1

OF/DE

1

MICRO

**National Library
of Canada**

**Bibliothèque nationale
du Canada**

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents

Automatic Generation of Natural Motion in Character Animation

by
David Buck

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of
Master of Computer Science

School of Computer Science
Carleton University
Ottawa, Ontario

November 20, 1989
© Copyright 1989, David Buck

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-59016-5

The undersigned recommend to the Faculty of Graduate Studies and
Research acceptance of the thesis

"Automatic Generation of Natural Motion in Computer Animation"
submitted by David Buck, B.C.S.
in partial fulfilment of the requirements for
the degree of Master of Computer Science

Thesis Supervisor

Chair, School of Computer Science

Carleton University
Jan. 16, 1990

Abstract

This thesis explores object-oriented techniques for automatically generating natural motion in computer-assisted character animation. These techniques have been implemented in Elastolab, a tool-kit for creating natural character animation. The computational mechanisms that support Elastolab include facilities to manage the calculations, control the timing for the animation, represent articulated figures, and perform physically-based modeling. These facilities are described in detail. These techniques are then used to model various aspects of an animated character to automatically create convincing animations.

Acknowledgements

I would like to thank my thesis supervisor Wilf LaLonde for his many discussions, ideas, and comments without which this thesis would not have been possible. He proposed many interesting avenues to pursue and my only regret is that I could not explore them all as part of the thesis.

I would also like to express my appreciation to Gerry Paquette for his insights into traditional character animation and for explaining the problems and limitations involved in making traditional hand-drawn animations. It was from these discussions that I learned the importance of fluidity, timing and spontaneity in animation.

Table Of Contents

1	Introduction.....	1
1.1	Goals	1
1.2	Previous Work	3
2	Animation Systems	5
2.1	Disney Style Animation	5
2.2	Principles of Classical Animation	6
2.2.1	Principles for Character Design.....	6
2.2.2	Principles for Directing the Action	6
2.2.3	Principles for Natural Movement.....	7
2.3	Components of a Computer Animation System.....	10
2.3.1	World Modeling	10
2.3.2	Character Modeling	11
2.3.3	Character Control.....	12
2.3.4	Rendering	12
2.3.5	Recording	13
3	Animation Support	14
3.1	The Computational Base	14
3.2	The Dependency Mechanism	14
3.2.1	Relative Objects.....	15
3.2.2	Dependencies	15
3.2.3	Lock and Unlock	16
3.2.4	Relative Object Classes	16
3.2.4.1	Numerical Values	16
3.2.4.2	Points.....	16
3.2.4.3	Paths.....	17
3.2.4.4	Surfaces.....	17
3.3	Animation Clocks.....	17
3.4	Arrows	18
3.5	Articulated Figures	20
3.6	Physically-Based Modeling.....	20
3.6.1	Organizing the Particle System	21
3.6.1.1	Particles	21
3.6.1.2	Particle Groups.....	21
3.6.1.3	Forces	22

3.6.1.4	Particle Systems.....	22
3.6.2	The Equations of Motion for a Particle	22
3.6.3	Oriented Particles.....	23
3.6.4	Springs	24
3.6.4.1	Hookean Springs	24
3.6.4.2	Angular Springs.....	24
3.6.5	Viscous Fluids.....	26
3.6.6	Gravity.....	27
3.6.7	Globular Forces	28
3.7	Numerical Techniques for Differential Equations.....	28
3.7.1	Euler's Technique	29
3.7.2	Runge Kutta Technique.....	29
3.7.3	Distributed Differential Equation Solving	31
3.7.4	Numerical Complications.....	32
4	Character Modeling.....	33
4.1	General Modeling Tools	33
4.2	Modeling the Skeleton.....	33
4.3	Modeling Skin.....	33
4.4	Modeling Joints	34
4.4.1	Previous Work on Joint Modeling	34
4.4.2	Kinematic Modeling of Joints.....	35
4.5	Modeling Muscles	36
4.6	Modeling Fatty Tissue.....	39
4.6.1	Modeling the Stomach	39
4.6.2	Modeling Jowls.....	44
4.7	Modeling Trailing Appendages.....	47
4.7.1	Modeling Tails, Ears, and Hanging Clothes	47
5	Other Considerations	50
5.1	Rendering and Recording Tools	50
5.2	Rendering.....	50
5.2.1	Conventional Rendering.....	50
5.2.1.1	Decomposing High Level Shapes.....	51
5.2.1.2	Hidden Surface Removal	51
5.2.1.3	Shading	52
5.2.1.3.1	No Shading	52

5.2.1.3.2	Flat Shading.....	52
5.2.1.3.3	Gouraud Shading.....	52
5.2.1.3.4	Texture Mapping	52
5.2.1.3.5	Cartoon Shading.....	52
5.3	Recording.....	53
5.3.1	Animation Backgrounds.....	53
6	Conclusions.....	55
6.1	A Summary of Elastolab	55
6.2	The Elastolab Implementation	56
6.3	Future Research Topics	58
7	References	60

Chapter 1

Automatic Generation of Natural Motion in Character Animation

1 Introduction

Man has always had a fascination with images. From the cave drawings of primitive man to the paintings of the Renaissance and modern photography, man has tried to capture reality in images. Until recently, however, these images were static. They were only still snapshots of a constantly changing world. Early devices such as flip books and zoetropes allowed artists to view single pictures one after the other in rapid succession. These animations, however, were limited to only a few seconds in duration. With the advent of film, it was possible to make animations that ran for minutes or hours. Using film, the art of animation flourished and matured. The animated characters became more expressive and believable. They had a certain smoothness and naturalness about them that even live action film couldn't quite capture.

More recently, computer generated animations have begun to be produced. Most computer animations, however, appear stiff and unnatural. They typically lack the spark that made traditional animation so appealing. Recent advances in computer animation provide techniques that can model deformable substances such as cloth [Weil 86], rubber [Terzopoulos 87], clay [Terzopoulos 88a] and foam [Miller 89]. These techniques can help add some life to computer animated characters. The real magic of traditional animation, however, is that the characters are exaggerated. They are caricatures of real life. They don't attempt to make realistic motions but rather try to accentuate the motion in order to highlight the important points.

1.1 Goals

Animated characters in traditional cel animation are quite unique. They squash, stretch, bounce and flop as they move around. When compared to their real-life counter-parts, they seem to be more lively in some way. The motions are not realistic, but they manage to

suspend our disbelief. In some ways, the motion of the animated characters appears more realistic than real-life. The motion looks very natural in the context of the animation because it exaggerates the natural motions found in real life.

The goal of this thesis is to develop a flexible object-oriented tool-kit for automatically creating this kind of exaggerated natural motion in computer animations. The Elastolab tool-kit described by this thesis is an animation and rendering system written in Smalltalk-80™. Elastolab allows an animator to create characters which automatically appear bouncy and lively. These characters are partially controlled by the animator and partially controlled by the computer. The animator plays the role of the director of the animation. He tells the characters how to perform the actions and gestures required to convey the story. The Elastolab system then creates the fluid motion in response to the movements of the character, and renders the animation. It automatically creates natural motions such as the bouncing of a fat character's stomach as he runs, the bulging of the muscles in an arm, and the swinging motion of clothing attached to the character. High realism is not needed in such animations. The only thing that matters is that the animation looks right.

These natural motions are specified using general tools when creating the original character model. The tools that model a bouncing stomach are the same tools that make a cape swing. These tools may be used to automatically animate almost any item connected to the character in response to the character's movements. The animator has an almost unlimited amount of freedom in creating characters that may be automatically and naturally animated. This freedom is critical in making animations where almost anything can happen and usually does.

Since the goal of this thesis is to investigate techniques for automatically generating natural motion, it is not concerned with the problem of providing a user interface for the animator. The problem of rendering and recording the animations is also not a major concern in this thesis. Although Elastolab contains facilities for rendering and recording animations, it uses standard graphics techniques. These techniques are briefly summarized in the Chapter 5 Elastolab does, however, use several novel object-oriented techniques to provide the general tools used in character modeling and natural motion creation. These techniques, the animations tools that they support, and techniques for using the tools to create interesting animations are the primary focus of this thesis.

1.2 Previous Work

Many researchers have already done much work in physically based modeling and character animation by computer. Alan Barr's group at the California Institute of Technology is involved in physically based modeling and teleological modeling [Barr 88a] [Barr 88b]. Teleological modeling is "goal directed" modeling. It determines the movements of objects by determining how those objects behave in response to the rest of the world. By modeling the behavior of birds, for example, flocks of birds can be animated in which the birds avoid collisions, yet tend to stay together [Reynolds 87]. Physically-based modeling is a subset of teleological modeling which animates objects that do not have a mind of their own. Their behavior can be determined strictly by applying the laws of physics.

In order to create chains of rigid links, dynamic constraints may be used [Barzel 88a] [Barzel 88b]. Dynamic constraints use inverse dynamics to calculate the forces that must be applied to the objects in order to fulfil a set of physical constraints. Dynamic constraints can, for example, keep two objects attached no matter what forces are applied to the two objects. Reaction constraints [Platt 88a] are similar to dynamic constraints except that they calculate the reaction forces that cancel other forces which would violate the constraints. Reaction constraints can calculate the forces needed to prevent a ball from penetrating a table.

Physically-based modeling has been used to simulate elastic and inelastic deformable objects [Terzopoulos 87] [Terzopoulos 88a] [Terzopoulos 88b] [Terzopoulos 89]. Elastic objects deform when forces are applied to them but return to their original shape when the forces are removed. Inelastic objects also deform when forces are applied to them, but they do not return completely to their original shape.

The physically-based modeling techniques described above can model cloth and rubber, but they have not been applied to character animation. In addition, the techniques used to simulate deformable objects are very computationally intensive. The techniques provided by Elastolab are designed for character animation and are much faster than those designed for simulating physical objects.

Nadia Magnenat-Thalmann and Daniel Thalmann have developed techniques to model characters by computer by using digitized points to form a polygon mesh surrounding a skeleton [Thalmann 88b] [Thalmann 88c]. They use Abstract Muscle Actions (AMA's) and goal-directed techniques to control the characters [Thalmann 87] [Thalmann 88a]. The goal-directed techniques allow high level actions such as grasping objects. They also use physically-based modeling to model hand deformations when grasping objects [Thalmann 88d] [Gourret 88] [Gourret 89].

The Elastolab system described here uses B-Spline surfaces for modeling the characters instead of the polygon meshes used by Nadia Magnenat-Thalmann and Daniel Thalmann. The techniques that Elastolab uses for animating deformable characters are better suited for modeling the large scale deformations of the character rather than the simulating the subtle deformations of the fingers and hands when grasping objects.

Jane Wilhelms is working on dynamics for character animation [Wilhelms 87a] [Wilhelms 87b] [Wilhelms 88] and on the collision detection and response problem [Moore 88]. By controlling the character completely by physically-based modeling, a very realistic behavior may be achieved. In an animation system that is totally controlled by dynamics and constraints, however, it is very difficult to create complicated actions. In the Elastolab system, the animator uses the simpler kinematic techniques to describe the motion of the character's skeleton. Elastolab then uses dynamics to model the natural effects of the skin, hair, and clothing based on the motion of the character's skeleton.

