Extracting 3D Motion from Hand-Drawn Animated Figures

by

Walter Roberts Sabiston

Bachelor of Science
Massachusetts Institute of Technology
1989

Submitted to the Media Arts and Sciences Section, School of Architecture and Planning, in partial fulfillment of the requirements for the degree of

Master of Science in Visual Studies
at the Massachusetts Institute of Technology
June 1991

Signature of the Author [illegible]
Walter Roberts Sabiston
Media Arts and Sciences Section
May 10, 1991

Certified by [illegible]
Muriel Cooper
Professor of Visual Studies
Thesis Supervisor

Accepted by [illegible]
Stephen A. Benton
Chairperson
Departmental Committee on Graduate Student

Extracting 3D Motion from Hand-Drawn Animated Figures

by

Walter Roberts Sabiston

Submitted to the Media Arts and Sciences Section, School of Architecture and Planning, on May 10, 1991 in partial fulfillment of the requirements of the degree of Master of Science in Visual Studies at the Massachusetts Institute of Technology

Abstract

This thesis describes a 3D computer graphic animation system for use by traditional character animators. Existing computer animation systems hinder expression; they require the artist to manipulate rigid 3D models within complex object hierarchies. A system has been developed that gives animators a familiar means of specifying motion--the 2D thumbnail sketch. Using a simple "flipbook" analogy, the user can quickly rough out a character's motion with hand-drawn stick-figures. Information from the thumbnail sketches is transferred to the user's 3D model by interactively positioning the model's 2D projection. By calculating the amount of foreshortening on hand-drawn limbs, the system is able to determine joint angles for the 3D figure. Automatic inbetweens are created by interpolating along splines passed through the model's joint nodes. The resulting 3D motion can be further edited from any camera angle. By preserving the initial, most spontaneous stage of traditional animation, this system enables users to produce more lifelike motion than is possible with standard keyframing systems.

Thesis Supervisor: Muriel Cooper
Title: Professor of Visual Studies

This work was supported in part by NYNEX, Hewlett Packard, and Paws, Inc.

Acknowledgements

Thanks a lot to Media Lab veterans Dave Small, Brian Anderson, Pascal Chesnais, Alice Lei, and Anne Russell--may you all eventually escape.

And thanks to groovy grads David Young and Michelle Fineblum--good researchers, great dancers, and damn fine human beings.

Thanks to David Atherton, the original Grinning Evil Death.

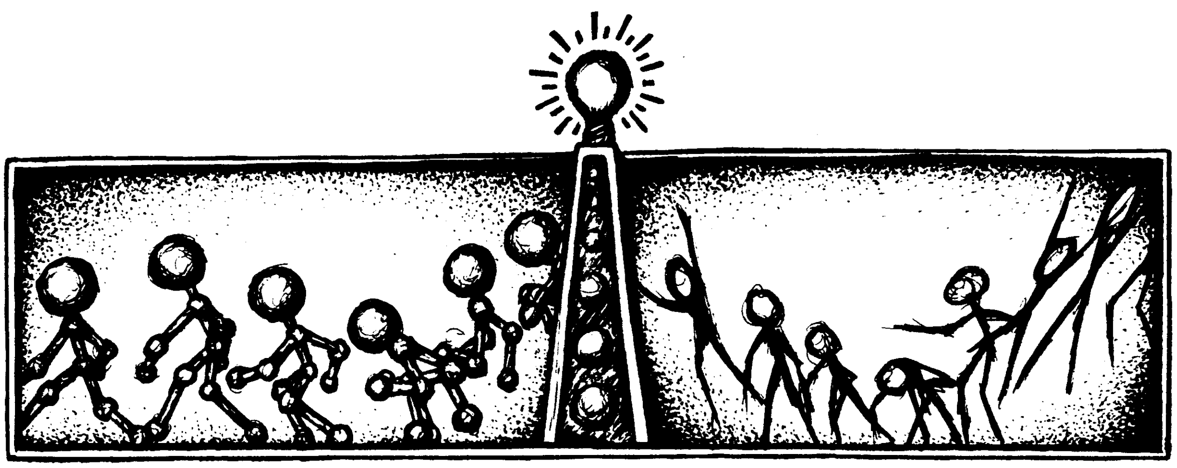
Thanks to my advisor, Muriel Cooper, for keeping the "Arts" in Media Arts and Sciences.

