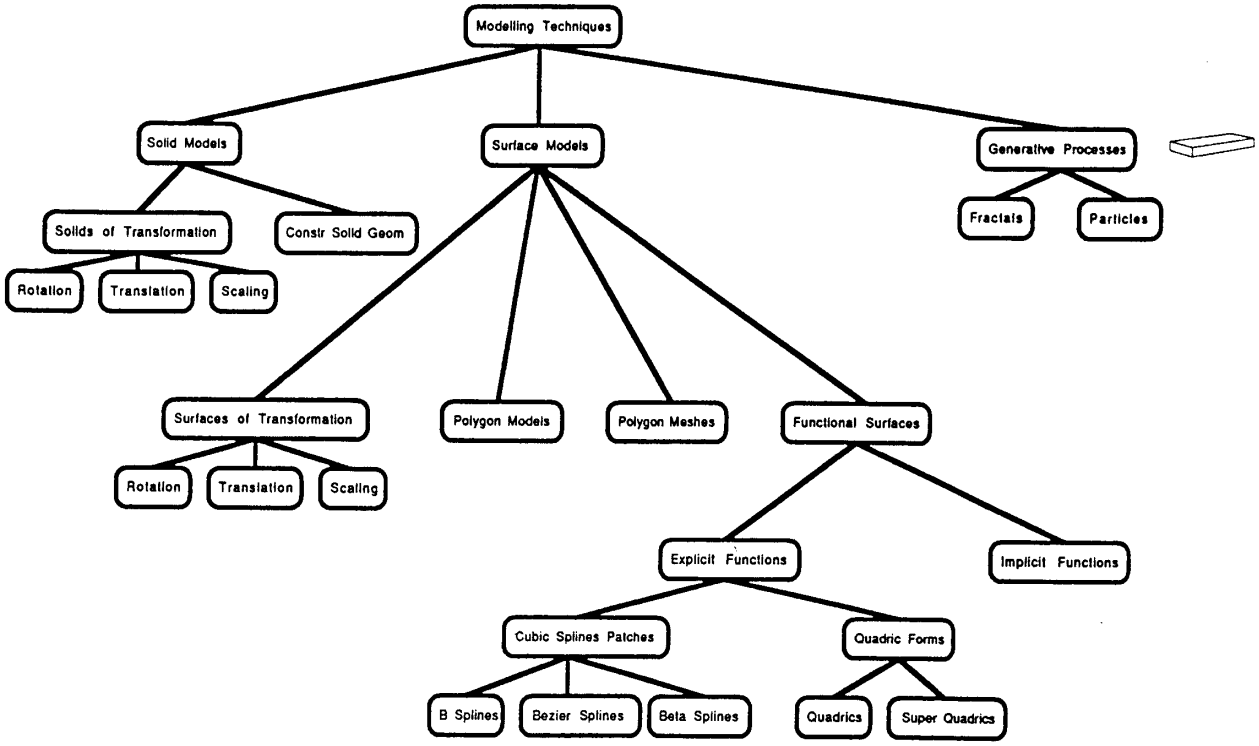


Figure 2: A Taxonomy of Modelling Techniques

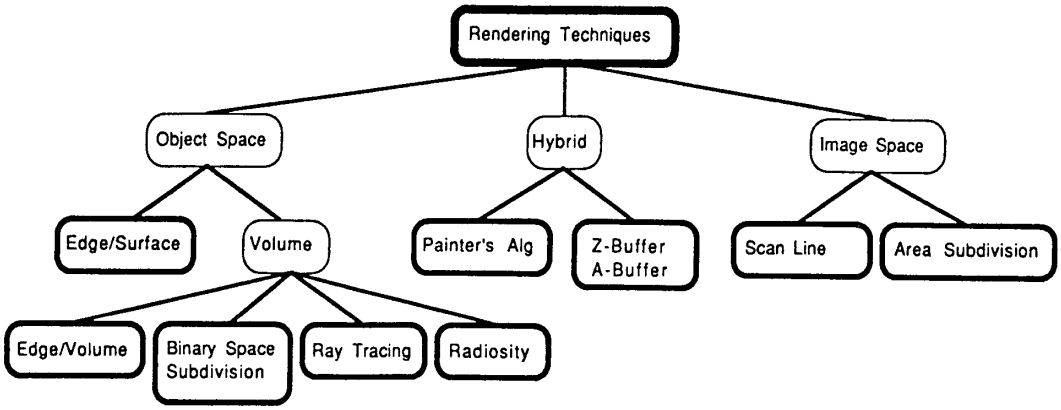
model. Some objects which have many curved surfaces are best modelled using curved spline patches. Patches are based on parametric cubic curves. Some of the latest work done in this area permits the user to have fine control over the shape of the patch; smooth curves, and sharp changes in slope can be described using beta splines (see [Barsky 81]). This technique allows the user to make a curved surface patch and then smoothly join it to others, forming a solid object.

2.2 Solid Modelling

In solid modelling, models are described by the volume of space which they occupy. A solid model has an inside and an outside. CSG or Constructive Solid Geometry is a solid modelling technique where objects are built from primitives that are usually defined as quadrics. (Sphere, cylinder, plane etc.) The primitives can be combined: for example, to make a hole through a sphere, a cylinder is subtracted from the sphere. CSG systems are often implemented using an octree data structure. Implicit surfaces, or SOFT objects, are different from parametric surfaces because they have a different mathematical definition, discussed in part two of this tutorial. Like patches they are defined by a set of keys, but unlike patches they have the property that they can be blended merely by placing their " keys" near each other. Part two presents methods for modelling and animating these surfaces.

2.3 Generative Processes

These can apply to either of the above categories: models can be generated by some algorithmic process, as with fractals [Mandelbro 77], or particle systems [Reeves 83]. A recent example of



LIGHTING MODELS

Constant Shading [Bouknight,80]
Colour Value Interpolation [Gouraud,71]
Surface Normal Interpolation [Phong,75]
Ray Tracing [Witted,80][Cook,82][Cook,84]
Radiosity [Cohen,85][Cohen,86][Greenberg,86]

TEXTURE MAPPING

2D Textures [Catmull,74][Blinn,76][Blinn,78][Feibush,80]
3D Textures [Perlin,85][Peachey,85][Wyvill,87]

Figure 3: A Taxonomy of Rendering Techniques

a generative process which has been used to produce 3D models is an extension of Conway's life game to 3D [Thalmann 86].

2.4 Composing Models

It may be appropriate to build one model from a variety of primitives. This is usually accomplished by structuring the model as a hierarchy, as the *Graphicsland* system does (see [Wyvill *et al* 86a]). Figure 1 shows a tree built, using a recursive hierarchy, from polygons, with particle systems as leaves. Fractal mountains appear in the background.

3 Rendering

During the 1970's and 80's, much research was put into the development of rendering algorithms. The main obstacle is that if a scene consists of n objects, then a naive algorithm will compare each object with every other object to check if it is occluded from a particular viewpoint ($O(n^2)$). If n is large, this is a time consuming task. The class of rendering algorithms whose time complexity depends purely on n are known as *object space* algorithms. Another class of rendering algorithms work in *image space*, where the screen resolution, or number of pixels, becomes the dominant factor (see [Sutherlan 74]).

One early algorithm simulates the paths of light rays: the *ray tracing* algorithm sends out rays from the eye point through the scene, and finds the intersections between the ray and

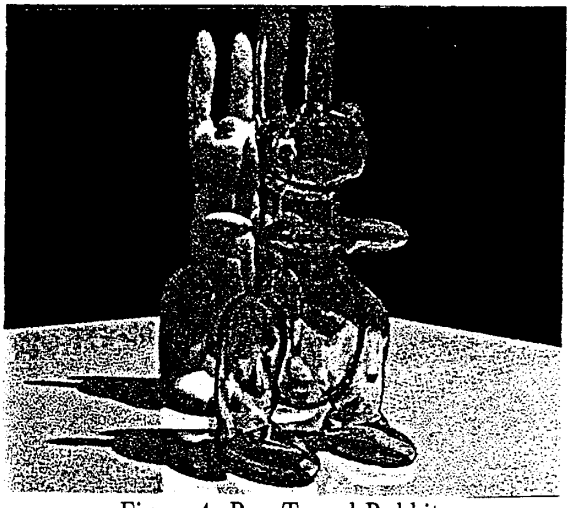


Figure 4: Ray Traced Rabbit

any object on its path. Ray tracing has the benefit that rays can be traced recursively, as they are transmitted through transparent objects and reflected from shiny ones. Rays can also be traced from an object intersection point towards the light source(s); if an object is on the ray path then the intersection point is in shadow. (Figure 4 shows a ray traced rabbit with shadows.) Early versions checked all m rays against all n objects in the scene. Since m can be tens of thousands, or millions, for anti-aliased scenes, the algorithm is slow. One way of speeding up ray tracing is to reduce the number of intersection calculations on a ray's path, either by surrounding each object, or hierarchy of objects, with a bounding volume, or by subdividing the scene space. In the latter case, the volume of the scene is split into cubic cells (*voxels*), and the objects are sorted into the voxels which contain them. (Some objects will appear in more than one voxel.) Each ray is intersected with each voxel that it must pass through and, if any objects are present in the voxel, only then is it necessary to check for intersections of the ray with those objects. Since many objects may be clustered in a few voxels, further subdivision may take place; in this case the voxel is often stored as an octree data structure. [Glassner 84], [Kay *et al* 86], [Cleary *et al* 87].

To manufacture highly realistic scenes great attention must be paid to the lighting model employed. An object may be partly illuminated by light reflected from other objects as well as by primary light sources. Most lighting models account for reflected light by adding a constant ambient light value into the illumination of every point. Recent work employs a technique known as *radiosity* [Immel *et al* 86], in which each surface is broken into patches, and the illumination effect on each patch by every *other* patch in the scene is calculated. This technique produces excellent results but requires large amounts of computer time.

Surface detail is often added during the rendering stage rather than by trying to model small details. Early work on *texture mapping* was done by [Catmull *et al* 80] and [Feibush *et al* 80], in which 2D images were mapped onto 3D objects. Later work [Perlin 85], [Peachey 85], and [Wyvill *et al* 87] uses 3D functions to specify the colour or direction of the reflection vector (surface normal) at any point in space. Figure 12 shows conventional textures on the walls and 3D "wood" texture on the four posts of the bed. A combination of sine waves with amplitude, frequency and phase modulation in x, y and z was used to create the 3D texture space in which the wood exists. For a full treatment of rendering algorithms, see the bibliography at the end of this paper.

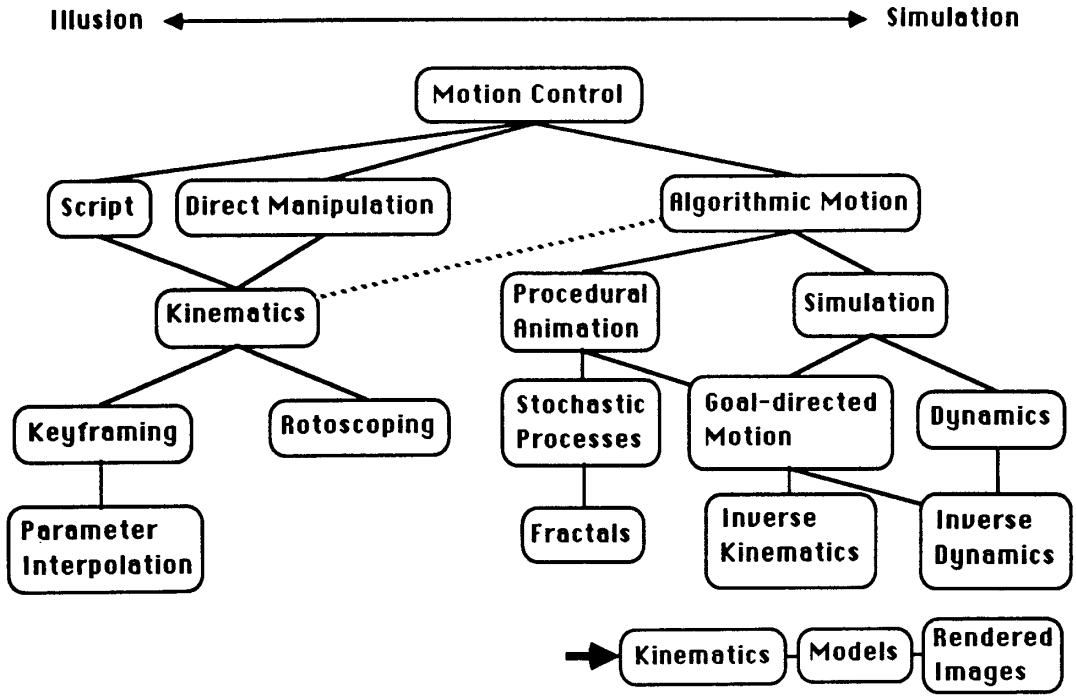


Figure 5: A Taxonomy of Motion Control Techniques

4 Animation - Controlling the Motion

To animate, we wish to simulate the appearance of motion of real objects. This motion control falls roughly into two classes:

- Simulation
- Illusion

To look natural, animation has to represent possible motion in the physical world. In this sense, the best animation is based on detailed simulation which accounts for the dynamics of the action. In such a simulation, a mathematical model representing the physical laws that govern the motion would, as a consequence, produce the desired effect. In many applications it is not necessary for an animation sequence to be an accurate dynamic model; unless the animation is intended to visualise the results of a simulation, it is only required to make the motion *appear* realistic to a human viewer. The technique of *faring* in hand animation can be regarded as a crude attempt to use a few simple rules to do this. Faring creates the *illusion* of acceleration that would be present in an accurate dynamic *simulation* of the same motion. The taxonomy of motion control (see figure 5) portrays this distinction.