Chapter 2

Animation Systems

2.1 Disney Style Animation

Most of the techniques used in modern cel animation originated in the Disney studios under the supervision of Walt Disney. A wonderful account of the early days of Disney animation is given by Frank Thomas and Ollie Johnston [Thomas 81], two of Disney's "Nine Old Men". Disney pushed his animators to make their drawings more realistic, yet more exaggerated. His goal was to create an effect which was more real in some ways than life itself.

The Disney animators were often frustrated with the medium of cel animation. It was not cost effective to shade the pictures or to include subtle lines. With these limitations, they had to develop better techniques for conveying emotion and attitudes. Facial expressions could be misinterpreted. They had to use poses and gestures along with simple drawings to convey the proper message. In order for the poses and gestures to be recognized and understood, they had to be exaggerated.

Before an animator can exaggerate an action, he has to study and understand it. To do this, Disney had his animators study motion and form in live-action film and in live models. They would then try to draw the characters and animate them. As they developed their techniques, they found many recurring themes. Over time, these themes were named and became the buzz-words in the animation studios. They became known as the Principles of Classical Animation. These principles served as guidelines to help animators make the characters and the actions more appealing.

In most computer animation, the action is very rigid and unnatural. By applying some of these Principles of Classical Animation to computer animation, the Elastolab animation system can make the animations more flexible and more natural.

2.2 Principles of Classical Animation

The Principles of Classical Animation cover many areas. They describe how to design characters, direct the action, and create natural movement. Some of these principles describe how to make an action clear to the viewer and others describe how to make it look natural. In general, the techniques to make the action look natural require less artistic input and are more easily automated by computer. It is those principles that are incorporated into Elastolab. The principles of animation are summarized below. They are described in more detail in [Thomas 81].

2.2.1 Principles for Character Design

Appeal:

Characters should be appealing to the viewer. This means that there should be some aspect of the character that draws the attention of the audience and holds it. Simplicity of design, charm, and interesting shapes all contribute to the appeal of a character.

2.2.2 Principles for Directing the Action

Staging:

Staging is a technique for making a gesture or movement as clear as possible to the audience. A good rule of thumb is that the action should be clear even in silhouette. To do this, the characters must be positioned properly, the viewing angle must be suitable, and the action should (if possible) take place away from the character's body.

Anticipation:

In order for a character's actions to be understood, the character must prepare for the action before the action actually starts. If a character is about to run off-screen, it should enter a pose that indicates its intent before the running begins. (Often, the run itself is just a blur and the anticipation is the only clue to tell the audience what happened). If a character is going to look up to the sky, it should first look down slightly, pause, then look up. This prepares the audience for the actual action. As the actions become faster, the anticipation becomes more critical.

Secondary Action:

Secondary actions are actions which take place at the same time as the primary action and help to explain the situation. If a character shakes his head as he turns away, the head shaking is a secondary action to the action of turning, yet it helps explain the mood of the character.

Timing:

In order for an action to be properly interpreted, it must be properly timed. In animation terms, the timing determines how many frames are needed to complete the action. Fast actions require fewer frames than slow actions. Changing the number of inbetweens between two keyframes could dramatically change the interpretation of the action.

Solid Drawing:

When drawing characters, the animator should avoid symmetries. The arms of the character should be in different poses. The eyes should be drawn in perspective so that one is smaller than the other. Animators call symmetries "twinning". If there is too much twinning, the character looks flat. By removing the twinning, the character has a more solid appearance.

Straight Ahead Action and Pose to Pose:

There are two ways of planning an action. Straight ahead action involves planning the action frame by frame one after the other. Pose to pose action plans the start pose and the end pose several frames apart, then fills in the inbetween poses to match the end poses. Straight ahead action allows more wild and spontaneous actions. Pose to pose action is better for planning actions and provides a division of labor by allowing the assistant animators to draw the inbetweens.

2.2.3 Principles for Natural Movement**Squash and Stretch:**

"Squash and Stretch" is probably the most well known principle of animation. The concept of squash and stretch was first applied to bouncing balls and was later applied to many aspects of animated characters. The idea is that the character should deform as it moves and interacts with its environment. When

a ball hits the floor, it will squash down and its sides will bulge out. When it bounces up, it will stretch upwards pulling the sides in.

It is important for the squash and stretch to conserve the volume of the object. If the volumes are not conserved, the character appears to be inflating and deflating.

It's also important for the sides to bulge out in a natural way. Too many computer animators think that squash and stretch are linear transformations. This idea is clearly wrong. If a square hits the floor, the sides will bend into curves. They will not remain straight as shown in Figure 2.1.

Figure 2.1

The Elastolab Animation system supports Squash and Stretch by providing the animator with physically-based modeling tools to use for defining the character. The squash and stretch effects are generated automatically by Elastolab using springs, gravity, and other forces.

Follow Through and Overlapping Action:

When animators first started using pose to pose animation, the movements of the characters seemed very jerky and uncoordinated. Since the animators were animating from one pose to the next, the characters seemed to pause at each pose. This pause could be eliminated using overlapping action and follow through.

Overlapping action prevents pauses at the keyframes by making the character start one action as it is completing another. If a character puts on his hat, walks to the door, and opens it, the action of putting on the hat overlaps with

the start of the walk. As the walk slows down, the action of opening the door starts. This overlap creates a continuity of motion.

Follow through makes various parts of a character lag behind the rest. This applies to clothing, hair, tails, fatty tissue, or any other appendage loosely connected to the character. There are two aspects to follow through. First, when the character starts moving, the appendage must trail behind a bit before catching up. Then, when the character stops, the appendage must overshoot before returning to its resting position.

In Elastolab, follow-through is created by the physically-based modeling tools. Elastolab calculates the inertia of the objects to make them automatically overshoot the target position before returning to rest. Depending on the animator-controlled parameters, the object may even swing back and forth slightly before coming to rest.

Slow In and Slow Out:

In most cases, an action does not start or stop suddenly. The action must start gradually and speed up until it reaches its stable speed. Conversely, when the action stops, it must slow down before it actually comes to rest. Slow in and slow out are techniques to perform this speed up and slow down naturally.

The Elastolab system supports slow-in and slow-out in two ways. For kinematically controlled objects, the animator can use spline curves to determine the motion of the character. These splines may be parameterized to perform the slow-in and slow-out operations. For physically based objects, the slow-in and slow-out are controlled automatically by the interaction between inertia and external forces. The objects will accelerate or decelerate naturally.

Arcs:

When drawing inbetween pictures between two keyframes, the easiest and most straight-forward approach is to draw the inbetweens along a straight line path. This technique works well some of the time, but very often, it is necessary to draw the inbetweens along arcs between the starting and ending

frames. By using arcs to position the inbetween drawings, the action looked less mechanical and more fluid.

Arcs are handled in Elastolab by using spline curves for kinematically animated objects and physically based modeling for dynamically controlled objects. In the case of dynamically controlled objects, the arcs are created automatically by the inertia of the object and the forces acting on that object.

Exaggeration:

The actions of an animated character must be exaggerated in order for them to be seen and understood by the audience. Like theater acting, facial expressions cannot easily be seen in an animated character. In order to compensate, the character must use more exaggerated gestures and poses in order to convey the character's attitudes to the audience. In animation, the animators would make the drawings so exaggerated that the individual frames sometimes looked wild. In action, however, the exaggeration would convey the proper message without losing its credibility.

Exaggeration is created in Elastolab by allowing the animator to control the parameters to the physically based models. By choosing different parameters, the motions can appear very stiff, or they may appear very loose and rubbery. The animator can choose the parameters depending on the effect he desires.

2.3 Components of a Computer Animation System

In order for a computer to assist the animator, the computer must provide powerful and general tools for modeling the world and the characters, controlling the characters, and rendering the animation. Ideally, these tools would be easy to use and would provide fast feedback between the computer and the animator. The following sections summarize the major components of a general animation system.

2.3.1 World Modeling

There are many techniques available in the computer graphics field for modeling objects in the world. Many man-made objects are easily modelled with polygons or quadric surfaces such as spheres, cylinders, and cones. Smooth deformable surfaces such as skin and clothing may be modelled by bicubic spline surfaces such as Bezier surfaces [Komatsu 88]

and B-Spline surfaces [Huitric 85] [Nahas 88]. Trees may be modelled by graftals [Smith 84]. Fire and grass can be simulated with particle systems [Reeves 83].

Since the Elastolab system is designed primarily for character modeling and animation, it currently does not support some of the world modeling tools such as graftals and quadric surfaces. When objects are defined using world modeling tools, they are not usually flexible and are difficult to deform. Instead, Elastolab supports bicubic spline surfaces which are much more flexible than quadric surfaces and polygon meshes. Particle systems are also supported since they provide a convenient mechanism for creating natural motion. So far, however, they have not been used to model fire or grass.

2.3.2 Character Modeling

Character modeling is a subset of world modeling, but it has its own unique problems. For this discussion, a character is defined as any world object which can change its shape. A rubber ball bouncing on the floor would be called a character.

The most common type of character, however, is an articulated figure. Articulated figures are most easily modelled in layers as described by Chadwick et al [Chadwick 89]. The lowest layer is the skeleton layer consisting of bones and joints. Most of these joints have limits beyond which they can't bend. On top of the character's skeleton, there are layers of muscle, fat, and skin which can stretch, fold, and twist in response to the movement of the character's skeleton. Finally, the character could have a tail, loose clothing, fur or hair which must also react in response to the skeleton.

In order to design a character, then, the designer must specify not only the shape and colour of the character but also how the various parts of the character can move and how they relate to the movements of the other parts. To support the creation of complex characters, Elastolab provides tools for building the skeleton of an articulated figure and for fleshing the skeleton with layers of muscle, fat, and clothing.

One particularly difficult area is facial modeling. The face is difficult to model because it has many small muscles that deform the face in complex ways. As well, the human vision system is very sensitive to the subtle nuances in a facial expression. Although facial animation was not included in the goals of Elastolab, it may be possible to model faces using some of Elastolab's tools. So far, however, no work has been done in that area.

2.3.3 Character Control

Once a character has been modelled, the animator must specify how that character will move. The traditional computer animation approach for specifying movements closely mimics the pose to pose technique for cel animation. The animator specifies the poses of the characters at the keyframes and the computer fills in the inbetween frames by interpolating the motion.

Another approach which is becoming more popular is to use puppetry techniques combined with real-time animation. The animator controls the character as he would control a puppet. Special sensors send the location and orientation of the animator's hands to the computer which then produces a rendered version of the character. This process is performed in real-time as the animator's actions are recorded by the computer. A post-processing stage can later create a more realistic rendering of the character. The advantage of the puppetry approach is that the animator can make the actions more spontaneous. It is also easier to try the action again if the desired effect was not achieved.