No thanks to the Media Lab's intellectual property policy on artworks.

Thanks especially to my Mom and Dad, Rush, Willy, and Susan--the coolest family in the world.

Table of Contents

1. Introduction. 5
2. Background and Related Work. 9
    2.1 The Principles of Character Animation. 9
    2.2 3D Computer Animation Techniques. 14
    2.3 Related Research. 17
    2.3 Why Character Animation?. 21
3. Approach. 23
    3.1 The 3D Model. 23
    3.2 The Interactive Projection. 26
    3.3 The Reverse Projection Algorithm. 27
    3.4 Attaching the Costume Objects. 33
4. Program Design and Operation. 37
    4.1 3D Skeleton Construction. 38
    4.2 Flipbook Creation with Cel Objects. 39
    4.3 Translating 2D Sketches into 3D Motion. 42
    4.4 Keyframing and Detail Animation. 44
5. Implementation. 46
    5.1 *InstantConversion*. 46
    5.2 *NewConvertLine*. 49
6. Results. 52
7. Conclusion. 56



1. Introduction

In its brief history computer animation has become a powerful form of film-making. During the last ten years, it has brought about a revolution in the fields of scientific visualization, medical imaging, and film special effects. A new class of television advertising, the "flying logo", has developed around the slick imagery characteristic of computer graphics. However, computer animation as a form of popular entertainment is still something of a novelty. Too many computer animated short films rely on flashy visuals to hide weak or nonexistent storylines; they substitute chrome-plating and fancy texture-mapping for personality and wit. Ironically, these qualities epitomize the computer-animated short's predecessor, the traditional hand-drawn cartoon. The enduring popularity of classic Disney and Warner Brothers cartoons testifies to the value of character, plot, and pacing in a short film. Why has computer animation failed to inherit these standards?

Perhaps the problem with computer animation stems not from ignorance of good film-making practice, but from difficulties in working with the medium itself. Computer animation systems are notoriously hard to use; they make the creation of lifelike 3D characters nearly impossible. For example, positioning the hands and feet of a jointed 3D figure can require the trial-and-error setting of up to thirty-six joint angles. It's no wonder that classically trained character animators avoid computer animation systems. In the time it takes a computer user to set up a single 3D character pose, most animators can sketch out an entire motion sequence. Of course, 3D models do have some inherent advantages over hand drawings--once posed, models can be viewed from any camera angle. In addition, the computer can automatically interpolate between poses, reducing the amount of work for the animator. However, these advantages have not won computer animation much support from traditional animators. The systems remain too hard to use. The eighty year history of character animation has provided these artists with a reliable means of self-expression, using a medium they know and love--old fashioned pencil and paper. They are not about to abandon that means in favor of object hierarchies and joint angles. As a result, the only people willing to work with computer animation systems are programmers and computer artists--people with little knowledge of the timing, story, and character development that makes traditional animation so interesting.

The appeal of high-tech photorealism is wearing thin; audiences grew weary of flying logos soon after they appeared on television. With the success of Who Framed Roger Rabbit and The Little Mermaid, traditional animation is enjoying a second renaissance. If computer animation is to survive, it can no longer rely solely on its novelty. In order to produce films with story, style and personality, computer animators need tools that encourage improvisation and experimentation rather than prevent it. What's more, they need a system that will allow them to employ the basic principles of frame-by-frame character motion. Eighty years of accumulated knowledge is too valuable to ignore; if the computer can't learn those principles, then perhaps it can at least serve as a tool for those who already know them.

This thesis demonstrates that traditional animation techniques can be used to control the motion of 3D figures. A system has been developed which employs the initial, most improvisational stage of character animation--the thumbnail sketch sequence. Using a simple "flipbook" analogy, the user can quickly rough out a character's motion as a series of stick-figures. Viewed in rapid succession, these stick-figures approximate the 3D character's motion as seen from one camera angle. The hand-drawn figures therefore serve as makeshift 2D projections of the user's 3D model. Information from the thumbnail sketches is transferred to the 3D model by interactively positioning the model's true 2D projection over the sketches. Assuming the animator has a reasonable knowledge of perspective and foreshortening, the system can work backward from the 2D projection to recover a true 3D pose. The result is a skeleton that moves exactly like the hand-drawn stick figures, only in three dimensions.