At the lowest level, 3D Computer animation is based on specifying the position and orientation of an object at some time. This *kinematic* description of animation can be manufactured by a whole variety of techniques. At the highest level an animator may wish to give a command such as "walk to the door", it would then be up to the system to calculate how that walk is to be achieved, and finally produce the kinematic description for the low level part of the system (inverse kinematics). Such *goal directed* animation is the subject of much current research with applications in robotics as well as the entertainment/advertising animation industry. At an

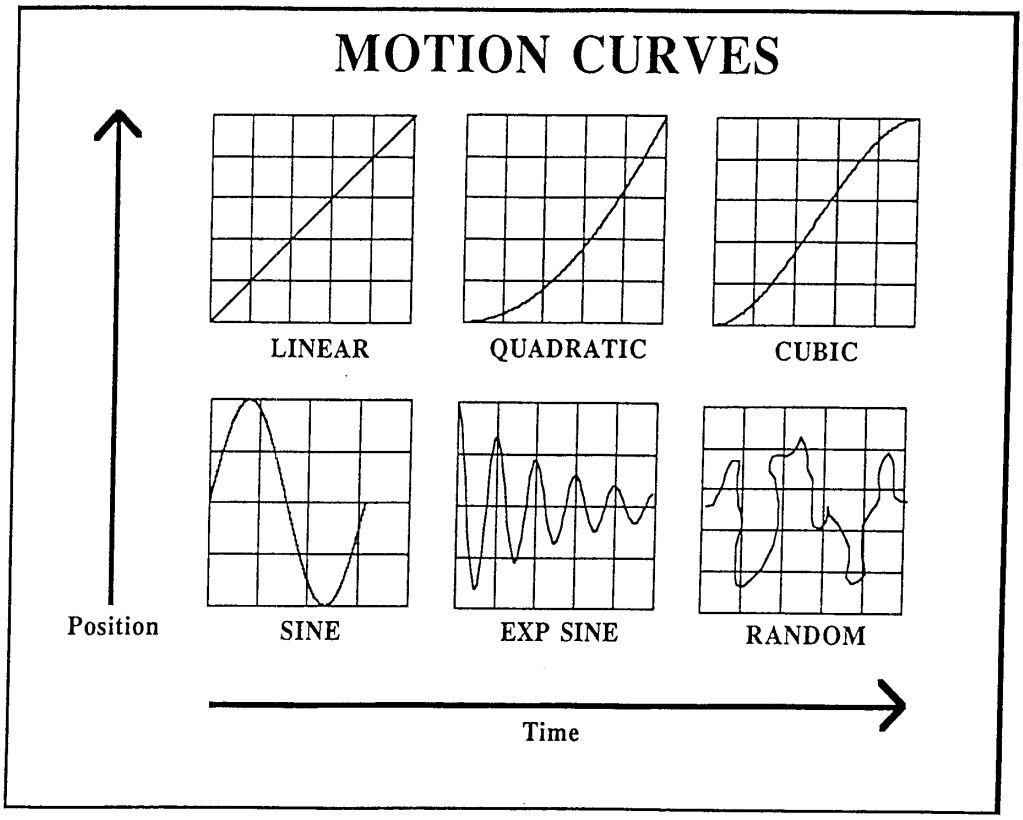


Figure 6: Sample Motion Curves

intermediate level an animator may wish to show a falling human, a somewhat easier task to calculate than walking, since this problem only has one solution if enough initial information is specified. Given the constraints and initial condition, the system can then formulate a set of differential equations that describe the fall. This *dynamics* approach should produce an accurate simulation of the motion, provided the dynamic model is sufficiently detailed. To illustrate the difference between a dynamic simulation and the kinematic approach (aided by a script file or interactive system) consider a bouncing ball. The animator can define a ball and move the position of the ball along a a damped sine curve as illustrated in figure 6. The dynamic solution would calculate the position of the ball from the Newtonian equations:

$$v^2 = u^2 + 2gss = ug + 0.5it^2 \qquad g \text{ is } 9.81\text{m/sec}^2$$

Provided the initial conditions were known this very simple dynamic model can look quite reasonable except the motion will not cease very accurately. In fact the physical model used is incomplete. No term has been added for air resistance or to account for the material that the ball is made from. These factors will cause the motion to damp out. An example of using a dynamic solutions applied to human animation is given in section 5.

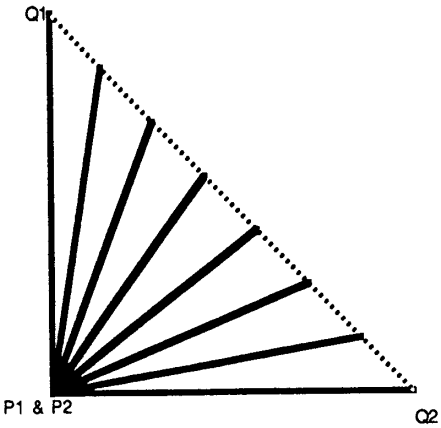


Figure 7: Using linear interpolation for rotation doesn't work

4.1 Key Frame Interpolation

In hand animation, an animator will draw a character in some position, make a second drawing of the character in a new position and then manufacture the inbetween frames. The digital equivalent to this is to interpolate between key positions of models. A simple approach would be to use linear interpolation between key frames, but in real life objects don't start and stop suddenly. There is usually some kind of smooth acceleration depending on the situation. Neither should interpolation necessarily take place along a straight line in space. Particularly with camera moves, a series of straight line segments would make the motion appear discontinuous as the camera suddenly changes direction. A smooth curve would be far more natural.

Figure 7 shows another problem introduced by linear interpolation in space. To move the rod from $P1Q1$ to $P2Q2$, implies a rotation. Linear interpolation would make the rod change length as it rotates. Smooth motion curves can be applied to any part of the animation in both space and time. In space, as shown in figure 7, and in time also to make the rod ease in and ease out of the motion (cubic curve) as it moves. Models, lights, cameras, fades, colours, and so forth can all follow some interpolated path, according to a curve. The simplest method is to use a function such as a cubic polynomial, from which is sampled the position at any instant in time. Since the function is continuous, time between key frames can be scaled arbitrarily by the animator before the final sampling is taken. Some examples are given in figure 6.

4.2 Parametric Interpolation

Parametric systems use keyframe interpolation at a slightly higher level than simple key frame systems. Each object is defined by a set of parameters, and the values of these parameters are interpolated between the keys. If a model is defined by surface patches, for example, the control points can be interpolated and the patches reshaped from the new data. Examples of this type of system are BBOP or EM, both developed at the NYIT Graphics Lab and described in [Sturman 84] and [Hanrahan *et al* 85]. More elaborate control of the motion is discussed in [Steketee *et al* 85], in a system which guarantees continuous acceleration while still having local control of keypoints, and the ability to join motions. Doris Kochanek points out that splines can avoid the discontinuities in the direction and speed of motion produced by linear interpolation. She introduces the use of three control parameters, bias, tension, and continuity,

to provide the animator with more control over the motion [Kochanek 84].

As well as smooth motion through time, it is important that camera moves follow smooth motion paths through space. Using Kochanek's spline technique, the animator can place a camera at a number of key frames, then interpolate smoothly between them through space and time. Similar control can be applied to any other object. Sometimes a discontinuity in time is required, as when bouncing a ball. One way of representing this motion is to apply a damped sine wave: at any time the position of the ball can be found from the curve shown in figure 6. Sometimes the animator will want more control than provided by a damped sine wave. Kochanek's curves allow the animator to smoothly change the "continuity" parameter to produce the desired results.

Sometimes the animator will want to change one model into another as opposed to the normal inbetween problem of interpolating between different positions of the same model. The main problem with such a *metamorphosis* is that the source and destination models do not precisely match. If the objects are polygonal they may contain different numbers of polygons or the polygons may not have the same numbers of vertices. These problems are not found when using the same model in different positions. A more detailed examination of metamorphosis is given in part 2.

4.3 Scripting Systems

Keyframe animation has the advantage that an animator can interact with the motion and specify movements by direct manipulation. However, much animation is algorithmic in nature, and difficult to specify interactively. For this reason, many systems allow the user to specify the animation in a script. There are many examples, [Chuang *et al* 83], [Thalmann *et al* 85], [Reynolds 82], [Zeltzer 82]. The Ani system, [McPheeter *et al* 84] is typical. Ani (part of the Graphicsland system) reads a script and produces a description of every frame in PG, a modelling language. Ani is a multiple track system. Instead of describing each keyframe each object pursues its own course of action, the animator creates tracks of animation and places objects on their own track. The object is moved by interpolation as in a parametric system, however the difference is that the tracks are independent, and a total description of each frame does not exist explicitly. A track is effectively a list of known events in time. A good example of such a system is *Twixt* [Gomez 85]. Scripts in this system are kept as a log of the animators input which is purely interactive. An example of an animation script is given in section 5 .

A somewhat more powerful but less high-level approach than a scripting system is to extend an existing general purpose programming language to provide an environment where it is easy to specify algorithmic animation while also offering the full power of a programming language. The MIRA animation system ([Thalmann *et al* 85]) is a good example of the language Pascal extended to cope with animation.

Most animation systems store the models in some sort of hierarchy. A model is manufactured from different parts, each containing sub-parts, and so on. Each node in the hierarchy contains a geometric transformation matrix which defines how the part is translated, rotated and scaled relative to its parent. For example, a train puffs smoke which moves along with the train; after 1/2 a second the train continues leaving the smoke behind to disperse in the wind. The whole animation is then repeated every 2 seconds. A script system allows such algorithmic animation to be defined without the animator having to learn a full programming language. The problem is algorithmic: in each cycle, the smoke, which has it's own local motion associated with it, must be attached to the engine. The easiest way is to move the smoke relative to the chimney until the smoke must be left behind. While the smoke is "owned" by

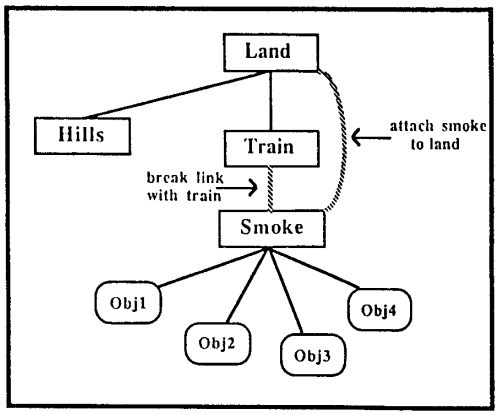


Figure 8: Animation Hierarchy (see example under script systems)

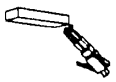
the chimney (which in turn is "owned" by the train) everything that happens to the train (i.e. geometric transformations) also happens to the chimney and in turn to the puff of smoke. As soon as the chimney ceases to own the puff, the puff no longer moves with the train. However, the global position of the smoke in the world is required to remain the same. To achieve this, the puff must be attached to some object, in this case the land after 1/2 a second figure 8 shows the structure. One of the problems with altering the topology of an object in motion is what happens to the orphaned part, in this case the smoke puff. It has velocity and possibly some acceleration, since it is travelling with the train. In the example, the puff will come to an abrupt halt unless the data structure carries sufficient information to allow the object to be eased out of its motion. Many animation systems have separate data structures for the model and for the animation. By using the modelling hierarchy to also store the motion control information, velocity and acceleration become information which is readily accessable when the topology of the system is altered dynamically.

4.4 Rotoscoping

When human animation was first attempted, movement data was often retrieved via *motion recording*. Rotoscoping is one such method, where joint coordinates are hand digitized by viewing at least two orthogonal views of a previously recorded scene (on film or video) [Ridsdale *et al* 86]. Though such a technique is tedious, it began the trend toward analysing human motion for computer animation. Later techniques included the use of goniometers and (expensive) video scanning equipment to record 3-D motion. The results allowed some of the first reasonable animation sequences of human figures, but the techniques used were ultimately too specific to adequately address the human animation problem.

4.5 Inverse Kinematics

Direct kinematics seems to be used mostly in low level animation systems where movements must be mostly described by the animator. To simplify the methods for specifying motion, higher levels of description are needed. One such method is inverse kinematics. Rather than calculate the position of a distal segment based on given joint rotations (as in direct kinematics), inverse kinematics calculates the joint rotations based on the position of the distal segment



[Wilhelms 87]. This allows for more natural movement specifications such as "move hand to cup".