Elastolab currently contains only primitive tools for interactively controlling the character using parametric keyframe interpolation. It is fairly easy, however, to take motions specified by more powerful character control systems and use them in Elastolab. The character's basic motion may be specified with a traditional animation language, a parametric keyframe interpolation system, or an interactive real-time system. The only requirement for Elastolab is that it be able to calculate the locations and poses of the characters at any instant of time. In this sense, all of the character controlling techniques are treated as parametric keyframe animation. The only difference is in the time interval between keyframes and the user interface.

2.3.4 Rendering

In order for the computer to actually draw the images, a graphics rendering system must be used. Fortunately, the techniques for graphics rendering have been well developed. There are several commercial packages available that can do a very good job at rendering the frames. It may, however, be nice to make the images look more like a cartoon and less like real objects. Some techniques for rendering cartoons will be presented later.

2.3.5 Recording

Finally, once the frames have been rendered, they must be recorded. Ideally, there should be some way to do a quick low quality rendering and to view it interactively before actually recording it on film. Once the action looks correct, the animation must be completely rendered and recorded onto either film or videotape. Elastolab contains some simple tools for displaying black and white wire-frame animations in real-time on the computer monitor. Although similar techniques can be used to create full colour animations, the graphics system used by Elastolab is not fast enough to display these full colour animations.

Chapter 3

Animation Support

3.1 The Computational Base

In order to implement an animation system, a software base must be constructed to support the higher level mechanisms. This chapter proposes a hierarchy of object-oriented mechanisms to serve as the base for a sophisticated animation system. Each component in the base typically uses facilities lower in the base and is in turn used by higher-level facilities. Each facility is described here in a separate section. The first section discusses the dependency mechanism which tracks changes in the model and attempts to minimize the amount of recomputation that is needed. The second section discusses animation clocks to provide a timing source for the animation. The third section introduces **Arrows** which perform most of the same operations as transformation matrices without most of the problems. The fourth section covers techniques needed to model the bones and joints of articulated figures. Finally, the fifth section discusses a physically based modeling technique to allow objects to be automatically animated with natural-looking motion.

3.2 The Dependency Mechanism

In a modelled animation system, there are many quantities that change with time. In order to create complex models, however, it is useful to have quantities that depend on other aspects of the world. The amount of bulge in an arm may depend on the elbow joint angle. The location of a character's nose depends on the location and orientation of his neck.

One way of calculating these values is to use constraints. You could imagine using a spreadsheet in which various cells are calculated based upon the values of other cells. By changing the value of one cell, all of the cells that depend on that data (either directly or indirectly) are recomputed.

In most animation systems, all of these cells are recalculated on every frame. This calculation is wasteful, however, if none of the basis values have changed. It would be preferable to recompute only those values which need to be recomputed. It would also be useful to only recalculate the values if they are actually required. There is no sense in recalculating a value if nobody ever uses that value afterwards.

In a complex environment, it would be useful to have a computational mechanism that would automatically track the effects of a change and recalculate the computed values only if they need to be recalculated and if they are required. Such a mechanism is used to maintain the windowing environment in Smalltalk-80™. With some minor alterations, this mechanism may be used to keep track of calculated values in an animation system.

3.2.1 Relative Objects

In Elastolab, there are two distinct categories of values. The first category contains fixed objects such as integers, floats, and points. These objects always have a fixed value independent of the values of other objects. Computing using fixed objects is straightforward since no calculations are required to determine their values.

Relative objects, in contrast to fixed objects, are objects whose values are calculated based on the values of other objects (which may be relative objects themselves). The resulting system can be represented as a directed acyclic graph of objects in which the links denote dependencies.

When a relative object computes its value, it caches the answer (which must be a fixed object). The mechanism proposed here insures that the cached answer (if present) is always correct. When an object changes, the cached answer is discarded and the new value must be recomputed the next time it is needed.

3.2.2 Dependencies

When an quantity in the simulated environment changes, various other quantities may need to be recalculated. In order to track these dependencies, each object maintains a list of the quantities that depend directly on its value. When the original quantity changes, it notifies its dependents of the change and the dependents in turn notify their dependents.

When a quantity calculates its value, it caches the answer. If the quantities that it depends on change, it sets the cached value to nil. This implies that the value must be recalculated the next time it is needed. This mechanism avoids much unnecessary recomputation by caching the answers yet allows the quantity to easily determine if the answer is still correct and recompute it if necessary.

3.2.3 Lock and Unlock

The mechanism for notifying dependents of changes in a quantity uses two messages called **lock** and **unlock**. When the base quantity changes, it must **unlock** itself. The **unlock** method sets the cached value to nil and unlocks all the dependents of that quantity.

When a quantity needs to compute its value, it sends itself a **lock** message. If the value was already cached, the **lock** method just returns the cached value. Otherwise, it locks the quantities it depends on, then uses them to re-calculate its own value.

3.2.4 Relative Object Classes

In an animation system, many different kinds of quantities must be computed. There are numerical values such as integers and reals, three dimensional points, three dimensional paths, and three dimensional surfaces. These quantities may be fixed or relative. When the relative objects compute their values, they calculate and cache fixed values.

3.2.4.1 Numerical Values

Numerical values can be integers, fractions, or floats. These values can be used for joint angles, parametric coordinates for paths in space or the strength of a spring to name a few.

3.2.4.2 Points

Points and vectors are handled identically by the animation system. They are four dimensional quantities with x, y, z, and w components. The extra w component allows perspective transformations to be handled more easily. Vectors in this form are called homogeneous coordinates. In order to convert a homogeneous point into a regular three dimensional point, the point must be normalized using:

$$\langle x \ y \ z \rangle' = \left\langle \frac{x}{w} \ \frac{y}{w} \ \frac{z}{w} \right\rangle$$

Points are used throughout the Elastolab system to locate objects in space.

3.2.4.3 Paths

A path is a line (possibly a curved line) through space. The path is usually parameterized by a parameter between zero and one. Paths supported by Elastolab include piecewise linear paths, interpolating cubic splines, Bezier curves, and B-Spline curves. Paths respond to a **pointAt:** message which returns the point corresponding to the given parameter value.

3.2.4.4 Surfaces

Three dimensional surfaces are parameterized by two parameters. Surfaces supported by Elastolab include Bezier surfaces, B-Spline surfaces, and Ellipsoids. Surfaces respond to a **pointAt:** message which returns the point corresponding to the given parameter value (which must be a two-dimensional point).

3.3 Animation Clocks

In order to control the timing for the animation, an animation clock has been developed. The animation clock is not a real-time clock, but rather a sequential event scheduler. Events may be posted to trigger at a given time. The clock advances by taking the time for the next event and making it the current time.

When an event is triggered, it may send a message to an object, or execute a block of code or both. If the event is a repeating event, it will reschedule itself after it has been triggered.

In a typical animation, the camera will post a repeating event that sends a click message to the camera every 1/24 th of a second. The physically-based modeling system, however, may need to run more often than the camera's click rate, so it will post events that trigger between 2 and 10 times faster than the camera's click rate.

The animation clock maintains a time variable which is updated every time the clock changes. The time variable indicates the time in seconds from the start of the animation. It is a relative object which may have dependents. In this way, values that are time dependent are automatically managed with the dependency mechanism described earlier.

3.4 Arrows

In any computer graphics system, it is often necessary to convert coordinates from one coordinate system to another. Typically, the transformation only involves rotations, translations, and scalings. This is the case when objects are moved about in the world. They may be rotated and translated, but they are seldom distorted. Techniques for modeling distortions will be described later.

Until recently, rotation matrices have been used to represent transformations from one coordinate system to another. A 3x3 matrix can represent any rotation or scaling operation. A 4x4 matrix can additionally represent translations. By performing matrix multiplies, multiple sequential transformations can be combined into a single transformation matrix.

Transformation matrices, however, can introduce skewing due to numerical noise. Repeated applications of a rotation matrix, for example, can eventually create a transformation which is not a pure rotation. This problem has been addressed by [Barzel 88a p.E-51].

Recently, quaternions have been used in computer graphics to represent rotations. A quaternion b is a number of the form:

$$b = w_b + x_b i + y_b j + z_b k$$

where $i^2 = -1$, $j^2 = -1$, $k^2 = -1$, and $ijk = -1$. It follows that $ij = k$, $jk = i$, and $ik = -j$.

Quaternions will be represented as:

$$b = \langle w_b, x_b, y_b, z_b \rangle$$

When the quaternion has unit length ($\sqrt{w^2 + x^2 + y^2 + z^2} = 1$), the x , y and z components represent a vector about which to rotate (scaled by $\sin(\phi/2)$) and the w component represents $\cos(\phi/2)$. Thus, a quaternion describes a rotation about an arbitrary axis by an arbitrary angle. In the following discussion, all quaternions are assumed to have unit length.

When two quaternions are multiplied, the resultant quaternion represents the operation of applying the first rotation followed by the second. It is relatively easy, therefore, to compose transformations using quaternions.

The inverse of a quaternion is simply:

$$b^{-1} = \langle w_b, -x_b, -y_b, -z_b \rangle$$

Quaternions, however, cannot represent translations or scalings. In order to overcome this deficiency, we introduce the concept of an **Arrow**. An arrow is an object that contains a quaternion to keep track of orientation, a vector to keep track of translations, and a scalar to keep track of scalings. Arrows, therefore, represent a transformation composed of a rotation followed by a scaling followed by a translation. Arrows can be represented as:

$$X = \langle x_r, x_s, x_t \rangle$$

Where x_r is a quaternion representing the rotation, x_s is a scalar representing the amount of scaling, and x_t is a vector representing the translation.

Arrows can then be used to transform vectors from one coordinate system into another. This feature makes them useful in hierarchical models such as articulated figures. Transformation of a vector is performed by:

$$v * X = (x_r^{-1} * v * x_r) * x_s + x_t$$

Like quaternions, arrows can be multiplied in order to create a composite transformation. Multiplication of arrows is defined by:

$$X * Y = \langle x_r * y_r, x_s * y_s, x_t * Y \rangle$$

It is often useful to perform an inverse transformation for any given transformation. Calculating the inverse transformation for a matrix is a relatively complex and time consuming task. The inverse of an arrow, however, is simply defined as:

$$\mathbf{X}^{-1} = \left(\mathbf{x}_r^{-1}, \frac{1}{x_s}, (x_r * (0 - x_t) * \mathbf{x}_r^{-1}) / x_s \right)$$

Arrows can perform any three dimensional transformation that does not distort the object. Because they are smaller and faster than matrices and they are not susceptible to skewing errors, Arrows are ideal for use in a three dimensional animation system.

3.5 Articulated Figures

Since most characters are articulated figures, Elastolab provides tools to build characters using a hierarchy of bones and joints. In order to create an articulated figure, some point in the skeleton is chosen as the base. Usually, this point is in the pelvis of the character. The base is then connected to several bones. The bones are connected in turn to joints connected to other bones.

All of the bones and joints maintain a local coordinate system. For bones, the bone itself runs along the z axis of the coordinate system. The transformation required to transform the local coordinate system of some bone or joint into the coordinate system of its parent is stored in the bone or joint using an arrow. All bones and joints also maintain a world transformation which converts the local coordinates into world coordinates. The world transformations are maintained using the dependency mechanism described earlier. If one joint angle changes, the world transformations for all dependent bones and joints are de-cached and must be recomputed if they are needed.