Although the conversion from 2-to-3D results in a single wire armature, the armature can be fitted with a "costume" of polygonal solids. Whole object hierarchies may be attached to any limb of the user's 3D skeletal model. In this way the user may build up a library of movements and apply them to a wide range of characters. Facial expression and any other detail movement occurring within these hierarchies is accomplished via standard animation techniques, such as keyframing.

The system developed as part of this research will be of the most use to traditional animators. The tools themselves are simple in the same way as are pencil and paper--your success with them depends on your ability to draw figures in motion. Even for novice users, however, this system should provide a more intuitive means of manipulating 3D figures than is possible with standard commercial animation products. And in the hands of a skilled animator, it is a quite flexible tool, ideal for the animation of complex jointed figures.

This thesis is arranged as follows. Chapter 2 provides background information on animation and presents the related work of other researchers. Chapter 3 describes in detail my approach to the 2D/3D conversion process. Chapter 4 discusses the development of the necessary software tools and their function within a conventional keyframing animation program. Chapter 5 explains the specific implementation of the figure conversion algorithms. Chapter 6 presents results, of the tools as well as initial animation tests. Chapter 7 concludes with a summary and suggestions for future work.

2. Background and Related Work

2.1 The Principles of Character Animation

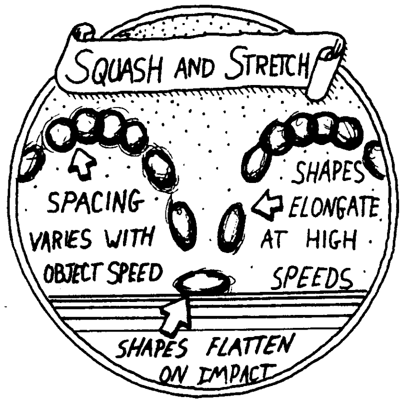
Although character animation is at every stage a two-dimensional process, the final results often look very three-dimensional. Many animated characters, especially those in Walt Disney's classic features, resemble moving sculptures as much as drawings--probably because animators are trained to think of their characters as solid, having mass and volume. The basic principles of animation encourage the idea that animation drawings are projections of solid objects. Animators are taught to draw with the entire character in mind, not just those parts which are visible or moving. A short summary of the fundamental techniques of character animation will reveal the substantial amount of 3D knowledge that they assume. It may also explain why character animation is inherently a frame-by-frame process and is therefore ill-suited to automation or standard keyframing.

Frank Thomas and Ollie Johnston's Disney Animation: The Illusion of Life [1] lists twelve fundamental principles that can be used to create lifelike, appealing character motion. These principles are not necessarily "rules." Rather, they are qualities or techniques that repeatedly turned up in the early Disney animators' most successful experiments:

> The animators continued to search for better methods of relating drawings to each other and had found a few ways that seemed to produce a predictable result. They could not expect success every time, but these special techniques of drawing a character in motion did offer some security. As each of these processes acquired a name, it was analyzed and perfected and talked about, and when new artists joined the staff they were taught these practices as if they were the rules of the trade. To everyone's surprise, they became the fundamental principles of animation. (p. 47)

These are some of the principles "discovered" by those animators and how they are usually applied to 3D animation:

Squash and Stretch -- Objects, especially living things, rarely hold the same shape throughout a motion. A bouncing ball looks more lively if it is drawn as an elongated oval during its moments of great acceleration and as a flattened lump when the ball actually hits the ground. This technique is especially effective in the animation of living creatures; the alternating extension and contraction of muscles in a running figure can cause enormous changes in its shape. The typical notion of squash and stretch does not translate well to computer graphics, however, because 3D objects are rigid. The only convenient way to change a 3D object's shape is to scale it, and scaled objects do not really look like they are "squashing." Instead, squash and stretch of 3D figures can be achieved through clever manipulation of the character's jointed body. In preparing for a jump, people tend to draw their limbs together before leaping forward with all limbs extended. This action simulates squash and stretch without actually changing the shape of the limbs. Although the action is one of folding/unfolding more than squashing/stretching, the effect is the same. John Lasseter uses this technique impressively in Luxo, Jr. [2]--his hinged baby desk-lamp folds itself together before springing open to jump across the floor.