Wilhelms [Wilhelms 87] notes that inverse kinematics provides a method of constraining bodies within their world. A walking figure, for example, must have its feet constrained to the ground (the tendency is for a swing motion where the foot drops below the ground level). Reach goals, often used in robotics, are also solved with inverse kinematics by specifying the location a robotic arm should reach for, and letting the joint calculations be automatically generated. This leads us to a problem found in human animation, but not in robotics: how is a movement made to appear human?

The manner in which a robot arm reaches its goal is not important (ignoring collisions for now), however, when animating human forms the motions must appear real. In what order and with what speed and extent do joints rotate to allow the distal segment to reach its goal? What constitutes a natural human motion is not clear. The redundancy of the human figure contributes to this problem. Even if one movement was determined, this solution would not generalise to other movements, and certainly not to other limbs. To date, no adequate solution for this problem has been offered.

4.6 Motion Dynamics for Animation

Dynamics analysis considers mass, force, and the effects of both combined. It is a simulation of the world, and therefore offers a more automatic approach than kinematics. Dynamics analysis also offers a more "complete" analysis of a scence as it considers effects of bodies on each other. It is, however, computationally expensive and has therefore not been implemented in many systems until recent years.

The simulation used in dynamics is based on Newton's second law [Wilhelms 87]:

```
f = m*a
```

Dynamics equations are developed for each degree of freedom in a scene. A rigid, articulated figure with 18 degrees of freedom which is free to move or rotate in its environment has a total of 24 dynamics equations associated with it. Various types of equations can be used, depending on the application. Lagrangian, Gibbs-Appell and recursive dynamics equations are quite common.

To define the forces and torques acting on a body, several methods can be employed. One method is to provide automatic calculation of forces and torques which have known equations associated with them (such as gravity). Another method is to model springs and dampers for certain joints and segments. The final method relies on user input. Clearly, this input cannot be too complex if the system is to be useful.

Wilhelms [Wilhelms 87] notes three problems associated with dynamics analysis:

- the cost of the analysis is high.
- a numerical instability exists when bodies are complex.
- motion control is restricted to high levels only.

Wilhelms has implemented a system called *Deva* which uses dynamics analysis to animate rigid, articulated bodies [Wilhelms 85]. Armstrong and Green have also been using dynamics analysis for motion control, and have published several interesting results [Armstrong *et al* 85].

5 The Chmilar/Herr experiment

To illustrate the difference between dynamics and kinematics I ran a somewhat ad hoc experiment where two animator/programmers were asked to make a short piece of animation showing a human figure falling from a diving board. The figure must make 3/4 of a somersault before reaching a mattress on the ground. The figure should then bounce from the mattress. Both animators assumed the figure is limp.

5.1 The Kinematic Solution

To produce the kinematic solution requires that the animator has a keen eye for detail and is able to capture the essential parts of the motion to convey what is happening. Using the *Graphicsland System* the motion is prototyped, alterations are made and a new cycle begun, everything is done by trial and error. By observing a physical model it was seen that the man would do half a twist and obviously accelerate under gravity. As the figure begins to fall he will rotate about the heel position until he leaves the platform. The man will then spin about his centre of mass. The major motions were blocked out and observed with a rigid stick figure. By trial and error various motions of limbs were added. What followed was a critical adjustment of the various parameters (joint angles etc.) which define the motion. It became apparent that the hips should "lead" the motion with other body parts following. It was also noticed that a pair of good flailing arms conveyed the idea that the figure was falling, more convincingly than getting the acceleration of the figure precise. The final animation script in Ani is as follows:

```
# detailed body limb motions

spin upper_body x -15 1..6 smooth
spin upper_body x  30 9..15 smooth
spin upper_body x -15 20..21 steady
#
spin upper_body x -10 21..24 smooth
spin upper_body x  30 24..29 smooth
spin upper_body x -20 32..33 steady
#
spin upper_body x  -5 33..35 smooth
spin upper_body x  17 35..38 smooth
spin upper_body x -12 38..39 steady

spin head x -14 1..9 smooth
spin head x  18 11..17 smooth
spin head x  -4 20..21 steady

spin r_arm x 25 1..8 smooth
spin r_arm x -40 11..20 smooth
spin r_arm z -70 9..18 smooth
spin r_arm x 15 20..21 steady
#
spin r_arm z 40 21..24 smooth 22
spin r_arm x 40 21..23 smooth
spin r_arm x -80 23..32 smooth
```

```
spin r_arm x 40 32..33 steady
#
spin r_arm z 10 33..36 smooth 34
spin r_arm x 20 33..34 smooth
spin r_arm x -40 34..38 smooth
spin r_arm x 20 38..39 steady

spin l_arm x 22 1..7 smooth
spin l_arm x -34 11..20 smooth
spin l_arm z 50 14..20 smooth
spin l_arm x 12 20..21 steady
#
spin l_arm z -30 21..24 smooth 22
spin l_arm x 38 21..23 smooth
spin l_arm x -76 23..32 smooth
spin l_arm x 38 32..33 steady
#
spin l_arm z -10 33..36 smooth 34
spin l_arm x 18 33..34 smooth
spin l_arm x -36 34..38 smooth
spin l_arm x 18 38..39 steady

spin r_lower_arm x 30 10..18 smooth
spin r_lower_arm x -30 20..21 steady

spin l_lower_arm x 30 9..18 smooth
spin l_lower_arm x -30 20..21 steady

spin r_leg x 3 1..5 smooth
spin r_leg x 7 11..16 smooth
spin r_leg x -10 20..21 steady
spin r_leg z -20 12..16 smooth
spin r_leg z  23 16..20 smooth

spin l_leg x 3 1..5 smooth
spin l_leg x -10 11..15 smooth
spin l_leg x 7 19..20 steady
spin l_leg z -25 12..15 smooth
spin l_leg z 20 16..20 smooth
#
spin l_leg x -15 20..29 smooth 23
spin l_leg x 15  29..33 smooth 32
#
spin l_leg x -6  32..37 smooth 34
spin l_leg x  6  37..39 smooth

spin l_lower_leg x 3 1..6 smooth
spin l_lower_leg x -3 20..21 steady
```

```
#
spin l_lower_leg x 7 21..26 smooth
spin l_lower_leg x -7 32..33 smooth

spin r_lower_leg x 2 1..5 smooth
spin r_lower_leg x -2 20..21 steady
#
spin r_lower_leg x 6 21..25 smooth
spin r_lower_leg x -6 32..33 steady

spin lower_body x -20 14..20 smooth
spin lower_body x  20 20..21 steady
#
spin lower_body x  10 21..24 smooth
spin lower_body x -30 24..29 smooth
spin lower_body x  20 32..33 steady
#
spin lower_body x   8 33..35 smooth
spin lower_body x -22 35..38 smooth
spin lower_body x  14 38..39 steady

# major body motions

# spin on heels
spin    body y -180  1
spin    body y -180  8..20 slowin 16
move    body 0 3.3 0  1     #translate body up to rotate around heels
move    body 0 0 0  10      #translate to rotate around hips
rotate  body x 270  1..20 slowin 14
move    body 0 24 -6  1

# fall after 90 degrees reached
move    body 0 20.7 -2.7  10
move    body 0 0 -2.7  10..20 steady  #NOTE: not slowin!

# bounce one
move    body 0 4 -2.7  20..26 slowout
move    body 0 0 -2.7  27..32 slowin

# bounce two
move    body 0 2 -2.7  33..35 slowout
move    body 0 0 -2.7  36..38 slowin
```

5.2 The Dynamic Solution

To solve the problem of the falling man by the dynamics approach requires a different set of skills compared with the kinematic. The essence of dynamics is that motions are described by differential equations. Two methods were considered:

- consider forces and torques involved and use $F = ma$ and $\tau = I\alpha$
- consider energy and use Lagrange's equation

It turns out that the first approach requires a great deal of algebra to arrive at the required set of differential equations compared with using Lagrange's equation. Some of the reasons why Lagrange's equations are useful:

- physical intuition helps if you want to get the forces right
- dealing with energy (scalar) is somewhat easier than using vector quantities
- Lagrange's equation allow the easy use of constraint equations
- allows us to introduce new coordinates that can make the problem easier

The major steps in deriving differential equations with Lagrange's equation are:

1. Pick suitable coordinates.
2. Write constraint relations between coordinates.
3. Write the kinetic and potential energy expressions.
4. Use Lagrange's equation to get the differential equations.
5. Differentiate the constraint relations to get enough equations so that all the accelerations can be solved for (accelerations are the crucial quantities to solve for in these problems).
6. Write expressions for the generalised forces; the limits on how far each limb can rotate, damping, the mattress force.

The first major difference between the dynamic and kinematic approach is that the problem is too hard to solve dynamically without making some simplifying assumptions (at least in the one week alloted for this experiment).

1. The man lies in a vertical plane (a full 3D version is being prepared).
2. The body has only 3 segments - body, arms, legs.
3. Initially leaning over the edge of the diving board.

Two systems of differential equations are used

System 1 Constraints included to bind feet to diving board.
System 2 No constraints to bind feet to diving board.

Initially system 1 is used, when the body is below the board, system 2 is used. The final conditions for system 1 are the initial conditions for system 2.

Figure 9 shows the body model used for the animation. Since left and right sides of the body do not work independently, only five coordinates are needed to describe the position of the body:

x, y position of centre of mass of the body (mass = m)
θ angle body makes with horizontal
θ_1 angle legs make with horizontal
θ_2 angle arms make with horizontal

Other coordinates are introduced for convenience:

x_1, y_1 position of centre of mass of the legs (mass = m_1)
x_2, y_2 position of centre of mass of the arms (mass = m_2)

Moments of inertia for arms, legs and body:

$$I = \frac{1}{12} m l_c^2 \quad I_1 = \frac{1}{12} m_1 {l_{c_1}}^2 \quad I_2 = \frac{1}{12} m_2 {l_{c_2}}^2$$

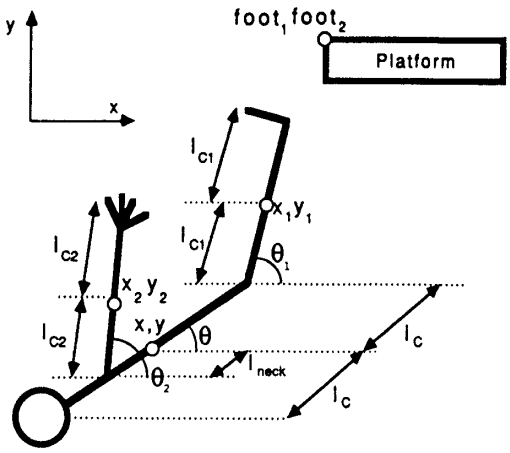


Figure 9: Body model for dynamic animation

Constraint Equations

$$x_1 = x + l_c \cos\theta + l_{c_1} \cos\theta_1$$

or

$$\begin{aligned} f_{1x} &= x_1 - x - l_c \cos\theta - l_{c_1} \cos\theta_1 \\ f_{1y} &= y_1 - y - l_c \sin\theta - l_{c_1} \sin\theta_1 \\ f_{2x} &= x_2 - x + l_{\text{neck}} \cos\theta - l_{c_2} \cos\theta_2 \\ f_{2y} &= y_2 - y + l_{\text{neck}} \sin\theta - l_{c_2} \sin\theta_2 \end{aligned}$$

While the foot is constrained to the platform:

$$\begin{aligned} f_{\text{foot}_x} &= \text{foot}_x - x_1 - l_{c_1} \cos\theta_1 \\ f_{\text{foot}_y} &= \text{foot}_y - y_1 - l_{c_1} \sin\theta_1 \end{aligned}$$

Kinetic Energy

$$T = \frac{1}{2}mv^2 + \frac{1}{2}I\omega^2$$

$$\begin{aligned} v &= \text{speed of centre of mass} \\ \omega &= \text{angular velocity of body about centre of mass} \\ v &= \sqrt{x^2 + y^2} \qquad (x, y \text{ are components of velocity vector}) \\ \omega &= \dot{\theta} \end{aligned}$$

For our man:

Kinetic Energy

$$T = \frac{1}{2}\left(m(\dot{x}^2 + \dot{y}^2) + m_1(\dot{x}_1^2 + \dot{y}_1^2) + m_2(\dot{x}_2^2 + \dot{y}_2^2)\right) + \frac{1}{2}(I\dot{\theta} + I_1\dot{\theta}_1 + I_2\dot{\theta}_2)$$

Potential Energy $U = mgy$

where y is the height of centre of mass above some reference level.

For our man:

$$U = g(my + m_1 y_1 + m_2 y_2)$$

Lagrangian

$$\mathbf{L} = T - U$$

Generating the differential equations

$$\frac{\partial \mathbf{L}}{\partial q_i} - \frac{d}{dt}\frac{\partial \mathbf{L}}{\partial \dot{q}_i} + \sum_k \lambda_k \frac{\partial f_k}{\partial q_i} = -Q_i$$

Where

$\mathbf{L} \equiv$ Lagrangian

$q_i \equiv$ i'th coordinate where $q_i = x, y, \theta, x_1, y_1, \theta_1, x_2, y_2, \theta_2$

$f_k \equiv$ k'th constraint equation where $f_k = f_{1x}, f_{1y}, f_{2x}, f_{2y}, f_{\text{foot}_x}, f_{\text{foot}_y}$

$\lambda_k \equiv$ k'th undetermined multiplier.