The articulated figure, therefore, defines the skeleton of the character to be animated. The animator can specify the position of the body and the joint angles at the keyframes. The computer can then interpolate the positions and joint angles to create the inbetweens using parametric keyframe animation.

3.6 Physically-Based Modeling

Once the skeleton of the character has been animated, appendages such as clothing, fatty tissue, and muscles must be animated. This animation, however, can be performed automatically using physically-based modeling techniques. Physically-based modeling simulates Newton's Laws of Physics to create natural and realistic motions automatically. By modeling the physics of the model, many of the Principles of Classical Animation arise

naturally. Squash and stretch, overlapping action, follow through, arcs, and slow in and slow out are all natural effects of physically-based modeling. Also, by adjusting various parameters, it is possible to create exaggerated motion.

To animate a physically-based model, Elastolab numerically solves a system of differential equations that describe the behavior of key points in the system. This approach has been used by several other physically-based modeling systems. Since Elastolab uses object-oriented techniques, however, it provides general facilities for organizing the particle system and solving the differential equations.

3.6.1 Organizing the Particle System

One way of modeling a flexible object is to treat it as a system of particles interconnected by springs. This technique has been used by Gavin Miller to control snakes and worms [Miller 88a], by John Platt and Alan Barr to create deformable objects [Platt 88a], and by Demetri Terzopoulos to create plastic and rubber objects [Terzopoulos 87] [Terzopoulos 88b] and to simulate tearing [Terzopoulos 88a] and melting [Terzopoulos 89].

A particle system must, obviously, model the particles. It is useful, however, to organize the particles into groups. Forces may then be applied to the particles to make them accelerate or decelerate naturally. A force may act on specific particles, or it may act on all particles in a group. These components of the particle system will be discussed in more detail in the following sections.

3.6.1.1 Particles

Particles are assumed to be infinitely small points. The particles have mass and they maintain their locations and velocities. It is not necessary for the particle to store its acceleration since its acceleration is recalculated at each step of the differential equation solver. Particles are relative objects that are dependent on the time variable maintained by the animation clock.

3.6.1.2 Particle Groups

It is often useful to organize particles into groups. Some types of forces, for example, apply from all objects in one group to all objects in another group. By collecting the

particles into these groups, it is not necessary to maintain pointers from each particle to all other particles that can affect it.

3.6.1.3 Forces

Elastolab supports three kinds of forces. Particle to Particle forces are forces that apply between two specific particles. Spring forces are good examples of particle to particle forces. Each spring is connected to two specific particles and only affects those two particles.

Group to Group forces are forces that act from all particles in one group to all particles in another group (which may be the same one). Globular forces [Miller 89] are good examples of such forces. Globular forces are forces that act between each pair of particles in a group. Using this technique, it is possible to simulate water, foam, and powder.

Finally, there is a class of forces which have no clear source or a source which isn't modelled in the simulation. These forces are No Source forces. Viscous forces are good examples of No Source forces. A viscous force tends to slow down objects that are travelling in a viscous fluid. The source of the force is the fluid itself which is not explicitly included in the simulation.

3.6.1.4 Particle Systems

A Particle System, then, must keep track of the particle groups in the system, and the forces to be applied within the system. Although it is possible for the particles and groups to keep track of their own forces, it is not easy to manage this efficiently. Newton's Second Law of Motion states that a force on one particle is accompanied by an equal and opposite force on another. It is easier, therefore, for the particle system to tell each force to apply itself to both particles than it is to try to get the particles to calculate the forces on themselves.

3.6.2 The Equations of Motion for a Particle

The equations of motion for a particle can be taken directly from physics texts.

$$\frac{dv}{dt} = \frac{\sum F_i}{m}$$

$$\frac{d\mathbf{p}}{dt} = \mathbf{v}$$

where $\mathbf{v}$ is the velocity of the particle, $\mathbf{F}_i$ is the i 'th force on the particle, m is the mass of the particle, and $\mathbf{p}$ is the position of the particle. Since the forces are known as well as the initial values of the position and velocity of the particle, we can calculate subsequent values of the velocity and position. This is known as an initial value problem. Conventional differential equation solvers can solve these equations to compute the position and velocity of the particle at consecutive time steps.

3.6.3 Oriented Particles

Simple particles can move through space but are unable to spin since they have no orientation. For many applications this model is sufficient. Often, however, it is useful to give particles orientation. To do this, we must be able to apply torques to the particles to make them spin faster or slower.

The equations of motion that control angular velocity and orientation are similar to those that control velocity and position. Forces that increase or decrease the angular velocity of an object are known as torques. The torque is related to the angular acceleration by the equation:

$$\frac{d\omega}{dt} = \tau I$$

Where τ is the torque and I is the moment of inertia. The moment of inertia of an object depends on the shape of the object, the mass distribution, and the axis of rotation. To make it easier to model a simple particle system, however, the moment of inertia is assumed to be a constant independent of the axis of rotation. The moment of inertia is specified by the user and is stored as an attribute of the particle.

Given the angular acceleration of the particle, we can solve for the orientation by:

$$\frac{dQ}{dt} = \frac{1}{2}\omega Q$$

where ω is a vector representing the angular velocity of the particle and Q is a quaternion representing the orientation of the particle.

3.6.4 Springs

Springs provide a flexible and intuitive way to model many natural objects. Springs also tend to be fairly well behaved numerically so long as the spring constants are not too strong. Strong springs produce stiff differential equations which cause problems for the numerical differential equation solvers.

3.6.4.1 Hookean Springs

The most popular type of spring is a Hookean spring. Hookean springs exert a force proportional to the spring's displacement from its natural position. In a one dimensional spring, the force is:

$$F = -k(x - x_0)$$

where k is the spring constant, x is the spring length, and x_0 is the natural spring length. Three dimensional springs work in a similar manner except that the direction of the force is along the length of the spring:

$$\mathbf{F} = -k(\mathbf{x}_1 - \mathbf{x}_0)$$

where $\mathbf{x}_1$ is the location of the end point of the spring and $\mathbf{x}_0$ is the location of the start point. Since the spring force at one end of the spring is opposite to the force at the other end, it is only necessary to compute the force on one end. The force on the other end is just the negation of the first.

3.6.4.2 Angular Springs

Angular springs are the rotational analogues to Hookean springs. An angular spring can be placed on a particle to make it tend to point in some direction (usually towards another particle). Every angular spring has a natural orientation relative to the particle. If the spring is pulled away from that orientation, it exerts a torque on the particle. This torque tries to rotate the particle so that the spring returns to its natural orientation. The strength of the

torque is proportional to the angle between the actual and the desired orientation. This torque is shown in Figure 3.1.

Figure 3.1

The equation for the torque on the first particle is:

$$\tau = \left(\frac{\mathbf{P}_2 - \mathbf{P}_1}{|\mathbf{P}_2 - \mathbf{P}_1|} \times \mathbf{D} \right) \cdot k_a \left(\frac{\mathbf{P}_2 - \mathbf{P}_1}{|\mathbf{P}_2 - \mathbf{P}_1|} \cdot \mathbf{D} \right) \cos^{-1}$$

where $\mathbf{P}_1$ is the location of the first particle, $\mathbf{P}_2$ is the location of the second particle, $\mathbf{D}$ is a vector pointing in the natural direction of the spring, and k_a is the spring constant. This torque must also be applied to the second particle. In doing so, the torque is converted into a force by the equation:

$$\mathbf{F} = \tau \times \left(\frac{\mathbf{P}_2 - \mathbf{P}_1}{|\mathbf{P}_2 - \mathbf{P}_1|^2} \right)$$

In addition to the torque that tries to make the particle point in the right direction, there is also a torque which tries to match the twist between the two particles. If the first particle is twisted about the axis of the spring, a torque will be applied to the second particle to make it twist as well. In order to calculate this torque, the angular spring maintains a natural normal for the first particle. In the equilibrium state, this vector points in the direction of the second particle's up direction. The torque is calculated by orienting the particles to point to each other, then calculating the rotation necessary to rotate the first particle's natural orientation to the second particle's Y axis. This is shown in Figure 3.2.

Figure 3.2

3.6.5 Viscous Fluids

If a physically based particle system uses only undamped springs, the behavior of the system becomes very unnatural since the springs never come to rest. Undamped springs will oscillate forever without any loss of energy. There are two ways to dampen the motion of the springs. The first way is to apply a damping factor to the Hookean spring force. This damping factor would be a force that tends to resist changes in the length of the spring. Usually, damping forces are chosen to be proportional to the rate of change of the spring length.

Another technique is to simulate the spring motion in a viscous fluid. In a viscous fluid, the damping force on a particle is proportional to the velocity of the particle instead of the rate of change of the spring length.

The force applied to a particle by a viscous fluid is:

$$F = -cv$$

where c is a damping factor and v is the velocity of the particle relative to the fluid.

In order to dampen angular motion, a similar technique can be used:

$$\tau = -c_A \omega$$

where c_A is the angular damping constant.

Both types of damping are supported by Elastolab. The viscous fluid damping is usually more effective since it dampens all motion, not just motion along the spring's length. Viscous damping can, for example, make a pendulum come to rest whereas spring damping cannot. Both forms of damping can be used together to make a pendulum slowly come to rest, yet keep the length of the pendulum fairly constant.

3.6.6 Gravity

The force of gravity is a group to group force. Normally, we are only concerned with the force between a massive object and all other light objects. We do not normally care about gravitational interactions between the light objects.

The general form of the force of gravity between two objects is:

$$\mathbf{F} = \frac{Gm_1m_2(\mathbf{x}_2-\mathbf{x}_1)}{|\mathbf{x}_2-\mathbf{x}_1|^3}$$

where G is the gravitational constant, m_1 is the mass of the first object, m_2 is the mass of the second object, $\mathbf{x}_1$ is the location of the first object, and $\mathbf{x}_2$ is the location of the second object.

Normally, however, it is not necessary to simulate gravity so exactly. Most of the time, it is sufficient to model gravity as a No Source force downwards that is proportional to the mass of the object:

$$\mathbf{F} = g\mathbf{d}$$

where g is the gravitational acceleration and $\mathbf{d}$ is a unit vector pointing towards the center of the massive object. Elastolab supports both gravitational models.

3.6.7 Globular Forces

Globular forces are forces that act between each pair of particles in a particle system. Different forces can simulate liquid, foam or powder. Brute force techniques for computing globular forces, however, are $O(n^2)$ in the number of particles being simulated. Such a solution becomes very expensive when simulating a large number of particles.

By making some assumptions about the forces, however, it is possible to reduce the time complexity for computing the globular forces to $O(n)$ for the average case. Since the forces become very weak as the distance between particles increases, we can impose a limit on the radius of influence of a particle. We assume that any other particle outside of this radius is unaffected by this particle. By then subdividing the space into voxels, it is easy to locate particles within a small distance of a given particle and compute only those forces. If there is not much clumping of particles, the computation required approaches $O(n)$.