SQUASH AND STRETCH
SPACING VARIES WITH OBJECT SPEED
SHAPES ELONGATE AT HIGH SPEEDS
SHAPES FLATTEN ON IMPACT
Figure 1: Squash and Stretch

Slow In and Slow Out -- When inbetweens are used in traditional animation, they are rarely spaced evenly between the key drawings. Instead, the inbetweens tend to cluster near one of the two keyframes, with fewer drawings near the center of interpolation. The technique is called

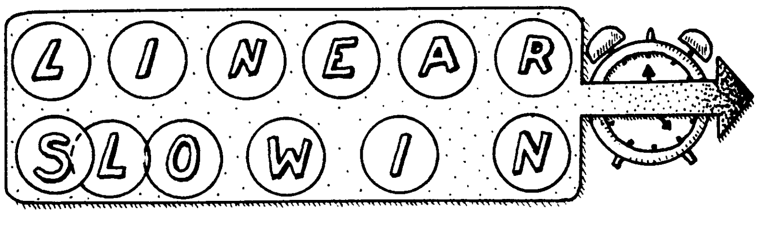


Figure 2: Linear vs. Slow In

slow-in/slow-out because the action slows down as it approaches each keyframe. The purpose of slow-in/slow-out is to emphasize the more important key drawings while moving quickly between them. This is the one basic principle of traditional animation that is very easy to apply to computer animation. Since most computer-generated motion is interpolated, the user can easily apply mathematical curves to the spacing of inbetweens. If this technique is over-used, however, the motion tends to look erratic and mechanical.

Straight Ahead Action and Pose to Pose -- These are two alternative methods for producing frame-by-frame motion. Animation is called straight-ahead when the animator begins with the first frame in a scene and works forward one frame at a time--the results are loose, frantic, and unpredictable, since there is no overall plan to the motion. In pose to pose animation the artist works out the sequence's most important drawings first, then returns to fill in the motion. Neither of these methods really applies to standard computer animation, because it isn't a frame-by-frame process for the artist. The system described by this thesis restores the frame-by-frame process, and so these two techniques again become relevant.

Anticipation -- This technique applies to any kind of animation, because it has more to do with storytelling than drawing. Anticipation is merely a way of letting the viewer know what is about to happen next. If a character is preparing to pitch a ball, it will lean back with its leg in the air and its hand behind its head before the actual throw. In reality the windup gathers momentum for the throw; however, it also tells the viewer that the pitch is coming. Without that anticipatory move, the throw has a very good chance of being missed entirely.

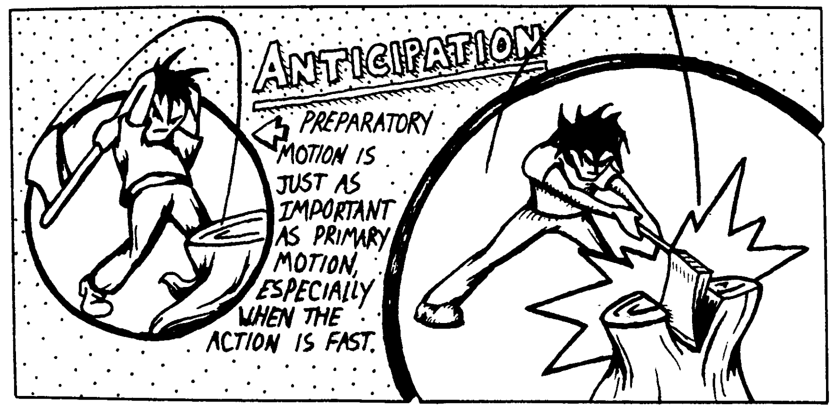

ANTICIPATION
PREPARATORY MOTION IS JUST AS IMPORTANT AS PRIMARY MOTION, ESPECIALLY WHEN THE ACTION IS FAST.