$Q_i \equiv$ force or torque applied to the i'th coordinate

Solving for λ_k produces the value of the force that maintains the k'th constraint. The Q_i values allow us to account for damping, the bounce on the mattress and joint limiting torques. Using Lagrange's equation will produce a set of differential equations which have to be solved for the various accelerations. Accelerations required are:

$$\ddot{x}, \ddot{y}, \ddot{\theta}, \ddot{x}_1, \ddot{y}_1, \ddot{\theta}_1, \ddot{x}_2, \ddot{y}_2, \ddot{\theta}_2$$

Using the above results in a linear system of 9 equations for the 15 unknowns which are the accelerations and:

$$\lambda_{1x}, \lambda_{1y}, \lambda_{2x}, \lambda_{2y}, \lambda_{\text{foot}_x}, \lambda_{\text{foot}_y}$$

The additional 6 equations are defined by the constraint equations. The accelerations are found by taking 2nd time derivative giving:

$$\dot{f}_{1x}, \dot{f}_{1y}, \dot{f}_{2x}, \dot{f}_{2y}, \dot{f}_{\text{foot}_x}, \dot{f}_{\text{foot}_y}$$

We now have a system of differential equations which basically describe a triple pendulum. (Three sticks joined together).

Restoring torque Each joint has an equilibrium position and negative and positive limits as indicated in figure 10 which shows how the torque τ_{restore} increases away from the equilibrium position.

Damping There are two types of damping forces:

Linear damping force $F_{\text{damping}} = -b\dot{x}$ where b is the damping constant ($b > 0$).

Rotational damping torque $\tau_{\text{damping}} = -b\dot{\theta}$ is the force opposing the motion.

Mattress Force Spring force acting on the centre of mass in a vertical direction only:

$$F_{\text{mattress}} = 0 \quad \text{if} \quad y > y_{\text{mattress}}$$
$$-k(y - y_{\text{mattress}}) \quad \text{if} \quad y <= y_{\text{mattress}}$$

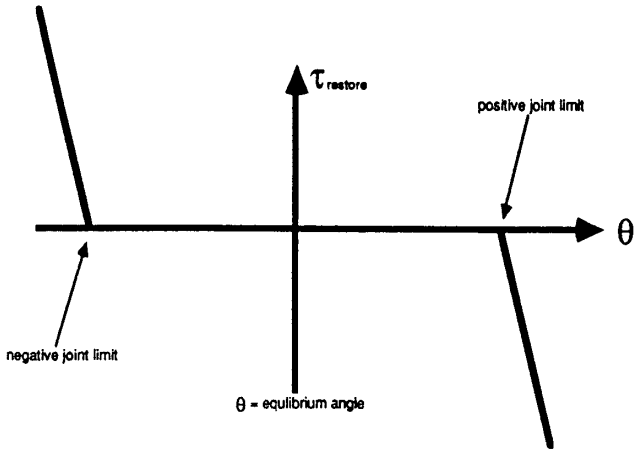


Figure 10: Joint Limits

Q_i**'s for our man** The following expressions summarise the generalised forces for each degree of freedom.

$$
\begin{aligned}
Q_x &= -b_x\dot{x} \\
Q_y &= -b_y\dot{y} + \begin{cases} 0 & \text{if } y > y_{\text{mattress}} \\ -k_y(y - y_{\text{mattress}}) & \text{otherwise} \end{cases} \\
Q_\theta &= -b_\theta\dot{\theta} \\
Q_{x_1} &= -b_{x_1}\dot{x}_1 \\
Q_{y_1} &= -b_{y_1}\dot{y}_1 + \begin{cases} 0 & \text{if } y_1 > y_{\text{mattress}} \\ -k_{y_1}(y_1 - y_{\text{mattress}}) & \text{otherwise} \end{cases} \\
Q_{\theta_1} &= -b_{theta_1}\dot{\theta}_1 + \tau_{\text{restore}\theta_1}(\theta_1 - \theta) \\
Q_{x_2} &= -b_{x_2}\dot{x}_2 \\
Q_{y_2} &= -b_{y_2}\dot{y}_2 + \begin{cases} 0 & \text{if } y_2 > y_{\text{mattress}} \\ -k_{y_2}(y_2 - y_{\text{mattress}}) & \text{otherwise} \end{cases} \\
Q_{\theta_2} &= -b_{theta_2}\dot{\theta}_2 + \tau_{\text{restore}\theta_2}(\theta_2 - \theta)
\end{aligned}
$$

5.3 Making the animation

The symbolic arithmatic to produce the differential equations is fortunately handled by a commercial package (Macsyma). Macsyma code has been written that will generate a program to do the simulation including calls to a commercial differential equation solving package. The values of the positions and angles are output at each display step. This constitutes a kinematic description for the animation program which handles the graphics of the man.

5.4 Conclusions to be drawn from the Chmilar-Herr experiment

The animation frames are shown in 11. To view the two animations please flip the pages of this report. The top right hand corner contains the Chmilar-kinematic animation and the

bottom right is the Herr-dynamic animation. Except in simple cases dynamic solutions take considerable effort. Even if the programming is minimised, with better general purpose software to solve dynamics problems, complex systems require large numbers of equations to be solved and corresponding large amounts of compute time. Our man had only 5 degrees of freedom and produced 9 differential equations. A more complete body model in 3D should have between 30 and 40 degrees of freedom, if further detail is required, such as finger joints, the number is more than 200. In our experiment the dynamic solution modelled the acceleration and subsequent bounce of the man more convincingly than the kinematic. However, the kinematic solution was able to account for more detailed motion and in 3D, such as the half twist on the body as it falls.

One problem with the dynamic approach is that it is difficult to achieve some particular motion. If the objective is to find out what happens when certain forces are applied, such as where a body might be flung in a car accident, then dynamic animation is effective.

Both the kinematics and dynamic solutions to such problems are very complex. What is required is a hybrid of the two approaches, the dynamics to be sufficiently well integrated into the animation system to be a first cut at the final animation. In the final analysis the animator will use whatever tools that best suit his purpose. It is the quality of the software tools that will distinguish the system he chooses.

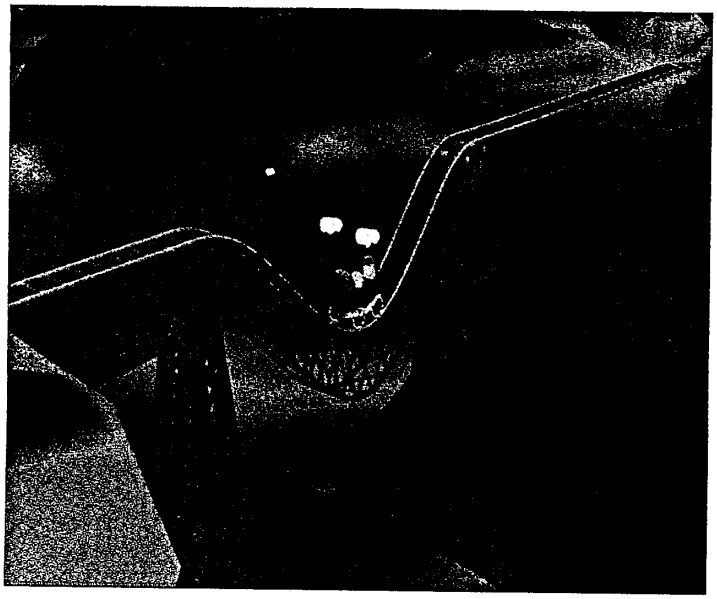


Figure 28: Scene from *The Great Train Rubbery*

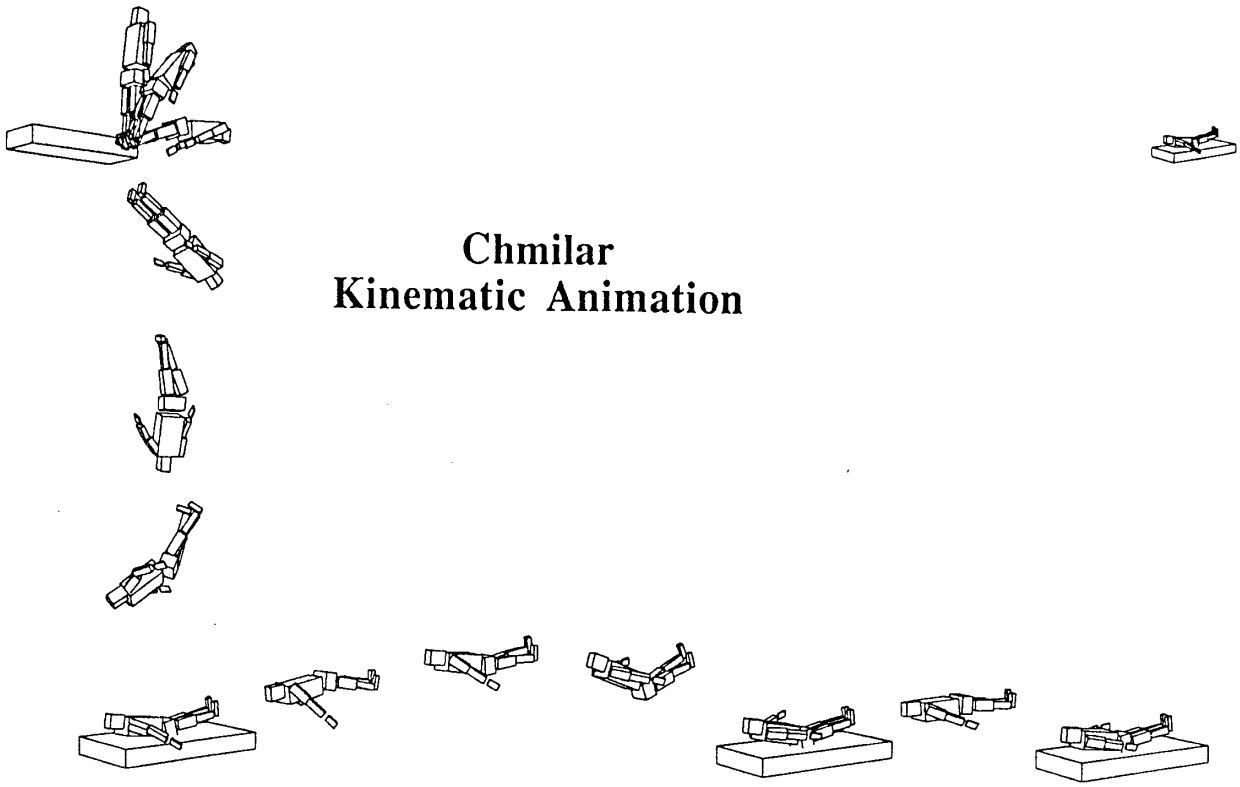


Figure 11: Frames from the Chmilar-Herr experiment

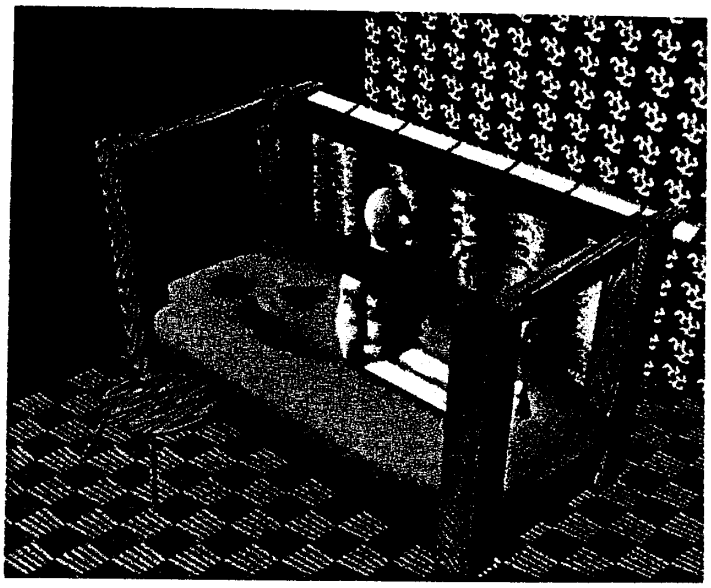


Figure 12: Creating the right impression

Part II
SOFT Objects

6 Introduction

SOFT Objects refers to a particular method of representing 3D models which change shape as they move thus enabling animators to go beyond rigid polygonal models, such as rotating logos, characteristic of computer generated animation. A technique suited to representing such flexible surfaces was developed by Blinn to model constant energy surfaces in molecules (Blobby Molecules, see [Blinn 82]). He uses an iso-surface in a scalar field defined by a number of key points. Perhaps the best way to visualise this idea is by analogy. Each key can be considered to be a hot star in space radiating heat. The iso-surface, or contour, connects all the points which have the same temperature value. Each key can be associated with a *shape function*, the simplest is a sphere so that the key radiates heat equally in all directions. The temperature drops as the distance from the key increases. At some chosen temperature, the iso-value, corresponding to some particular distance away from the key, the spherical surface is found. As two keys approach each other, (see figure 13) the space between the keys heats up changing the shape of the iso-surfaces.