Elastolab supports globular forces, but since they are not used often, only the brute force $O(n^2)$ technique is provided.

3.7 Numerical Techniques for Differential Equations

The equations of motion can be expressed as differential equations. In order to calculate the positions, velocities, and orientations of the particles, however, we must somehow solve these differential equations. It is difficult to solve the differential equations analytically in a general way, so we must rely on numerical techniques.

The problem we are trying to solve is an initial value problem. Given the initial positions and velocities of the particles and knowing how to calculate the forces and torques on them, determine the positions and velocities of the particles a short time later. In mathematical terms,

Given:

$$\frac{dy}{dx} = F(x, y)$$

$$y(x_0) = y_0$$

we must calculate

$$y(x_0+h)$$

3.7.1 Euler's Technique

The simplest way of solving differential equations is to use Euler's technique. Euler's technique takes the derivative at the current point and linearly extrapolates to calculate the value of the function a short time later:

$$y(x_0+h) = y_0 + hF(x,y)$$

Although this method is easy to implement, it is not very stable. It works well when the springs being simulated are not very strong, but as the springs grow stronger, Euler's method needs to take smaller and smaller steps. Euler's method is known as a first order method since the error term is proportional to h . For many applications, however, this accuracy is sufficient. Elastolab does, therefore, support Euler's technique as a differential equation solver.

3.7.2 Runge Kutta Technique

A more stable technique is the Runge Kutta technique. The particular version implemented in Elastolab is a fourth order Runge Kutta. The fourth order Runge Kutta technique determines the value of the function at the desired end point by using estimates of the derivatives at the start point, the midpoint, and the end point.

The fourth order Runge Kutta method operates in eight stages:

Stage 1: Calculate the derivative of the function at the start point

$$\frac{dy_0}{dx} = F(x, y_0)$$

Stage 2: Estimate the function value at the midpoint by linear extrapolation

$$y_1 = y_0 + \frac{h}{2} \frac{dy_0}{dx}$$

Stage 3: Calculate the derivative of the function at the estimated midpoint

$$\frac{dy_1}{dx} = F(x + \frac{h}{2}, y_1)$$

Stage 4: Make a second estimate of the midpoint value by extrapolating from the start point using the new midpoint derivative

$$y_1 = y_0 + \frac{h}{2} \frac{dy_1}{dx}$$

Stage 5: Calculate the derivative at the new midpoint

$$\frac{dy_1'}{dx} = F(x + \frac{h}{2}, y_1)$$

Stage 6: Estimate the value at the endpoint using the revised midpoint value

$$y_2 = y_0 + h \frac{dy_1'}{dx}$$

Stage 7: Calculate the derivative at the estimated endpoint

$$\frac{dy_2}{dx} = F(x + h, y_2)$$

Stage 8: Calculate a revised endpoint value using the initial points and the four derivatives previously calculated.

$$y_2' = y_0 + \frac{1}{6} \frac{dy_0}{dx} + \frac{1}{3} \frac{dy_1}{dx} + \frac{1}{3} \frac{dy_1'}{dx} + \frac{1}{6} \frac{dy_2}{dx}$$

In the particle system, the differential equations are coupled. Consequently, calculation the forces on one particle requires the locations of all other particles it depends on. In order to compute the functions, then, each stage must be completed for all particles before proceeding to the next stage.

3.7.3 Distributed Differential Equation Solving

It would be convenient if each particle could calculate its own position using some technique (whether it be Euler, Runge Kutta, or some other differential equation solver) independently of the calculations of the positions of the other particles. It should in principle be possible to have some particles computing their positions using Euler's method and others using Runge Kutta. Some spring forces, for example, are very weak and can easily be calculated using the simpler Euler approach. Other spring forces are strong and require the more involved Runge Kutta technique. Each particle should also be able to use different step sizes. Euler usually requires smaller step sizes than does Runge Kutta.

Implementing such a system, however, is complicated by the coupling between the particles. Each particle must be able to ask other particles for their positions at any time and get back a reasonable result independent of the technique that the other particles use for computing their positions.

To solve this problem, Elastolab uses the animation clock to drive the individual differential equation solvers. A differential equation solver may be asked for its value at any time as the animation clock advances. To calculate this value, the differential equation solver will use all of the information available at that time to make a best guess of its value. If all of the particles use the same type of differential equation solver with the same step size, then this technique is identical to the conventional techniques.

Each particle must maintain several differential equation solvers that can solve for the particle's velocity, position, angular velocity, or orientation. These differential equation solvers are first class pluggable objects; i.e., they are objects that can be told what protocol to use to communicate with the model (in this case, the particle). They need to know how to ask for the derivative, how to ask for the function value, and how to set a new function value. In this way, the differential equation solvers are very general.

Each differential equation solver keeps track of its current state. For Runge Kutta, for example, the state corresponds to one of the eight stages described earlier. In each state, the solver will perform some operations, then post a request to the animation clock to wake it up at some later time. If an object is asked for its location, it will use the best information that it has to estimate its location at the given time. By making such an estimate of its

location, this technique can couple particles using a Runge Kutta solver to other particles using an Euler solver. It is even possible to use an Euler solver to compute the velocity of a particle and a Runge Kutta solver to compute the location.

3.7.4 Numerical Complications

Even the Runge Kutta technique sometimes becomes unstable. The most common solution for this problem is to use an adaptive step-size approach. After the Runge Kutta technique computes the next value of the function, the amount of numerical error is estimated. If the error is too high, smaller steps are required. If the error is too low, larger steps may be taken.

Currently, Elastolab does not support adaptive step sizes. The step size must be manually chosen to be small enough to generate the correct motions. It would be useful future enhancement, however, to include adaptive step sizes to make the Runge Kutta technique faster and more stable.

Chapter 4

Character Modeling

4.1 General Modeling Tools

It is impractical to model each part of a character as a special case. There will always be some effect that the animator wants which isn't incorporated in a fixed model. Instead, Elastolab provides a set of general tools for building models. These tools can be used together in a very flexible way to model the character. In this way, the animator is never limited to a fixed class of objects which can be modelled. The tools provided by Elastolab have been described in Chapter 3. This chapter demonstrates how these tools may be used to model various parts of a character.

4.2 Modeling the Skeleton

The skeleton of the character is modelled using the articulated figures described in Chapter 3. By using skeletons as the underlying model, it is possible to create libraries of general actions that can be applied to all characters with similar skeletons. The animation and rendering of the skin and clothing of the character can be done automatically by the animation system. These action libraries, however, have not yet been developed or organized for Elastolab.

4.3 Modeling Skin

Several approaches have been used to model the skin of a character. Many animation systems model the skin using a polygon mesh. This technique makes it easy to generate the data by digitizing real life models. It also lends itself to physically-based modeling techniques by treating the vertices as particles and the edges as springs. In order for such a physically-based model to retain a reasonable shape, however, other internal spring forces must be applied.

Another technique for modeling skin is to use bicubic surfaces. These surfaces are well suited for modeling skin since they form smooth surfaces unlike polygon meshes. It is

difficult, however, to develop physically based models that work correctly on bicubic surfaces.

It is possible, however, to create natural motion by using physically-based modeling to control the control points of the bicubic surface. This is the approach that Elastolab uses to quickly generate convincing natural motion. In the modeling techniques described below, the skin is modelled by B-Spline surfaces with control points that are controlled by Elastolab's tools such as joint dependent calculations and physically-based models.

4.4 Modeling Joints

When modeling an articulated figure, special care must be taken to model the joints so that they look natural when the joint bends. If bones are modelled as cylinders, for example, the cylinders will not meet properly at the joints making the character look like a marionette.

To make the joint look natural, two aspects of the joint must be taken into account. On the outside angle of the joint, the skin must stretch around the joint bones. On the inside angle of the joint, the skin must compress to form a crease.

4.4.1 Previous Work on Joint Modeling

Various techniques have been used to model joints. Wyvill uses soft objects [Wyvill 88a] to model the character and the joints. This approach rounds the skin on the outside angle of the joint, but cannot create the crease on the inside angle.

Thalmann and Thalmann [Thalmann 88b] [Thalmann 88c] [Thalmann 88d] use Joint-Dependent Local Deformations to insure that the joint is rounded. An exponential function is then used to form the crease at the joint [Thalmann 88d]. Since the character is modelled by polygon meshes, however, there are many points that need to be moved and the surface is not smooth.

Komatsu [Komatsu 88] models the skin using Bezier surfaces. This approach rounds the skin on the outside angle by empirically moving some control points on the surface and computing the rest of the control points to maintain smoothness constraints. It creates the crease in the inside angle by merging some control points together to create a first order discontinuity in the Bezier surface.

Chadwick [Chadwick 89] uses Free Form Deformations [Sederberg 86] to bend the arm at the joint. A parameter in the model allows for creasing once the joint bends beyond a given angle.

4.4.2 Kinematic Modeling of Joints

The approach used by Elastolab is similar to Komatsu's except that it uses B-Spline surfaces instead of Bezier surfaces. B-Spline surfaces have fewer constraints than Bezier surfaces and it is easier to maintain first and second order continuity using B-Splines. If the entire character is modelled using B-Spline surfaces, it is easier to control some parts by physically-based models and other parts by kinematic models.

Suppose that an animator wants to model an arm. Since each bone in the articulated figure maintains a local coordinate system, it is a simple matter to define B-Spline control points in that local coordinate system. By arranging these control points in a square around the arm, a roughly cylindrical arm can be modelled. By making the squares smaller as they approach the wrist, the arm can be tapered so that it is wide at the shoulder and narrow at the wrist as shown in Figure 4.1

Figure 4.1

Empirical tests using Elastolab have shown that the elbow looks more natural if the bone does not run through the center of the arm. If the bone is moved down so that there is less fleshy material below and more above, the joint becomes a bit more pointed when bent and the bulge of the arm muscle is more easily exaggerated (as described later).

To make sure that the arm is rounded at the elbow, two extra control points are added underneath the elbow. These control points are calculated to be halfway between the control points at the end of each bone (see Figure 4.2).

Creating the crease inside the joint is a bit more difficult. To do this, two new control points are calculated to be halfway between the ones at the ends of the bones. Two more control points are defined to be halfway between the two new control points and the location of the joint. (The original control points at the ends of the bones are not used). This configuration causes a first order discontinuity inside the elbow (creating a crease) yet insures that the muscles in the upper and lower arm don't intersect. The model is shown in Figure 4.2. For clarity, Figure 4.2 is a side view. As such, each point in Figure 4.2 actually corresponds to two B-Spline control points - a near point and a far point. Compare this figure to Figure 4.1.

Figure 4.2

4.5 Modeling Muscles

In the real world, the muscles in our bodies expand and contract to make the joints bend. In a computer model, however, it's easier to bend the joints and make the muscles expand or contract accordingly. When a muscle contracts, it becomes thicker and when it expands, it becomes thinner. In reality, the amount of muscle bulge is determined not only by the joint angle, but also by the amount of force being applied to the muscle. In an animation, however, it is sufficient to control the muscle bulge with just the joint angle.