Figure 3: Anticipation

Overlapping Action -- This is one of the most important concepts in character animation. It is the idea that single bodies are actually comprised of several moving parts and that those parts can move independently. When a character comes to a sudden halt, its arms and legs may continue to move, as will any loose flesh or baggy clothing. The overlapping action of these parts keeps the animation "alive." Otherwise a still figure would be reduced to just a static drawing. Overlapping action is harder to convey in computer animation, simply because there aren't as many loose objects to swing around. Most computer-generated figures have no moving clothing; their flesh looks like solid plastic. However, overlapping action can still be applied to the movement of arms, feet, or perhaps a tail. Its use almost always enhances the "life" of a character, so even a little overlapping action is better than none.

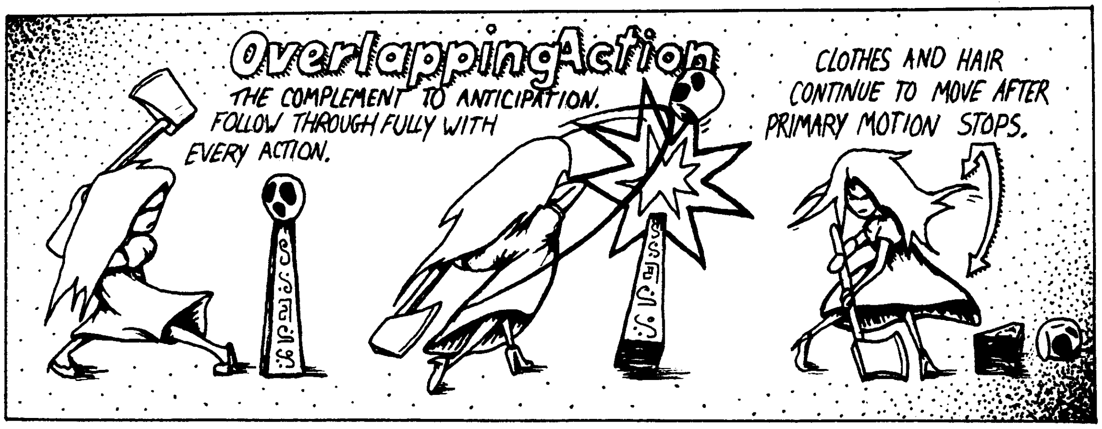

OverlappingAction
THE COMPLEMENT TO ANTICIPATION. FOLLOW THROUGH FULLY WITH EVERY ACTION.
CLOTHES AND HAIR CONTINUE TO MOVE AFTER PRIMARY MOTION STOPS.

Figure 4: Overlapping Action

Solid Drawing -- Disney artists are always encouraged to think of their characters as three-dimensional. Every drawing should convey the correct weight, depth and balance. During the course of an entire animation, a character is likely to be seen from many angles. Being able to draw a character from multiple perspectives is therefore a prerequisite for animators. This principle doesn't really apply to most 3D computer animation, since the computer renders the character and can do so perfectly from any angle. However, solid drawing is vital to the success of this thesis, which relies on the animator's drawing ability in order to determine 3D position.

The principles of character animation obviously contain a wealth of information about conveying meaning through motion--information that applies to the 3D world as well as the 2D. Why do computer animation systems seem to deliberately discourage the use of this knowledge? Most of them make keyframe definition such a chore that frame-by-frame character positioning becomes impossible. Perhaps software designers assume that users are either unable or unwilling to create an animation frame-by-frame.

At one time the only people with access to computer animation equipment were programmers. Most programmers have little experience with traditional character animation; hence, they are not particularly interested in a computer animation system that requires a lot of drawing. They want the computer to do the work! After all, one of the exciting things about computer animation is that it eliminates the tedious, repetitive elements of animation--effectively minimizing the work required of the user. Unfortunately, time-saving strategies are almost always synonymous with a reduction in quality. Most computer animation is lifeless, characterized by jerky interpolated motion and meaningless camera fly-bys. Attempts to incorporate the principles of traditional animation in computer animation have met with only limited success--computer animators are trying to use frame-by-frame motion principles on systems with interfaces that prevent the necessary user control.