It can be seen from the figure that the surfaces will bulge, then join. Eventually the keys will be coincident and form a new sphere. This method of defining surfaces is extremely useful in computer animation, since the keys can be moved independently to produce objects which change shape over time. To model a more complex surface than a sphere, many keys are used.

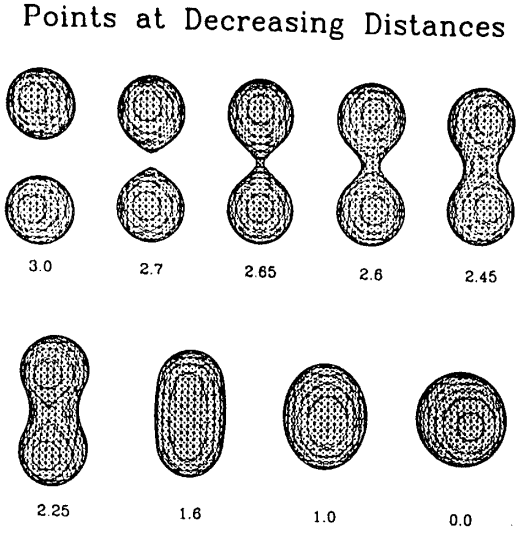


Figure 13: Two SOFT keys approach to form a single sphere

It is relatively simple to check if a particular point in space is inside or outside the surface (hot or cold) by summing the contributions from each key and comparing the field value to the iso-value. The problem is to find out exactly where the surface should be.

6.1 Building a model

As far as the modeller is concerned, the keys form a skeleton of the object. The railway engine in (see figure 14)is a good example. The left hand figure shows the keys which form the skeleton, in this case each key is in the shape of an ellipsoid defined by a skeleton of 3 axes. The right hand figure shows the effect of polygonising the surface. Effectively the keys are covered in a polygon mesh which approximates the surface. Figure 15 is a rendered version of the railway engine, a frame from the film "The Great Train Rubbery". The engine is moving through a region of *abstract texture space*. As it enters the space any point in space is given a colour defined by the 3D texture function. In this case the well known Mandril texture is extruded along the direction of motion of the train producing an interesting visual effect. For further details see [Wyvill *et al* 87].

6.2 Finding and Rendering the Surface

Various different researchers have produced algorithms for rendering these surfaces. At about the same time new algorithms for finding implicit surfaces, along with field functions based upon a cubic function, were developed in Japan (meta-balls) [Nishimura *et al* 85] and in Canada (SOFT Objects) [Wyvill *et al* 86c]. In the latter work the surface is sampled by uniformly subdividing space into cubic regions. Each cube can then be replaced by polygons which approximate the surface. This process has been referred to as clothing, but the more accurate term is polygonising, *polygonisation*[Bloomenth 88b]. In contrast, Blinn renders the surface directly from the function. Although it is possible to directly render implicit surfaces with techniques such as ray tracing, prototype surfaces built from polygons are extremely useful. Many

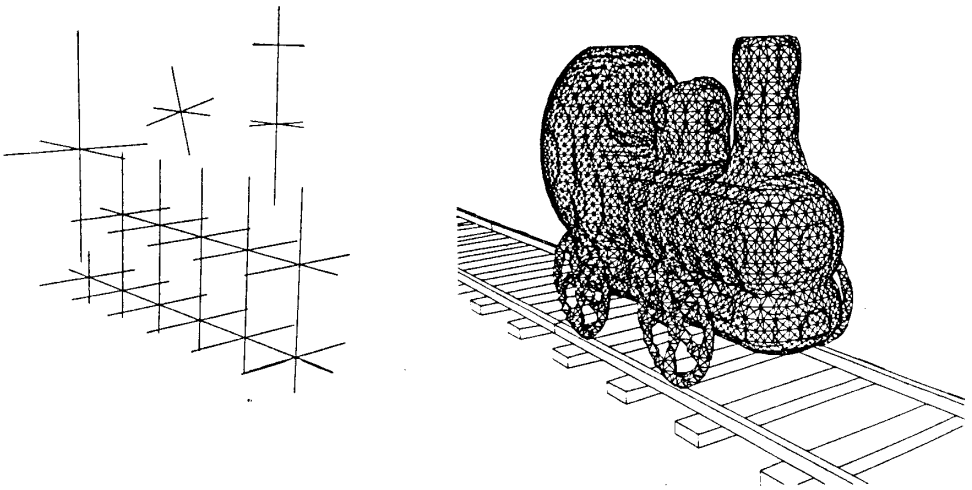


Figure 14: Skeleton of the train and polygon version of the train

Figure 15: Rendered version of the train in a scene from The Great Train Rubbery

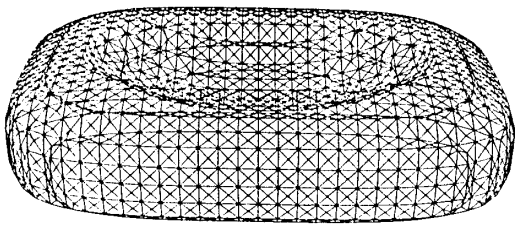


Figure 16: Negative key subtracts the dent from the positive super ellipsoid

workstations, such as the Silicon Graphics Iris, are capable of producing perspective images of large numbers of polygons per second. Keys can be moved interactively and if the surface can be re-calculated quickly enough, interactive editing can take place. Also polygon renderers are common, so there is a large incentive to search for fast polygonisation algorithms. The efficiency of the earlier algorithm has been improved, [Bloomenth 88b][Herzen *et al* 87][Jevans *et al* 88b]. Despite the abundance of names for these implicit surfaces in our research we refer to them as SOFT Objects. Over the last few years, a system for modelling and animating these SOFT Objects has been built at the University of Calgary. The system is called Graphicsland from which many of the examples are taken.

6.3 Positive and Negative Keys

Each key contributes to the field. The contribution can be positive as in figure 13, or negative as illustrated in figure 16. Figure 12 also illustrates the use of negative keys. The crude human figure is made from positive keys. In fact in Graphicsland this becomes a hierarchy of transformations of a single SOFT key, a primitive sphere which can be stretched to form an ellipsoid. It is relatively simple to replace the positive primitive by its negative counterpart, the man now becomes invisible and subtracts from a positive surface, thus leaving a man shaped depression in the mattress.

6.4 Implicit Surfaces

Two methods of producing 3D curved surfaces are commonly used in computer graphics:

- Parametric
- Implicit

Parametric surfaces define a set of points P, such that:

$$P = (x(u,v), y(u,v), z(u,v))$$

Whereas implicit surfaces are simply formulated as:

$$f(P) = 0$$

Iso-surfaces or SOFT Objects, fall under the more general term: implicit surfaces. In computer graphics much attention has been paid to methods for modelling parameteric surfaces

(such as spline patches), but until recently implicit surfaces have been largely disregarded. The exception is some work done by Ricci, ([Ricci 73]), on a constructive geometry which he used to build and modify shapes. Ricci defined the shapes as the boundary between the half spaces $f(P) < 1$ and $f(P) > 1$, in other words those points satisfying $f(P) = 1$. The Constructive Solid Geometry (CSG) that evolved has emphasized the operations performed on primitives, generally limiting these to quadrics. Ricci points out that implicit surfaces offer more succint definitions than parameteric surfaces. An example from a paper by Bloomenthal [Bloomenth 88a] shows the following two definitions of a sphere:

- parametric: $P = C + (rSin(\theta)Cos(\phi), rSin(\theta)Sin(\phi), rCos(\theta)), \theta \in (0, \pi), \phi \in (0, 2\pi)$
- implicit: $f(P) = |P - C| - r$

The implicit formulation offers the designer freedom to arbitrarily constrain a surface, however producing pictures of these generalised surfaces is far from straightforward.

6.5 Field Functions

The shape of the primitives depends on the function used to define the field, and is not restricted by the renderer. To polygonise the surface all that is required is to know if a point is inside or outside (Hot or Cold). To render the surface more precisely, for example with ray tracing, the only additional information required is that the surface normal must be found at the point of intersection with a ray (e.g, [Jevans *et al* 88a]). Thus many mathematical functions lead to some interesting SOFT objects. In the original work done by Blinn, an exponential function was used. In our previous work in this area we used a cubic, [Wyvill *et al* 86c] As the distance r increases, the field value falls from a maximum of $1(r = 0)$ to $0(r = R)$. The value of R is a characteristic of a particular SOFT primitive, the distance beyond which it has no effect on its neighbours. Using the heat analogy; beyond R there is no radiated heat from the key. We chose the cubic coefficients so that the field and its derivative (with respect to r) dropped to zero. This enables fields to be combined efficiently and without approximation, using large numbers of keys. The field at any point depends only on keys in that locality. The above formulation only allows keys with a spherical field to be defined.

6.5.1 Creating Ellipsoid Primitives

In order to find the position of the iso-surface formed by a generalised version of the cubic field function, a few concepts have to be defined. The primitive shape is defined by the shape function (also referred to as the field function) around a key. The key is given by 3 vectors normal to each other, which intersect at the origin of the key. These can be thought of as the axes of the primitive. The shape function defines the surface due to the key. At the origin of the key the field has a fixed value, due to that key, known as the force. In most of our examples, this value is fixed at plus or minus one. The contribution to the field due to the key decreases with distance from the origin, until the radius of influence is reached when the contribution is deemed to be zero.

Figure 17 shows a 2D representation of the iso-surface (in bold) due to a single key and the surface at which the force falls to zero (outer ellipse). Given a point P at distance r from the origin O of the key the contribution of that key at some distance r is determined as follows:

Calculate the distance R where the field value turns to zero along the line OP due to the key by solving the field function.

If $r \leq R$ then the contribution is zero. See figure 17. Otherwise the field is found from r and R. (Decay function).

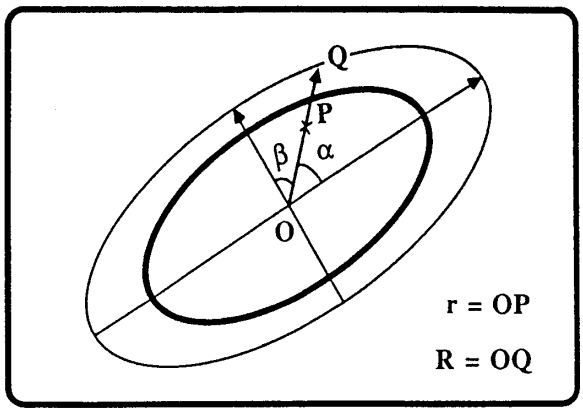


Figure 17: 2D iso-surface

The shape function must provide a continuous closed surface. A useful family of such functions called *super ellipsoids* was made popular by the Danish scientist and poet, Piet Hein, [Gardner 65]. Other functions that could be used are the family of super quadrics [Barr 81]. The super ellipsoids are found from the equation of the ellipsoid:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$$

Where a, b and c are the axes of the ellipsoid. Piet Hein observed that some pleasing shapes can be made by changing the power from 2 to some real power, n. The mattress in figure 12 is a super ellipse with $n = 4$. To make this useful for building SOFT objects the shape function must provide a means of calculating R (see step 1 above).

The following function is formed by replacing x, y, z in (1) with $R\cos(\alpha)$, $R\cos(\beta)$ and $R\cos(\gamma)$ where α, β and γ are the angles made by the axes of the key and the vector OP. (see figure 17)

$$\frac{R^n \cos^n(\alpha)}{a^n} + \frac{R^n \cos^n(\beta)}{b^n} + \frac{R^n \cos^n(\gamma)}{c^n} = 1 \qquad (1)$$

where n is a real value. Figure 18 shows four primitives with values of n at 2 (an ellipsoid) 2.5, 3 and 4. As n is increased the shape becomes more like a cube with rounded corners. This form of the super ellipsoid provides a primitive defined by a key oriented at an arbitrary angle. The axes a, b, c can be used to alter the aspect ratio of the primitive. Since the axes are orthogonal, γ can be found in terms of α and β. Also since OP and the key axes' vectors are known, the cosine can be evaluated with a few multiplications. The value of the field can now be found from the values of r and R. It has been found empirically that substituting r and R into the cubic function given in [Wyvill *et al* 86c] provides good results. Reasonable results can also be obtained with fewer floating point calculations using

$$F = 1 - \frac{r^2}{R^2} \qquad (2)$$

although the surface blending is more rapid as keys approach each other, compared to the cubic. The contribution to the field F, of some key can be scaled by the *force* characteristic of that key. The force can be positive or negative providing the user with further control on the shape of the objects being modelled.