There are several different approaches to creating muscle bulge. For most applications, however, a very simple solution will do. If the arm is modelled by Bezier surfaces or B-Spline surfaces, it is sufficient to move two or more control points as a function of the

joint angle. This technique is shown in Figure 4.3. The animation produced by this muscle model is shown in Figure 4.4. Again, Figure 4.3 is a side view, so each point corresponds to two B-Spline control points. See Figure 4.1 for comparison.

Figure 4.3

For more realism, finite element methods may be used to model the muscle bulge [Gourret 88] [Gourret 89], but this technique is very slow. It does, however, have the advantage of being able to model deformations of the muscle caused by contacts with other objects in the world.

Figure 4.4

4.6 Modeling Fatty Tissue

Muscles and joints can be modelled kinematically. No additional information is needed to render them other than the joint angles of the character. If the entire character was modelled kinematically, however, it would appear stiff. There are many parts of an animated character that bounce, jiggle and flap. These effects must be modelled with dynamics.

Traditional techniques for computing dynamic motions have used fine-grained particle meshes and finite element techniques to achieve a realistic result. This approach, however, has two disadvantages when it's applied to character animation. First, the fine-grained approaches tend to create many high-frequency oscillations. As a result, the object looks more like a mold of gelatin than fatty tissue. To remove these oscillations, the system must be heavily damped. The second problem is the amount of time it takes to compute the motion of the character. A typical mesh would contain 1000 elements (10x10x10) to model one part of the character. The movement of each particle is determined by two or more differential equations and quite often, small step sizes are needed. This all adds up to thousands of complex calculations per frame.

The techniques used by Elastolab are simplifications of the traditional finite element techniques. They still work by simulating the laws of physics but with less accuracy than is needed for realistic simulations. Elastolab also allows the physically-based particles to be used as control points of B-Spline surfaces. In this way, Elastolab can create large-scale deformations without small oscillations and with much less effort than traditional techniques which model the surface or the volume of the objects directly.

4.6.1 Modeling the Stomach

Many animated characters have big stomachs that bob up and down as they walk. When the character turns, the stomach lags behind. There are several ways to model this effect. Two simple methods are described below.

The first technique for modeling the stomach is shown in Figure 4.5. The skin is modelled by a B-Spline surface. This surface has 24 control points. The top four and the bottom four are control points that are rigidly fixed to the skeleton. The control points that are along the spine behind the character are also fixed rigidly to the skeleton. The remaining control

points are connected by springs to special points that are fixed to the skeleton but are not control points for the B-Splines.

Figure 4.5

The result of a simple animation is shown in Figure 4.6. In this figure, the fixed control points are not shown. The dynamically controlled control points of the surface are shown as large black dots and the special points that are fixed rigidly to the skeleton are shown as small dots.

Depending on the character, the animator may wish to connect the control points behind the character to springs as well.

This technique creates a nice bobbing effect which can be made realistic or highly exaggerated depending on the choice of spring strengths. Since the B-Spline control points control large areas of the skin, the computation time is greatly reduced and the animation contains no high frequency ripples.

Figure 4.6

Using Elastolab, there are many different ways to model a given character. Some techniques create much different motions than others. The stomach, for example, could be modelled using angular springs. Instead of connecting the springs to points that are fixed

relative to the skeleton, we can connect the springs to the skeleton itself and use angular springs that try to keep the particles at a 90 degree angle to the skeleton. This technique is shown in Figure 4.7.

Figure 4.7

The animation created by the new model is shown in Figure 4.8. Compared to the first animation, this effect looks more like a bag filled with water. This effect may be desirable on many characters.

Figure 4.8

4.6.2 Modeling Jowls

A character with fleshy cheeks or a large fleshy chin can also be modelled with particles controlled by dynamics. Again, the physically-based modelling is used to control B-Spline control points instead of controlling the surface of the face itself.

When modeling jowls, the animator may wish to define the basic surface of the face, then add the chin or cheeks onto the existing rigid surface. In Elastolab, it is very easy to do this because of the flexible ways in which points in space may be specified. In this case, we can choose some points to be on the surface of the face at some given location. This location is specified in the parametric coordinates of the face surface.

The model is shown in Figure 4.9. The end control points of the jowl's surface are connected rigidly to the face surface. Two levels of control points are connected to the face in order to make sure that the jowls blend smoothly with the face. One physically based particle is used. It is connected by a spring to the surface of the face. The connection at the face contains an angular spring that tries to hold up the jowls. The strength of this angular spring determines how much the jowls will sag. Figure 4.10 shows an animation produced by this model.

Figure 4.9

Figure 4.10

4.7 Modeling Trailing Appendages

Various parts of a character trail behind the character as it moves. An animal's tail, ears, or pieces of clothing are good examples of this kind of appendage. In order to animate these properly, these appendages must follow a wave motion behind the character. This type of motion is well described by Preston Blair [Blair 86b]. The desired motion is similar to the motion of a whip.

This wave effect can be produced using the physically-based modeling system by chaining physically based particles together one after the other and connecting them by springs.

4.7.1 Modeling Tails, Ears, and Hanging Clothes

The technique used to model tails and ears uses chains of particles interconnected by springs. These particles will form the base around which the tails or ears are defined. In order to define the surface control points in terms of the particle's coordinate system, oriented particles are used.

When modeling the appendages, the animator must decide whether or not to use angular springs between the particles. If angular springs are used, the appendage will tend to return to a natural curve. If no angular springs are used, the appendage will just drop under gravity. To model a scarf, for instance, no angular springs are used. To model a tail, however, angular springs may be used to allow the animator to raise or lower the tail. The strength of the angular springs between the links determines how much the tail will droop under gravity.

In Elastolab, it is fairly easy to model hanging cloth. It simply consists of a grid of particles interconnected by springs as shown in Figure 4.11. In some cases such as arm sleeves, the particles are arranged into a cylindrical topology. An animation of the cloth is shown in Figure 4.12.

Figure 4.11

The difficulty in modeling clothing, however, is in modeling the interactions between the clothing and the character. The clothing must never penetrate the body of the character, but it is usually in constant contact with the character's skin. This is a difficult problem to solve efficiently and is not currently supported in Elastolab.

Figure 4.12

Chapter 5

Other Considerations

5.1 Rendering and Recording Tools

To create a complete animation system, there are many other components that must be considered in addition to the modeling and control techniques described in the previous two chapters. This chapter looks at the other components that must be implemented including image rendering techniques, and animation recording techniques. Many of these techniques have been implemented in Elastolab in order to create the animations described in Chapter 4.

5.2 Rendering

In recent years, there have been many major advances in the algorithms for rendering realistic computer graphics. Many images created by state of the art systems are almost indistinguishable from actual photographs.

For rendering characters, however, realism can be a double edged sword. When a character is realistically rendered, subtle errors and inaccuracies in the model can be revealed in the shading of the character. Also, when a character is rendered realistically, the audience expects more realistic behavior from the character. Facial expressions become much more important than they would be in a more abstract rendering.

The following sections discuss some rendering techniques that have been implemented in Elastolab. Some enhancements are also suggested that may make the images rendered by the system more like a cartoon and less like a movie.

5.2.1 Conventional Rendering

There are many different approaches to rendering realistic images. Ray tracing is becoming a very popular approach recently because it can render highly realistic images with correct reflections and shadows. For animation, however, ray tracing is still too costly. It typically

takes several hours of computer time to ray trace a single image. Elastolab renders its images by decomposing the B-Spline surfaces into polygon meshes and rendering the polygon meshes onto a Z-buffer.

5.2.1.1 Decomposing High Level Shapes

The lowest level of the rendering software provided in Elastolab only provides support for rendering polygon meshes. The other surfaces such as Bezier surfaces, B-spline surfaces and ellipsoids are rendered by decomposing them into polygon meshes to be rendered.

When high level shapes are decomposed, however, some information is normally lost. In order to prevent this loss, Elastolab provides a way to annotate the vertices of the polygon meshes with high level information when the mesh is constructed. Annotated points are a subclass of three dimensional points in which additional values may be included with the point and referred to by name. In this way, the points in the polygon mesh can store the local surface normal, the parametric coordinates of the original point, or the world coordinates (as opposed to the screen coordinates) of the original point. These annotations may or may not be provided depending on the information required by the rendering system to render the object.

The annotations for a point may be linearly interpolated along with the coordinates of the point as the rendering proceeds. This ability allows the system to estimate the values of these additional parameters for any point inside the polygon.

5.2.1.2 Hidden Surface Removal

In the animation world, Z-buffer rendering techniques are probably the most popular. A Z-buffer is a bitmap in which every pixel contains a number that corresponds to the depth of the object drawn at that pixel. When rendering onto Z-buffers, the entire image is drawn, but pixels are only changed when the new point being drawn is closer to the viewer than the point that's already stored in the Z-buffer. In this way, hidden surface removal is handled on a pixel-by-pixel basis.

One advantage of the Z-buffer approach over other hidden surface removal algorithms is that the image can be rendered in pieces and composited together by just merging the Z-buffers.

5.2.1.3 Shading

After the rendering engine determines that it has to write a pixel to the Z-buffer, it must determine what colour to write. The colour information is computed by the shading algorithm. The following shading algorithms are provided in Elastolab:

5.2.1.3.1 No Shading

No shading simply means that the object being rendered has a colour which is to be drawn directly onto the Z-buffer ignoring any lighting calculations.

5.2.1.3.2 Flat Shading

Flat shading computes the shade for each polygon to be rendered as affected by lighting and uses that shade to draw the entire polygon. This creates a faceted appearance making the polygons in the object clearly visible. If two adjacent polygons have even slightly different colours, the line that separates them can be clearly seen. This effect is called Mach banding.

5.2.1.3.3 Gouraud Shading

Gouraud shading attempts to remove the Mach banding to make the polygon edges less visible. It does this by calculating the shade of the polygons at each of the vertices, then linearly interpolating the shade across the polygon. This technique creates a much smoother image than does flat shading.

5.2.1.3.4 Texture Mapping

Texture mapping is a technique for mapping a picture onto the surface of an object. Texture mapping only affects the colour of the surface, not the shade of the surface. It is possible to mix texture mapping with flat shading or with Gouraud shading.

5.2.1.3.5 Cartoon Shading

Cartoon shading is an experimental technique for rendering cartoons. This technique attempts to recreate the relatively crude shading used in many traditional cartoon animations. When shading is used in cartoon characters, it usually consists of only two shades of colour for any given surface. The light shade is used for the main colour and the

dark shade is used for shadowed areas. This shading approach gives very sharp shadows that are easy to interpret.

To perform this shading, Elastolab calculates the shade at each point on the surface using Gouraud shading, but applies a threshold function to the shading values. Shades that are bright enough are changed to full intensity and shades that are too dim are changed to a lesser intensity.

5.3 Recording

Once the images are rendered, they must be recorded. Ideally, it should be possible to render at least ten to fifteen seconds worth of animation and play it back in real time on the computer screen. This ability would give the animators a chance to see the motion and determine whether to accept the animation or to fix it.

Unfortunately, the system currently running Elastolab is not fast enough to display full-colour animations in real-time. It is possible, however, to get good animation using black and white graphics. The following section discusses a technique that has been used in this system to create flicker-free real-time black and white animations.

5.3.1 Animation Backgrounds

Animation backgrounds are special forms onto which animations may be rendered. These forms store a background picture for the scene. The animations can then be drawn on top of this background.

In order to create smooth animations without flashing, the pictures to be drawn must be prepared off-screen and displayed onto the screen in one operation. Unfortunately, copying a large picture from an off-screen bitmap to the display can take too much time. To solve this problem, animation backgrounds only copy the changes to the display instead of the entire off-screen bitmap.

An animation background uses three separate forms; the background picture which shouldn't change, a foreground picture consisting of the background picture with the animations drawn on it, and the display. Drawing the animations is done in three stages. The draw stage, the display stage, and the erase stage.

In the draw stage, the individual animated objects are drawn onto the foreground form one at a time in deep-to-shallow order . The rectangles surrounding the damaged areas of the foreground form are stored for later use. The display stage draws the foreground onto the display. In order to do this quickly, only the display rectangles that have changed since the last frame are redrawn. Finally, the erase stage draws the background form back onto the foreground form for those areas of the foreground that were damaged during that frame. This process provides a fast and flicker-free way to display the black and white animations on the computer monitor.

Chapter 6

Conclusions

6.1 A Summary of Elastolab

The Elastolab system presented in this thesis provides a flexible object-oriented tool-kit for an animator to use in modeling and animating characters. This thesis developed a computational base for the tool-kit which supports Relative Objects to minimize recomputations, Arrows to represent arbitrary translations and rotations in space, and a novel object-oriented differential equation solver. This computational base was then used to build high level tools for the animator to use. These high level tools include Animation Clocks to control the timing of the animation, Articulated Figures to represent characters consisting of bones and joints, and Physically-Based Modeling to automatically create natural motion. The flexibility of this system was shown by modeling various aspects of a character such as joints (Figure 4.2), muscle bulge (Figure 4.4), a fat stomach (Figures 4.6 and 4.8), jowls (Figure 4.10), and cloth (Figure 4.12).

By using physically-based modeling to control B-Spline control points instead of controlling the surface of the object directly, Elastolab can render natural animations more quickly than conventional physically-based modeling systems. This technique was used in the animation of stomachs (Figures 4.6 and 4.8), jowls (Figure 4.10), and cloth (Figure 4.12). As the animations produced in this thesis have shown, the result is a natural, believable animation which is easier to model and requires far less time to compute.

Finally, the rendering and recording mechanisms provided by Elastolab were presented in Chapter 5. These mechanisms allow the animator to view the animations in wire-frame mode before starting the final rendering. The final rendering can be performed with hidden surface removal (using a Z-buffer algorithm), and with a variety of shading algorithms including flat shading, Gouraud shading, texture mapping, and cartoon shading.

The tools provided by Elastolab are designed to be very general. They can be connected together like building blocks to create very complex characters whose movements are partially controlled by the animator and partially controlled by Elastolab. The jowls animation, for example, used an articulated figure to control the movement of the head and neck, B-Spline surfaces to create the surface of the face, B-Spline surface points to define the control points of the jowls, and physically-based modeling to actually animate the jowls. Once the character is designed, the animator need only play the role of the director of the action. The tiresome job of rendering the individual frames and creating the natural movement is left to Elastolab.

6.2 The Elastolab Implementation

The Elastolab system was written in Smalltalk-80™ with a handful of C primitives to access the colour screen on a Mac II. It was developed over a period of 18 months and consists of 164 classes implementing a over 1400 methods for a total of almost 20,000 lines of code.

The speed of the Elastolab system is still rather slow, but the bottleneck is in the graphics rendering code, not in the physically-based modeling. Figure 6.1 shows the timings for some of the animations presented in this thesis.

Animation	Runge Kutta Step Size	Frames	Computing Time	Rendering Time	Total Time
Arm	NA	10	33 sec.	643 sec.	676 sec.
Stomach	24 steps/s	49	170 sec.	1972 sec.	2242 sec.
Stomach2	24 steps/s	49	381 sec.	1952 sec.	2333 sec.
Cloth	240 steps/s	97	3035 sec.	1740 sec.	4775 sec.

Figure 6.1

There were many difficult problems encountered in the development of Elastolab. Angular Springs, for example, require the calculations of torques, angular acceleration, and angular velocity. These values were implemented as scaled vectors. The orientation of the particle was stored as an Arrow. As well, the torques, when applied to other particles, must be

converted to forces. Using Arrows for the particle's orientation simplified the problem but still resulted in a difficult formulation of the equations.

Another difficult area was the definition of the articulated figures. The problem here was that the joints had to maintain local and world transformations. The local transformations determine how to transform the joint's local coordinate system into the parent's coordinate system. This local transformation, however, was common for all joints connected to the parent, so it should be the parent that calculates the local transformation to prevent the work from being duplicated.

The rendering component of Elastolab was very elaborate including code to perform the viewing transformations, polygon rendering, Bezier and B-Spline rendering using forward differences, and several lighting and shading models. The resulting rendering system was very flexible and could easily be extended to support environment mapping, Phong normal interpolation, and sophisticated lighting models such as Phong or Torrance-Sparrow lighting. As well, new primitive object types may easily be added such as NURB surfaces and superquadrics.

The dependency mechanism used to coordinate changes was very useful because it prevented unnecessary re-calculation of values. It did, however, use a fair amount of memory space since each object had to keep track of the objects that it depends on as well as the objects that depend on it. These collections were stored in instance variables in each object.

The individual forms in an animation took a very large amount of space. In order to reduce the space requirements for the animations and to display them more quickly, the frames were trimmed to eliminate white areas around the pictures.

Finally, when images were being drawn onto a Z-Buffer, there was not enough memory space to store the depth of each pixel on the entire 1024 x 768 pixel screen. To render onto a screen of that size, the Z-Buffer was divided into smaller rectangles and the data for those rectangles was stored into one large file on the disk. As the rendering software accessed each rectangle, it would be read into main memory and cached. A least-recently-used algorithm was used to determine which block to send to the hard disk if the maximum number of cached blocks was exceeded.

6.3 Future Research Topics

Elastolab provides the basic framework needed for a computer animation system that uses physically-based character models. In order to create a complete animation system, however, there is still much work to be done. A more powerful and complete user interface must be developed to allow the animator easy access to Elastolab's tools. It should provide a flexible, yet simple tool for creating and editing the skeletons of the characters. There should also be a system for adding the physically-controlled parts to the character. A real-time interactive editor would make it easier to input the character's actions and to edit those actions afterwards.

With a sufficiently fast graphics system, the physically based techniques described in this thesis should be computable in real-time, allowing the animator to use real-time interactive editors to specify the motion. The characters can be treated like puppets controlled by appropriate input devices like data gloves, joysticks, and body suits. In order to support these interactive systems, techniques must be provided to record and edit the motions of the puppeteer.

Another area for future research is to investigate techniques for modeling special parts of the character such as the face and hands. Facial animation requires much more detailed animator control than does animation of the rest of the character. The face must be very expressive, yet easy to control. It would also be useful if the computer could assist the lip-syncing process to make the character's mouth move in response to a live actor's voice. To animate the hands, it would be useful to implement high-level actions such as "pick up the glass" so that the animator can spend less time positioning joints and more time directing the action.

Finally, it would be useful to keep various libraries of objects, characters, poses, and actions. The animator should be able to quickly access this library and extract stock footage for animating some intricate and difficult actions. Using computers, the stock footage stored in these libraries would be more useful than the libraries for traditional hand-drawn animation. The computer libraries can store character-independent actions and can render them from any angle and under any lighting conditions. With proper use of a library, the task of creating a complex animation could be greatly simplified.

The ultimate goal of such a computer animation system is to combine the powerful computational abilities of computers with the artistic abilities of the animator. With the proper building blocks and a fast and convenient interface to them, the animator can quickly and easily create an animated sequence. Such a system would make it easier to tell a story and, to make that story come to life.

References

- [Abel 88] John F. Abel, "Structural Analysis by Finite Element Methods", SIGGRAPH '88 Course Notes for "Developments in Physically-Based Modeling", ppD1-D16
- [Armstrong 87] William W. Armstong, Mark Green, Robert Lake, "Near-Real-Time Control of Human Figure Models", I.E.E.E. Computer Graphics and Applications, June 1987, pp52-61
- [Badler 87] Normal I. Badler, Kamran H. Manoochehri, Graham Walters, "Articulated Figure Positioning by Multiple Constraints", I.E.E.E. Computer Graphics and Applications, June 1987, pp28-38
- [Baraff 89] David Baraff, "Analytical Methods for Dynamical Simulation of Non-penetrating Rigid Bodies", SIGGRAPH '89 Conference Proceedings, pp223-232
- [Barr 88a] Alan H. Barr, "Introduction to Physically-Based Modeling", SIGGRAPH '88 Course Notes for "Developments in Physically-Based Modeling", ppA1-A5
- [Barr 88b] Alan H. Barr, "Teleological Modeling", SIGGRAPH '88 Course Notes for "Developments in Physically-Based Modeling", ppB1-B6
- [Barzel 88a] Ronen Barzel, Alan H. Barr, "Controlling Rigid Bodies with Dynamic Constraints", SIGGRAPH '88 Course Notes for "Developments in Physically-Based Modeling", ppE1-E82
- [Barzel 88b] Ronen Barzel, Alan H. Barr, "A Modeling System Based on Dynamic Constraints", SIGGRAPH '88 Conference Proceedings, pp179-188
- [Blair 86a] Preston Blair, "Animation: Learn How to Draw Animated Cartoons", Walter T. Foster publisher, 1986
- [Blair 86b] Preston Blair, "How to Animate Film Cartoons", Walter T. Foster publisher, 1986
- [Blinn 87] James F. Blinn, "Nested Transformations and Blobby Man", I.E.E.E. Computer Graphics and Applications, October 1987, pp59-65
- [Blinn 88] James F. Blinn, "Where Am I? What Am I Looking At?", I.E.E.E. Computer Graphics and Applications, July 1988, pp76-81
- [Boyse 79] John W. Boyse, "Interference Detection Among Solids and Surfaces", Communications of the ACM, January 1979, Vol. 22, No. 1, pp3-9
- [Bruderlin 89] Armin Bruderlin, Thomas W. Calvert "Goal Directed, Dynamic Animation of Human Walking", SIGGRAPH '89 Conference Proceedings, pp233-242