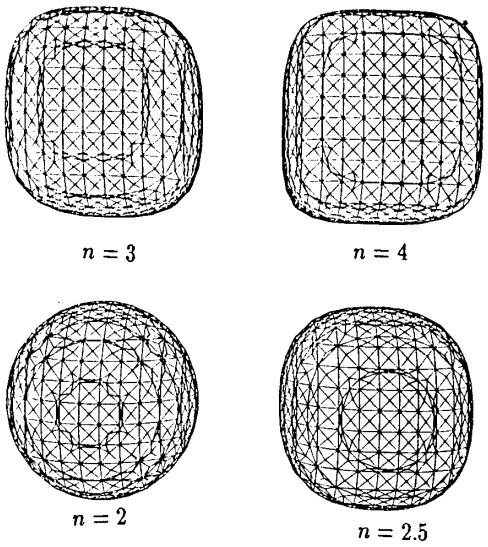


Figure 18: Super Ellipsoids

7 Computing the Surface

The problem is to find the surface of an object given a set of *keys* and their field functions. In [Wyvill *et al* 86c] an algorithm is described which finds the field value at successive points along one axis (starting at a control key), each point is the corner of a cube. In fact it is part of a grid of cubes sub-dividing the object space. Near the key the value will be greater than the iso-value (hot). When the field falls below the iso-value the surface has been crossed (the value becomes cold). It can then be determined which cube edges are cut by the surface, by comparing the field value of each vertex to the iso-value. If the surface cuts a particular edge the adjacent cube in the grid is manufactured and the process is repeated recursively until the entire surface has been covered by containing cubes. The surface can then be approximated by manufacturing a polygon (or set of polygons) in each cube. Details of this algorithm are given in [Wyvill *et al* 86c].

7.1 Improving the search

Finding the surface which *clothes* the keys requires the evaluation of the field function at many different points in space. The field value at any point is the sum of the field values due to each key. However the field function used falls to zero at some distance from the centre of the key. To reduce the number of field evaluations for each point we use space subdivision or cellular *clothing* in which the object space is divided into a 3D grid of cubic volumes or cells. The method employed is similar to that of ray tracers and other graphics rendering processes [Cleary *et al* 83]. Essentially, space is broken down into units that are more manageable. Each cell (unit) contains only those objects that affect it, in this case a subset of the keys. When finding the iso-surface, only keys whose influence extend into the current cell need to be examined for their field contributions. In this way, checking every key point in the 3D space is avoided [Wyvill 86]. Bloomenthal (see [Bloomenth 88b]) found that octree subdivision could be faster than the original uniform subdivison. We have also experimented with this technique [Jevans *et al* 88b].

The octree subdivision has a minimum and a maximum limit of recursion. This effectively

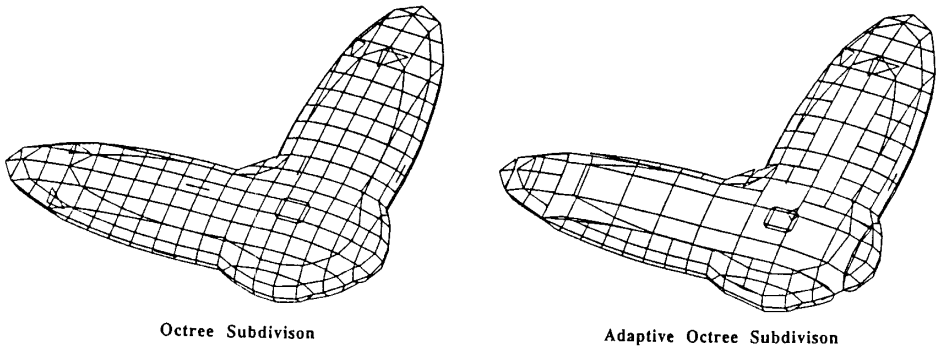


Figure 19: Results of octree and adaptive octree subdivision

controls the resolution of the polygonal surface and how accurately it reflects the functionally defined iso-surface. The minimum limit ensures that the octree subdivision will proceed a minimum number of times, regardless of the state of each node's vertices. This ensures that surfaces that are smaller than a cell will be found. The maximum limit stops the octree subdivision and effectively controls the resolution of the surface.

If all the cells are sub-divided equally then as many polygons are produced over relatively flat areas as are used to represent extreme curvature. Since the polygons are subsequently passed to a renderer when shading is requred it is desirable to reduce the number of polygons. This can be done by having more octree subdivisions in areas of greater surface curvature. Von Herzen developed such a technique using *restricted quadtrees* (see [Herzen *et al* 87]). Bloomenthal and independently Jevans have developed a method similar to this for use with SOFT objects. [Jevans *et al* 88b][Bloomenth 88b] One problem is that when two adjacent octree nodes have different levels of subdivision it is difficult to join the polygons in neighbouring nodes, This is shown in figure 19. (Bloomenthal now has an elegant algorithm that solves this problem). Although there are considerable savings to be had in the rendering process through the use of irregularly subdivided surfaces, introducing this technique reduces the performance of polygonising considerably.

One other advantage to the cellular subdivison is that frame coherance can be exploited. The idea is that a cell will be flagged if any change has ocurred in that cell since the last frame. Since each cell is independent of the other cells i.e. all the information necessary to describe that region of space is contained in the cell, then only the field values at the vertices of the flagged cells need be re-evaluated.

8 Polygonising

Choosing a polygon mesh which approximates the surface within a cell is responsible for 75% of the time taken in clothing the surface. During the development of the new clothing algorithm several new methods for building polygonal meshes from surface bounding boxes were explored. One very simple approach is to simply use the cubes representing each node as the polygons of the surface. This method is extremely fast but results in very crude approximations to the surfaces. The effect is like constructing the polygon mesh of many small boxes on it's surface. This method can produce reasonable surfaces but only if the subdivision's upper limit is set

unreasonably high.

To provide more accurate surfaces at lower subdivision levels an adaptation of the algorithm given in [Wyvill *et al* 86c] has been found to give a fast and reasonably accurate response. Each cube (octree leaf node) has vertices that are either hot (field value > iso-value) or cold (field value < iso-value). Vertices on the surface are taken as hot. There are 8 vertices at each node and thus 256 combinations of hot and cold. A table is constructed (once at initialisation time) which indicates the appropriate polygons to construct for each case. The cube vertices are numbered systematicaly, so that the bits set in a single 8 bit byte represent the hot vertices. The byte is stored for each cube and is used to manufacture a pointer into the correct part of the table. The polygons themselves are algorithmically constructed as in [Wyvill *et al* 86c] and are then stored in a lookup table. Our empirical results so far indicate that the use of the table increases the speed of this operation by more than one order of magnitude over the previous algorithm.

9 Animating SOFT Objects

Traditional animators often criticise 3D computer generated animation for the stilted way objects move. Computer generated objects or *characters* tend to lack the subtleties in motion seen in traditional animation. Characters do not have to be humanoid, objects can be given character such as the brooms in the Sorcerer's apprentice sequence from Disney's Fantasia. In other words they are anthropomorphic. The motion of such objects is controlled to a fine degree to characterise their movement. Characters tend to bend as they move. At times, they must conform to their surroundings: A figure sitting in a chair is an example, or flowing water. In computer animation, popular modelling techniques such as polygon meshes or spline patches do not lend themselves to the manufacture of objects which can be given this type of motion. SOFT Objects lend themselves to representing models that undergo some kind of shape change. The intention is to let the animator design the skeleton of the character or object and then automatically *clothe* this skeleton with a surface. If the skeleton moves then the surface changes its shape smoothly to conform. If the skeleton undergoes metamorphosis to a totally different skeleton or inbetweens to a skeleton in a new position then a new surface is calculated at every frame. The surface is represented in such a way that it maintains nearly constant volume as the skeleton moves, thus providing convincing *character coherence.* This property of coherence is very important. Intuitively, it means that throughout metamorphosis, the model remains recognisable as the same character. A model built from keys can be moved without shape change by maintaining the relative spatial relationship between the keys, in the absence of any influencing field. Achieving shape change without distorting the model beyond recognition requires that certain constraints be placed on the animation system. The following techniques can be used to animate a model built from a skeleton of SOFT keys to achieve shape change:

- Geometric Transformation: motion of keys relative to each other.
- Altering the Field: influence of global or local field.
- Changing key characteristic: Altering parameter describing a key.

The following sections describe how these methods can be used by high level motion specifications to create the appropriate blended surfaces. It should be noted that this method does not manufacture accurate human models. However it does provide a very fast way of producing blended surfaces that can approximate to a humanoid character. The methods described here

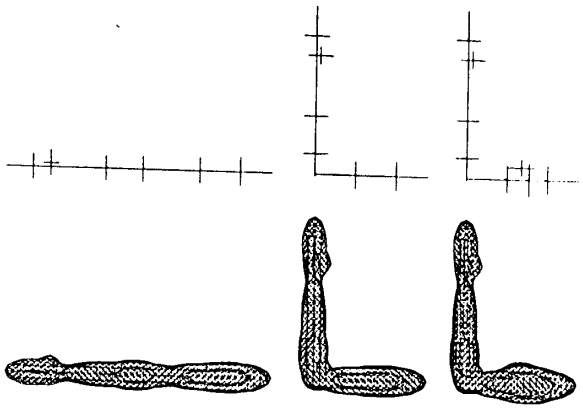


Figure 20: Bending an arm

are low level, they affect individual keys or groups of keys. It is intended that they can be assimilated into a higher level animation system as described in part 1.

9.1 Geometric Motion

By simply altering the relative spatial relationship between keys a surface can be made to alter its shape. Figure 20 shows the skeleton of arm with the position of the keys marked. By moving the keys with the skeleton the surface blends smoothly and the arm appears to bend. There are two ways the skeleton can be replaced by the keys: once, by the animator, specifying for each skeleton part a key configuration or procedurally by the system. The latter method could be applied once, to the model, or at every frame. If each limb is defined as a line, then soft keys can be evenly spaced along the length of the limb by simple linear interpolation. However, if the length of the limbs remain constant (as is normally the case) then it is quite adequate and usually more effective for the animator to interactively specify the keys at the start of the animation. The relationship between keys and skeleton remains constant throughout.

9.2 Path Deformation

Motion paths are extremely useful when applied to the keys of a SOFT object. A cartoon-like character can be made to conform to its surroundings by using the shape of an object to define the path which a group of keys must follow. This effect was demonstrated in the movie SOFT (see [Wyvill 86]). The character in this case was a group of letters spelling the word "SOFT". Each letter was made from about 20 keys, the path was drawn up a flight of stairs by marking various positions and passing a spline through these points using a spline technique similar to that described in [Kochanek 84]. Each key in the character is moved to the interpolated path position at each frame. Each point along the path represents a position at a particular instance in time. Rather than define a separate path for each key, the keys are grouped so that each group is moved together to the next point in the path. Normally a group is chosen according to some spatial relationship, for example all keys within a specific range of z values. To maintain "character coherence" the relative positions of the keys within a group must be maintained within certain constraints. In this case the amount each group is allowed to vary its position is constrained by the change along the path. Each key maintains its

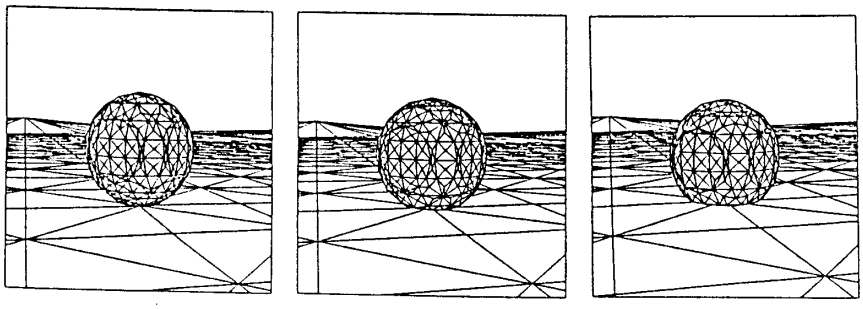


Figure 21: Global field deformation

position relative to the group. It is the origin of the group that follows the path. The positions of the groups change relative to the other groups by an amount specified in the path. Thus the model undergoes the desired deformation but maintains the overall shape and retains the character coherence property which distinguishes shape distortion from metamorphosis. When the letters are moved up the stairs in the movie SOFT the direction of motion was along the (-)ve z axis so the groups were chosen as keys with same z value. As each group advances along the path the y value of the group was altered according to the path specification (see [Wyvill *et al* 86b] for frames from the film).

9.3 Altering the Field

SOFT objects may also be animated by altering the field in which the objects exist. Local deformation can be achieved by moving other SOFT keys relative to these objects, or by placing some global influence into the field. A good example of local deformation is shown in figure 12. As the negative human figure lies on the mattress, his impression is left there. The mattress is deformed by the motion of some external influence. Global deformation is more difficult to specify in a completely general way. The field itself could have some external influence such as a plane of constant value. Objects approaching that plane will deform according to the value chosen. In figure 21 a ball is shown approaching a plane held at the iso value. The plane is of course invisible but shown here as a grid of polygons in fact placed slightly higher than the plane itself. The ball deforms as it approaches the plane and returns to its original shape as it moves away.

9.4 Changing key charactersistic

Each key is defined by three axes and a "force". The size and orientation of the axes may be changed and also the force, which is a scale factor applied to the key's contribution to the field. The effect of negative force has already been shown. By increasing the force the effect on the neighbouring keys will be altered, the effect is very similar to changing the size of the axes of the key. Only making the force (-)ve produces animation effects that cannot be achieved by other means. Since SOFT objects remain blended if they are in close proximity, scaling an individual key can appear somewhat like a muscle bulging as an arm is bent. Figure 20 shows the same arm bending only this time the keys representing the muscle area are made larger. To achieve high level control the system would have to know that each time the arm is bent a proportional scaling must be applied to certain keys. This can be achieved in a script system

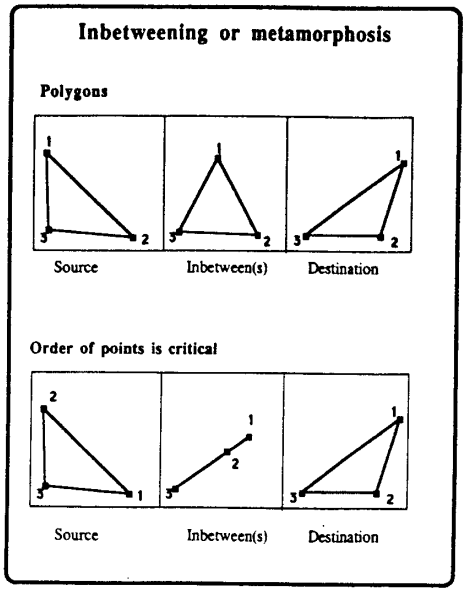


Figure 22: Inbetweening Polygons

by defining a method that is invoked each time the a level command is issued to bend the arm. This kind of procedural animation is difficult to achieve in purely interactive systems.

9.5 Inbetweening and Metamorphosis

One area that SOFT objects are particularly useful for is metamorphosis, or inbetweening. A method suitable for two-dimensional (cartoon) models, would have the character drawn on a vector oriented display in two positions, and the inbetween frames interpolated by the system. The simplest way to do this is to interpolate each point of the first (*source*) object to a corresponding point on the second (*destination*) object. Difficulties arise if the number of points is different on source and destination objects. New points have to be created or several points have to collapse onto a single point. Even if the number of points is the same on the two objects, if they are distributed in a different way the image will be scrambled as each point is interpolated. SOFT objects always guarantee a closed surface, so this problem does not arise. However, it is still easy to lose character coherence in the inbetween versions. Figures 22 and 23 illustrate these points.

Inbetweening is not only used to show motion of a character from one position to another, it can also be used to show metamorphosis from one character to another. If the characters are very different in shape and number of keys, then the scrambling problem is difficult to avoid. Peter Foldes uses software by Burtnyk and Wein [Burtnyk *et al* 76] in the film, "La Faim" and exploits this technique to good advantage. However to avoid *scrambling* is a difficult and tedious task which requires very careful design of the keypoints for inbetweening.

Burtnyk and Wein developed a computer version of this technique using skeletons. These skeletons defined a conformal mapping from one key frame to the next. The space itself was distorted, thus any line within the space was similarly distorted according to the mapping function. In contrast, 3D computer generated characters are often moved by applying geometric transforms to the different parts, which changes their relative positions but do not necessar-

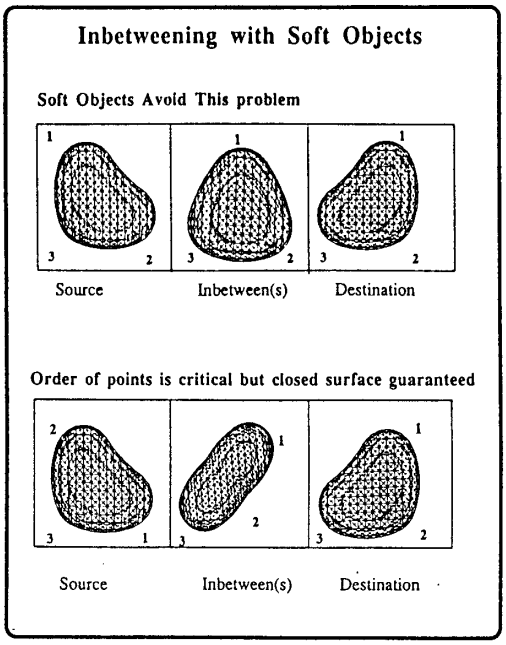


Figure 23: Inbetweening SOFT keys

ily give the character a smooth change of shape as can be achieved using the 2D techniques. However the advantages of using 3D characters are considerable as the computer can be used to render frames from any chosen camera position with the appropriate lighting all calculated automatically. An effective way of producing shape change in 3D animation is to use the inbetween technique. However extending the 2D technique to 3D introduces new problems. Reeves points out in [Reeves 81] that it is difficult to identify corresponding points (and polygons) on different characters. Even with functional representations, the parameters from which a surface is manufactured must be chosen so that the source model *matches* the destination model. Each of the parameters defining the *source* model must be changed to one of the parameters defining the destination model. The matching process chooses the appropriate *destination* parameters corresponding to the *source parameters*. At each intermediate stage during the inbetween, a model will be manufactured from an interoplated set of parameters.

In the following paragraphs several different heuristics for matching the models are presented. The shape of the intermediate models vary according to the chosen method, based on one or more of the heuristics.

9.6 Heuristics for Point Matching in Metamorphosis

In this section four approaches are described that we have found useful for defining the matching process. Although the SOFT object modelling system has been used to illustrate how these heuristics may be applied, the methods are general and can be extended to other modelling techniques. In practice an animator will want to experiment with different combinations of these techniques to arrive at the desired effect.

We start with two models; the source model and the destination model. The source model must be made to change into the destination model. The models are defined as a set of SOFT object keys as described above. Each key has the following properties:

Axes Vectors **v1,v2,v3**

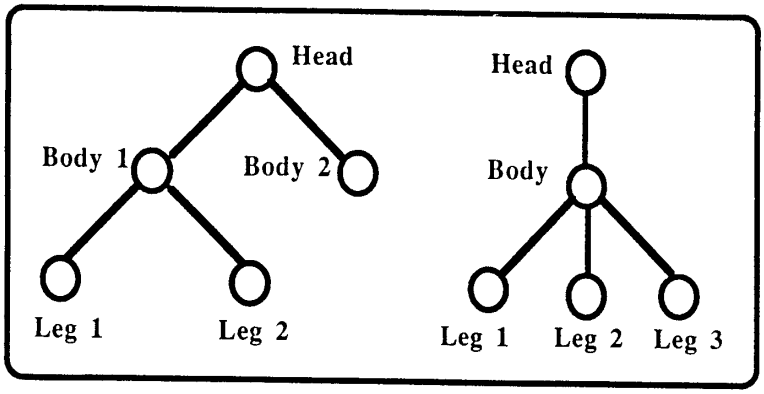


Figure 24: Hierarchical Matching

Postion	x,y,z
Force	F

Each of these methods assumth that ice objects have been pre-processed so that there are the same number of keys defining each object. This may involve creating zero weighted keys. A key can be weighted using the force, F, which scales the contribution to the field, or by scaling the axes. A zero weighted key has its axis vectors set to zero or $F = 0$. When a source key is inbetweened to a destination key, at each frame a new key is chosen which has an interpolated value for position, axes vectors and force. The start or finish position of the new keys is chosen by the appropriate method.

9.7 Hand Matched

The simplest method of establishing which keys are to be interpolated is to to order the source and destination keys by hand and to process each pair in turn. Since the number of keys is small compared to the number of polygons in an equivalent model, this method is feasible for some objects. However computer animation is generally moving towards higher levels of control so this method is considered a last resort.

9.8 Hierarchical Matching

In this heuristic it is assumed that each model is represented by a hierarchy of keys. Each node in the hierarchy has an arbitrary number of sibling nodes and one or zero child nodes. Nodes are matched at the same level in the hierarchy. An example hierarchy is shown in figure 24. The character with two bodies is matched to the character with three legs. It is assumed that the hierarchies are designed that each level of the source object has an equivalent level in the destination object. If a man is to change into a rabbit the heads will be matched, the arms of the man can match the front legs of the rabbit and so on (see figure 27). The main problem with this approach is that the hierarchies have to be constructed carefully. Not only do the levels have to match but within a level the nodes must either be ordered or labelled to match. Despite these drawbacks this method is still preferable to ordering all the keys by hand and for small sets of keys, with suitable interactive tools quite acceptable.

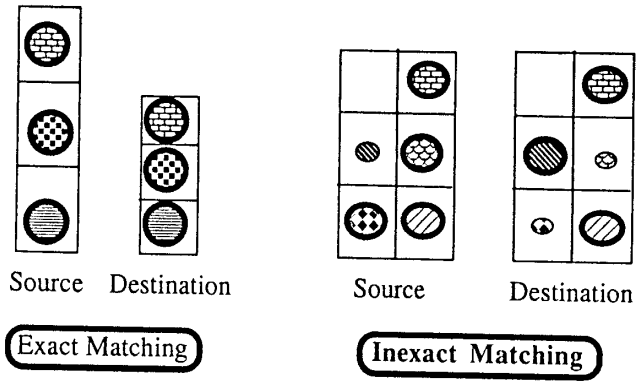


Figure 25: Cellular Matching

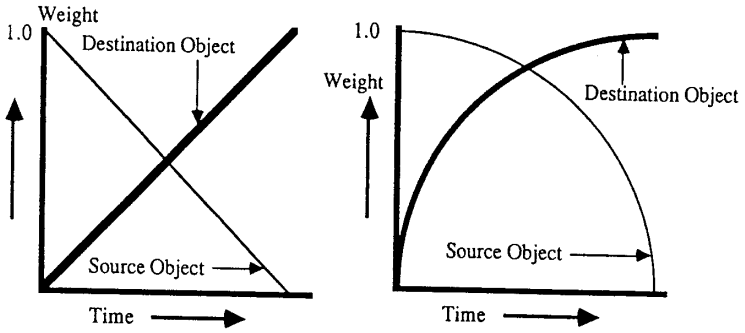


Figure 26: Surface Inbetweening

9.9 Cellular Inbetweening

In this technique the models are matched corresponding to the space they occupy. The world is first divided into a 3D grid of cells. This is done by finding the extents of each model and manufacturing the corresponding rectangular box. Each box is then divided along the x,y,z axes by some user defined amount. The two boxes may be different shapes but they are divided into an equal number of cells. The keys are then sorted into the cellular grid and the keys in each source grid cell are then interpolated to the keys in the corresponding grid cell of the destination model. The objects can have different sizes but the method tries to maintain some sort of position coherence between source and destination objects. Figure 25 shows a 2D version of how the keys are matched. Circles with similar shading patterns are matched between source and destination. In the top diagram the points match exactly, for every key in every cell in the destination object there is a corresponding key in the corresponding cell in the source object. In the lower diagram there are some cells containing keys in the destination object for which there are no keys in the corresponding cell in the source object In this case a zero weighted key (indicated as a small circle) will be manufactured in the source object. Similarly keys which exist in the source object are grown in the destination. An example of the use of cellular inbetweening is shown in figure 27.

Part	R-Keys	M-Keys	matched keys	number of cells	cells used
head	13	2	5	2^3	7
torso	7	16	2	3^3	6
left leg	3	13	6	2^3	6
right leg	3	13	5	2^3	6
left arm	9	1	3	2^3	4
right arm	9	1	3	2^3	4

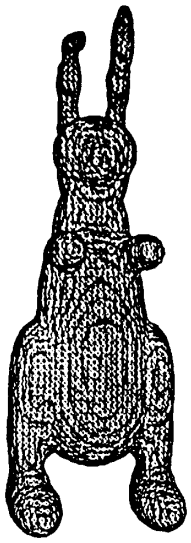
Table 1: Matching the man and the rabbit

9.10 Surface Inbetweening

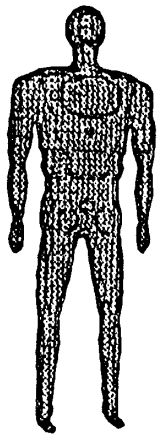
In this method there is no matching necessary. All the keys from both source and destination objects define each intermediate model. However the force property of each source key is weighted. The weighting is gradually changed from one to zero as the inbetween progresses. Also dependent on time is a second weighting applied to the force property of the destination keys. This value changes from zero to one. The shape of the weight value vs. time curve controls the shape of the intermediate model. This is shown in figure 26. A simple linear interpolation means that both source and destination objects are reduced to half the weight half way through the simulation. In practice this gives poor results as the SOFT surfaces around each key no longer merge. If the source is weighted by a cosine function and the destination weighted by a sine function each object is never weighted by less than $\frac{1}{\sqrt{2}}$. The objects can still be matched by one of the sorting techniques (hand, hierarchy, cellular), but the inbetweening process is different using surface inbetweening. An example is shown below.