- [Chadwick 89] John E. Chadwick, David R. Haumann, Richard E. Parent "Layered Construction for Deformable Animated Characters", SIGGRAPH '89 Conference Proceedings, pp243-252
- [Denber 86] Michel J. Denber, Paul M. Turner, "A Differential Compiler for Computer Animation", SIGGRAPH '86 Conference Proceedings, pp21-28
- [Duisberg 87] Robert Adámy Duisberg, "Animation Using Temporal Constraints: An Overview of the Animus System", Human-Computer Interaction, 1987-1988, Vol. 3, pp 277-307
- [Esakiv 89] Jeffrey Esakiv, Norman L. Badler, Moon Jung, "An Investigation of Language Input and Performance Timing for Task Animation", Graphics Interface '89 Conference Proceedings, pp86-93
- [Gardin 89] Francesco Gardin, Bernard Meltzer, "Analogical Representations of Naive Physics", Artificial Intelligence, Vol. 39, No. 2, March 1989, pp139-159
- [Girard 85] Michael Girard, "Computational Modeling for the Computer Animation of Legged Figures", SIGGRAPH '85 Conference Proceedings, pp263-270
- [Girard 87] Michael Girard, "Interactive Design of 3D Computer-Animated Legged Animal Motion", I.E.E.E. Computer Graphics and Applications, June '87, pp39-51
- [Gourret 88] Jean-Paul Gourret, "Modeling 3D Contacts and Deformations Using Finite Element Theory in Synthetic Human Tactile Perception", SIGGRAPH '88 Course Notes on: "Synthetic Actors: The Impact of A.I. and Robotics on Animation", pp222-230
- [Gourret 89] Jean-Paul Gourret, Nadia Magnenat Thalmann, Daniel Thalmann, "Simulation of Object and Human Skin Deformations in a Grasping Task", SIGGRAPH '89 Conference Proceedings, pp21-30
- [Hahn 88] James K. Hahn, "Realistic Animation of Rigid Bodies", SIGGRAPH '88 Conference Proceedings, pp299-308
- [Haumann 88] David R. Haumann, Richard E. Parent, "The Behavioral Test-bed: Obtaining Complex Behavior from Simple Rules", The Visual Computer, Vol 4, 1988, pp332-347
- [Huitric 85] Hervé Huitric, Monique Nahas, "B-Spline Surfaces: A Tool for Computer Painting", I.E.E.E., Computer Graphics and Applications, March 1985, pp39-47
- [Isaacs 87] Paul M. Isaacs, Michael F. Cohen, "Controlling Dynamic Simulation with Kinematic Constraints, Behavior Functions, and Inverse Dynamics", SIGGRAPH '87 Conference Proceedings, pp215-224
- [Isaacs 88] Paul M. Isaacs, Michael F. Cohen, "Mixed Methods for Complex Kinematic Constraints in Dynamic Figure Animation", The Visual Computer, Vol. 4, 1988, pp296-305

- [Kajiya 89] James T. Kajiya, Timothy L. Kay, "Rendering Fur with Three Dimensional Textures", SIGGRAPH '89 Conference Proceedings, pp271-280
- [Komatsu 88] Koji Komatsu, "Human Skin Model Capable of Natural Shape Variation", The Visual Computer, Vol 3, 1988, pp265-271
- [Lasseter 87] John Lasseter, "Principles of Traditional Animation Applied to 3D Computer Animation", SIGGRAPH '87 Conference Proceedings, pp35-44
- [Miller 88a] Gavin S. P. Miller, "The Motion Dynamics of Snakes and Worms", SIGGRAPH '88 Conference Proceedings, pp169-178
- [Miller 88b] Gavin S. P. Miller, "From Wire-Frames to Furry Animals", Graphics Interface '88 Conference Proceedings, pp138
- [Miller 89] Gavin S. P. Miller, Andrew Pierce, "Globular Dynamics: A Connected Particle System for Animating Viscous Fluids", SIGGRAPH '89 Course Notes for "Topics in Physically-Based Modeling", ppR1-R23
- [Moore 88] Matthew Moore, Jane Wilhelms, "Collision Detection and Response for Computer Animation", SIGGRAPH '88 Conference Proceedings, 289-298
- [Nahas 88] Monique Nahas, Hervé Huitric, Michel Saintourens, "Animation of a B-Spline Figure", The Visual Computer, Vol. 3, 1988, pp272-276
- [Pentland 89] Alex Pentland, John Williams, "Good Vibrations: Modal Dynamics for Graphics and Animation", SIGGRAPH '89 Conference Proceedings, pp215-222
- [Perlin 89] Ken Perlin, Eric M. Hoffert, "Hypertexture", SIGGRAPH '89 Conference Proceedings, pp253-262
- [Platt 88a] John C. Platt, Alan H. Barr, "Constraint Methods for Flexible Models", SIGGRAPH '88 Conference Proceedings, pp279-288
- [Platt 88b] John C. Platt, "Another Introduction to Numerical Analysis", SIGGRAPH '88 Course Notes for "Developments in Physically-Based Modeling", ppC1-C9
- [Pletinckx 89] Daniel Pletinckx, "Quaternion Calculus as a Basic Tool in Computer Graphics", The Visual Computer, Vol. 5, 1989, pp2-13
- [Porter 84] Thomas Porter, Tom Duff, "Compositing Digital Images", SIGGRAPH '84 Conference Proceedings, pp253-260
- [Potmesil 87] Michael Potmesil, Eric M. Hoffert, "FRAMES: Software Tools for Modeling, Rendering, and Animation of 3D Scenes", SIGGRAPH '87 Conference Proceedings, pp85-93
- [Reeves 83] William T. Reeves, "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects", SIGGRAPH '83 Conference Proceedings, pp359-376
- [Reynolds 87] Craig W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model", SIGGRAPH '87 Conference Proceedings, pp25-34

- [Reynolds 88] Craig W. Reynolds, "Not Bumping Into Things", SIGGRAPH '88 Course Notes for "Developments in Physically-Based Modeling", ppG1-G13
- [Sederberg 86] Thomas W. Sederberg, Scott R. Parry, "Free-Form Deformation of Solid Geometric Models", SIGGRAPH '86 Conference Proceedings, pp151-160
- [Sederberg 89] Thomas W. Sederberg, Alan K. Zundel, "Scan Line Display of Algebraic Surfaces", SIGGRAPH '89 Conference Proceedings, 147-156
- [Shoemake 85] Ken Shoemake, "Animating Rotation with Quaternion Curves", SIGGRAPH '85 Conference Proceedings, pp245-254
- [Smith 84] Alvy Ray Smith, "Plants, Fractals, and Formal Languages", SIGGRAPH '84 Conference Proceedings, pp1-10
- [Stekettee 85] Scott N. Stekettee, Normal I. Badler, "Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control", SIGGRAPH '85 Conference Proceedings, pp255-262
- [Terzopoulos 87] Demetri Terzopoulos, John C. Platt, Alan H. Barr, Kurt Fleischer, "Elastically Deformable Models", SIGGRAPH '87 Conference Proceedings, pp205-214
- [Terzopoulos 88a] Demetri Terzopoulos, Kurt Fleischer, "Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture", SIGGRAPH '88 Conference Proceedings, pp269-278
- [Terzopoulos 88b] Demetri Terzopoulos, Kurt Fleischer, "Deformable Models", The Visual Computer, Vol. 4, No. 6, 1988, pp306-331
- [Terzopoulos 89] Demetri Terzopoulos, John Platt, Kurt Fleischer, "Heating and Melting Deformable Models (from Goop to Glop)", Graphics Interface '89 Conference Proceedings, pp219-226
- [Thalmann 87] Nadia Magnenat-Thalmann, Daniel Thalmann, "The Direction of Synthetic Actors in the Film Rendez-vous à Montréal", I.E.E.E. Computer Graphics and Applications, Dec. '87, pp9-18
- [Thalmann 88a] Nadia Magnenat-Thalmann, Daniel Thalmann, "Synthetic Actors as an Interdisciplinary Concept for Human Animation", SIGGRAPH '88 Course notes on: "Synthetic Actors: The Impact of A.I. and Robotics on Animation", pp3-11
- [Thalmann 88b] Nadia Magnenat-Thalmann, Daniel Thalmann, "Construction and Animation of a Synthetic Actress", Eurographics '88 Conference Proceedings
- [Thalmann 88c] Nadia Magnenat-Thalmann, Daniel Thalmann, "Human Prototyping", from New Trends in Computer Graphics, Springer-Verlag-Heidelberg, 1988

- [Thalmann 88d] Nadia Magnenat-Thalmann, Daniel Thalmann, "Joint-Dependent Local Deformations for Hand Animation and Object Grasping", Graphics Interface '88 Conference Proceedings
- [Thomas 81] Frank Thomas, Ollie Johnston, Disney Animation: The Illusion of Life, Abbeville Press, New York, N.Y., 1981
- [Tost 88] D. Tost, X. Pueyo, "Human Body Animation: A Survey", The Visual Computer, Vol. 3, 1988, pp254-264
- [Uchiki 83] Tetsuya Uchiki, Toshiaki Ohashi, Mario Tokoro, "Collision Detection in Motion Simulation", Computers and Graphics, Vol. 7, No. 3-4, pp. 285-293, 1983
- [Waters 87] Keith Waters, "A Muscle Model for Animating Three-Dimensional Facial Expression", SIGGRAPH '87 Conference Proceedings, pp17-24
- [Whitaker 81] Harold Whitaker, John Halas, Timing for Animation, Focal Press Ltd., 1981
- [Weil 86] Jerry Weil, "The Synthesis of Cloth Objects", SIGGRAPH '86 Conference Proceedings, pp49-54
- [White 86] Tony White, The Animator's Workbook, Watson-Guptill Publications, New York, N.Y., 1986
- [Wilhelms 87a] Jane Wilhelms, "Toward Automatic Motion Control", I.E.E.E. Computer Graphics and Applications, April 1987, pp11-22
- [Wilhelms 87b] Jane Wilhelms, "Using Dynamic Analysis for Realistic Animation of Articulated Bodies", I.E.E.E. Computer Graphics and Applications, June 1987, pp12-27
- [Wilhelms 88] Jane Wilhelms, Matthew Moore, Robert Skinner, "Dynamic Animation Interaction and Control", The Visual Computer, Vol. 4, 1988, pp283-295
- [Witkin 87] Andrew Witkin, Kurt Fleischer, Alan H. Barr, "Energy Constraints on Parameterized Models", SIGGRAPH '87 Conference Proceedings, pp225-232
- [Witkin 88] Andrew Witkin, Michael Kaas, "Spacetime Constraints", SIGGRAPH '88 Conference Proceedings, pp159-168
- [Wyvill 88a] Brian Wyvill, "Soft Objects for Character Animation", SIGGRAPH '88 Course Notes on: "Synthetic Actors: The Impact of A.I. and Robotics on Animation", pp67-82
- [Wyvill 88b] Brian Wyvill, Mick Chmilar, Charles Herr, "A Simple Model of Human Animation", SIGGRAPH '88 Course Notes on: "Synthetic Actors: The Impact of A.I. and Robotics on Animation", pp99-110
- [Zeltzer 82] David Zeltzer, "Motor Control Techniques for Figure Animation", I.E.E.E. Computer Graphics and Applications, Vol 2, No. 9, Nov. 1982, pp53-59

[Zeltzer 85] David Zeltzer, "Towards an Integrated View of 3-D Computer Animation".
The Visual Computer, Vol. 1, 1985, pp249-259

END

1 9/0 2/9 1

FIN