Metamorphosis of rabbit to man Figure 27 shows some frames from a metamorphosis in which a rabbit is changed to a man. The frames in the top row were produced using the surface inbetweening method, with a sin/cos decay weighting function. The cellular method in combination with the hierarchical technique were used to produce the frames on the bottom row. The man and rabbit were first broken into two equivalent hierarchies and the cellular method used within each pair of parts. Table 1 indicates how well the keys were matched, M indicates man, R Rabbit. It can be seen from figure 27 that the hierarchical/cellular approach succeeds somewhat better than the surface method. Clearly other heuristics are necessary to stop the model from breaking apart. The actual inbetween is done along linear paths in space (though not in time) and it is the relationship between the paths that govern the intermediate shapes that are generated. The good point about this is that little effort is needed to get some sort of metamorphosis and although the figure may break into separate parts, these parts will individually maintain a closed surface.

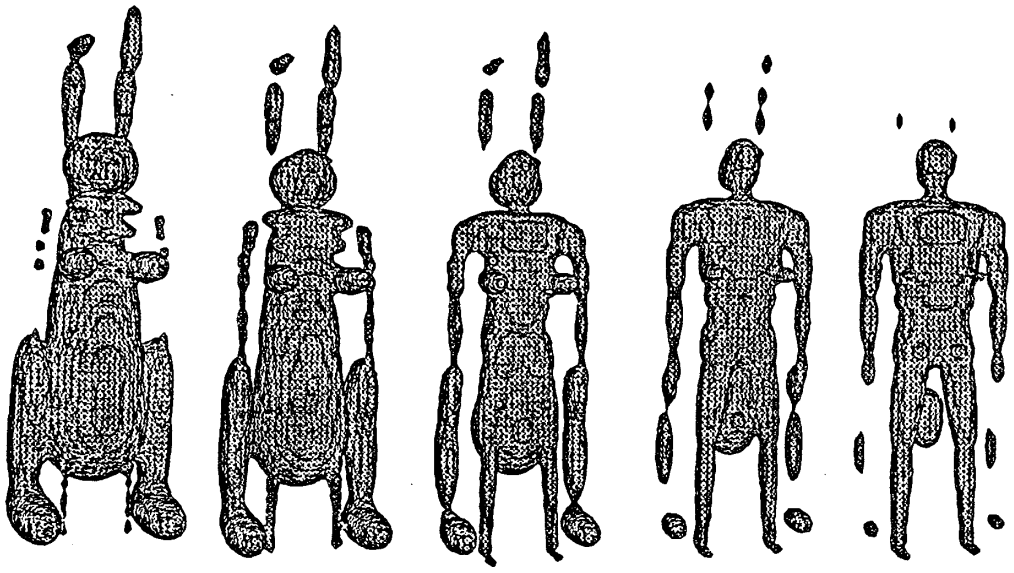
Lessons learned from the heuristics The main lesson to be learned is that good results cannot be obtained from the few heuristics described here. Other constraints have to be placed on the keys before they can be made to form reasonable models in the inbetween stages. Further constraint based heuristics may well prove a fruitful area of research.



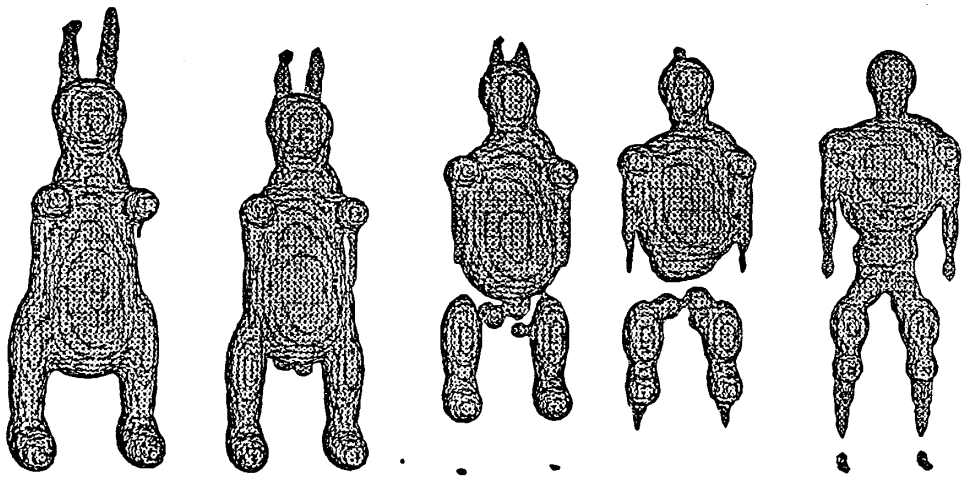
Rabbit (Source)



Man (Destination)



Surface Inbetweening



Cellular Inbetweening

10 Conclusion

Presented here is an attempt to impose some order on the many techniques in 3D computer animation. Particular attention has been paid to a relatively new area, the use of implicit surfaces for animation. Although there are still many outstanding problems to solve, such surfaces have an important role to play in the computer animators ammunition store of techniques. Hopefully the reader will at least know where to look for further information and be slightly less confused than before this journey through the jungle of animation.

11 Acknowledgements

I would like to acknowledge the help of my fellow worker in implicit surfaces, Jules Bloomenthal of Xerox Parc for the use of some of his material. Also my graduate students in the preparation of this paper and the film and video material shown during the presentation. I would particularly like to thank Jeff Allan, Mike Chmilar, Angus Davis, Anja Haman, Charles Herr, Dave Jevans and Trevor Paquette. Mike Chmilar composed the kinematic animation and Chuck Herr slaved over a computer full of hot differential equations to produce the dynamics of the falling man. This research is partially supported by grants from the Natural Sciences and Engineering Research Council of Canada.

References

[Armstrong *et al* 85] William Armstrong and Mark Green. The dynamics of articulated rigid bodies for purposes of animation. *The Visual Computer*, 1:231–240, 1985.

[Barr 81] Alan H. Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 11–23, Jan 1981.

[Barsky 81] B. Barsky. The beta-spline: a local representation based on shape parameters and fundamental geometric measures. *University of Utah, Dept. of Computer Science*, 1981. PhD dissertation.

[Blinn 82] James Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1:235, 1982.

[Bloomenth 88a] Jules Bloomenthal. Designing with implicit surfaces. *Visual Computer (in press)*, 1988.

[Bloomenth 88b] Jules Bloomenthal. Polygonisation of implicit surfaces. *Computer Aided Geometric Design (in press)*, 1988.

[Burtnyk *et al* 76] N. Burtnyk and M. Wein. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *CACM*, 19(10):564, Oct 1976.

[Catmull *et al* 80] Ed Catmull and Alvy Ray Smith. 3D transformations of images in scanline order. *Computer Graphics (Proc. SIGGRAPH 80)*, 279–285, July 1980.

[Chuang *et al* 83] Richard Chuang and Glenn Entis. 3-D shaded computer animation—Step by step. *IEEE Computer Graphics and Applications*, 3(3), December 1983.

[Cleary *et al* 83] John Cleary, Brian Wyvill, Graham Birtwistle, and Reddy Vatti. A parallel ray tracing computer. *Proc. XI Association of Simula Users Conference. Paris*, 77–80, 1983.

[Cleary *et al* 87] John Cleary and Geoff Wyvill. *Analysis of an Algorithm for Fast Ray Tracing Using Uniform Space Subdivision.* Technical Report 87/282/30, University of Calgary, Dept. of Computer Science, October 1987. Currently in press for *Visual Computer.*

[Feibush *et al* 80] Eliot H. Feibush, Marc Levoy, and Robert L. Cook. Synthetic texturing using digital filters. *Computer Graphics (Proc. SIGGRAPH 80)*, 294–301, July 1980.

[Gardner 65] Martin Gardner. Mathematical games. *Scientific American*, 222–236, September 1965.

[Glassner 84] Andrew S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 15–22, October 1984.

[Gomez 85] Julian E. Gomez. Twixt: a 3D animation system. *Computers and Graphics*, 9(3):291–298, 1985.

[Hanrahan *et al* 85] Pat Hanrahan and David Sturman. Interactive animation of parametric models. *The Visual Computer*, 1985.

[Herzen *et al* 87] Brian Von Herzen and Alan H. Barr. Accurate triangulations of deformed intersecting surfaces. *Computer Graphics (Proc. SIGGRAPH 87)*, 21(4):103–110, July 1987.

[Immel *et al* 86] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A radiosity method for non-diffuse environments. *Computer Graphics (Proc. SIGGRAPH 86)*, 20(4):133–142, 1986.

[Jevans *et al* 88a] David Jevans and Brian Wyvill. Ray tracing implicit surfaces. *Research Report No. 88/292/04*, Jan 1988.

[Jevans *et al* 88b] David Jevans, Brian Wyvill, and Geoff Wyvill. Speeding up 3D animation for simulation. *Proc. SCS Conference*, 94–100, 1988. Proc. MAPCON IV (Multi and Array Processors).

[Kay *et al* 86] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. In *Computer Graphics*, pages 269–278, ACM SIGGRAPH, 1986.

[Kochanek 84] D. Kochanek. Interpolating splines with local tension, continuity and bias control. *Computer Graphics (Proc. SIGGRAPH 84)*, 18(3):33–41, 1984.

[Lasseter 87] John Lasseter. Principles of traditional animation applied to 3D computer animation. *Computer Graphics (Proc. SIGGRAPH 87)*, 21(4):35–44, July 1987.

[Mandelbro 77] Beniot B. Mandelbrot. *The Fractal Geometry of Nature.* W. H. Freeman, New York, 1977.

[McPheeter *et al* 84] C. McPheeters and B. Wyvill. *A Tutorial Guide to the ANI Animation System.* Technical Report 84/187/45, University of Calgary, Dept. of Computer Science, 1984.

[Nishimura *et al* 85] H. Nishimura, A. Hirai, T. Kawai, T. Kawata, I .Shirakawa, and K. Omura. Object modelling by distribution function and a method of image generation. *Journal of papers given at the Electronics Communication Conference '85*, J68-D(4), 1985. In Japanese.

[Peachey 85] D. Peachey. Solid texturing of complex surfaces. *Computer Graphics (Proc. SIGGRAPH 85)*, 19(3):279–286, 1985.

[Perlin 85] K. Perlin. An image synthesizer. *Computer Graphics (Proc. SIGGRAPH 85)*, 19(3):287–296, 1985.

[Reeves 81] W. Reeves. Inbetweening for computer animation utilizing moving point constraints. *Computer Graphics (Proc. SIGGRAPH 81)*, 2:263–269, 1981.

[Reeves 83] William. Reeves. Particle systems- a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, 2:91–108, April 1983.

[Reynolds 82] Craig W. Reynolds. Computer animation with scripts and actors. *Computer Graphics*, 16(3), July 1982.

[Ricci 73] Ricci. Constructive geometry for computer graphics. *computer journal*, 16(2):157–160, May 1973.

[Ridsdale *et al* 86] G. Ridsdale, S. Hewitt, and T. W. Calvert. The interactive specification of human animation. *Graphics Interface*, 121–130, May 1986.

[Steketee *et al* 85] Scott N. Steketee and Norman I. Badler. Parametric keyframe interpolation incorporating kinetic adjustment and phrasing control. In *Computer Graphics*, ACM SIGGRAPH, July 1985.

[Sturman 84] David Sturman. Interactive keyframe animation of 3-D articulated models. *Proceedings of Graphics Interface*, 35–40, 1984.

[Sutherlan *et al* 74] Ivan Sutherland, Robert Sproull, and Robert Schumacker. A characteristic of ten hidden-surface algorithms. *Computing Surveys (ACM)*, 6(1), Mar 1974.

[Thalmann 86] D. Thalmann. A lifegame approach to surface modeling and rendering. *Visual Computer*, 2(6), Dec 1986.

[Thalmann *et al* 85] N. Thalmann, D. Thalmann, and M. Fortin. Miranim: an extensible director oriented system for the animation of realistic images. *IEEE Computer Graphics and Applications*, 5(3):61–73, March 1985.

[Wilhelms 85] Jane Wilhelms. Using dynamic analysis to animate articulated bodies such as humans and robots. *Proceedings of Graphics Interface*, 97–104, 1985.

[Wilhelms 87] Jane Wilhelms. Towards automatic motion control. *IEEE Computer Graphics and Applications*, 7(4), April 1987.

[Wyvill 86] Brian Wyvill. Soft. *SIGGRAPH 86 Electronic Theatre and Video Review*, Issue 24, 1986.

[Wyvill *et al* 86a] B. Wyvill, C. McPheeters, and R. Garbutt. The university of calgary 3d computer animation system. *Journal of the Society of Motion Picture and Television Engineers*, 95(6), 1986.

[Wyvill *et al* 86b] Brian Wyvill, Craig McPheeters, and Geoff Wyvill. Animating soft objects. *The Visual Computer*, 2(4):235–242, February 1986.

[Wyvill *et al* 86c] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, February 1986.

[Wyvill *et al* 87] G. Wyvill, B. Wyvill, and C. McPheeters. Solid texturing of soft objects. *IEEE Computer Graphics and Applications*, 7(12):20–26, Dec 1987. Originally presented at CG International 87, Tokyo.

[Zeltzer 82] David Zeltzer. Motor control techniques for figure animation. *IEEE Computer Graphics and Applications*, 2(9), November 1982.