Improving technology and dropping prices are making 3D computer graphics available to an enormous number of people--many of them artists. It is no longer necessary to ignore solutions simply because they require some minimum amount of user skill. A growing number of computer users are both willing and able to produce frame-by-frame 3D animation on a computer. The system proposed by this thesis will hopefully give them the necessary tools.

2.2 3D Computer Animation Techniques

Realistic motion is a primary goal of nearly all computer animation research. Even so, most of this work is being done within two very specific areas--*dynamic simulation* and *kinematics*. Neither of these fields is closely related to traditional animation; however, they are the only methods available to most users. A description of them will clarify the need for new techniques.

Dynamic Simulation

Dynamic simulation is the use of numerical integration to calculate the exact behavior of objects in a physical environment. In other words, it is the attempt to simulate real-world situations on a computer. Objects in a dynamic simulation have mass, density, and inertia; they respond correctly to the applied forces of torque, gravity, and friction. Dynamic simulation can be broken into two categories, forward and inverse dynamics. In forward dynamics motion is calculated from a set of forces applied to an object or objects. Results of animation produced in this way are unpredictable--the animator has control only over the initial conditions. The results of inverse dynamic simulations are easier to control, since the animator specifies a desired ending motion--the system's task then is to find a set of applied forces that will act on the object in order to produce that motion. A more comprehensive introduction to dynamic simulation is found in Wittenberg's Dynamics of Systems of Rigid Bodies [3].

Motion produced via dynamic simulation is very realistic, usually far beyond the abilities of a traditional animator. However, dynamics are computationally expensive and

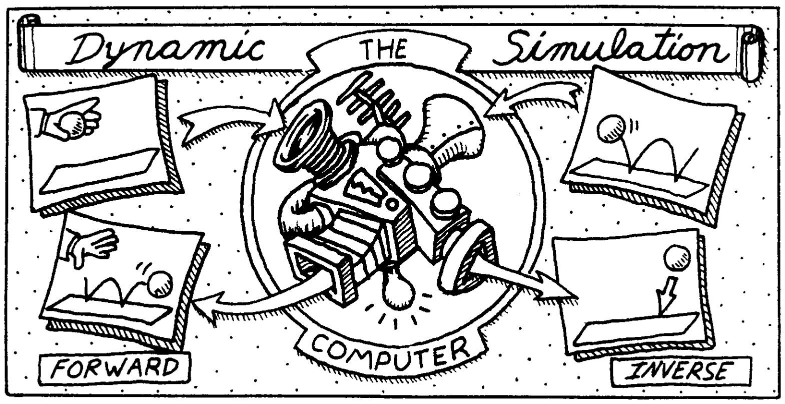


Figure 5: Dynamic Simulation

tremendously difficult to control. Once a physical simulation is initiated, the equations take over completely. The animator can't do anything but sit around and wait for results. If there are any problems, the entire motion must be simulated again with different initial conditions. This trial-and-error method of animation is useless to a traditional animator. Cartoon characters have personality and motivation--traits that require a great deal of user control to be done believably. Unfortunately, dynamic simulation lacks that flexibility.

Kinematics

The other major form of motion research, *kinematics*, has slightly more in common with character animation. At least, it is rooted in the traditional animation idea of *extremes* and *inbetweens*. Kinematic animation systems are based on the idea that the user need only specify a few "key" positions in a jointed figure's motion. The intervening frames are called inbetweens--they are computed automatically by interpolating between the user-specified key frames. For this reason kinematics is often called *keyframing*. The notion of keyframing is popular, for it dramatically reduces the amount of work required of an animator. Unfortunately, many computer artists rely too much on the computer's ability to generate inbetweens. They construct long animated sequences with only three or four keyframes separating hundreds of inbetweens. As a result, the animation is mechanical and lifeless--it looks like it was generated by a computer, because it was. In traditional animation only three or four inbetweens separate the extreme drawings. A computer animation with that ratio would stand a much better chance of looking realistic. Unfortunately, such animation is impractical if not impossible with most systems. Single keyframes take so long to establish that animators have to settle for far fewer than are necessary.

There are two types of kinematics--forward and inverse. Forward kinematics is probably the crudest, most difficult way of manipulating a 3D figure. Unfortunately, it is the easiest method to program and is thus employed by most animation systems. In a forward kinematic system, 3D figures are represented as

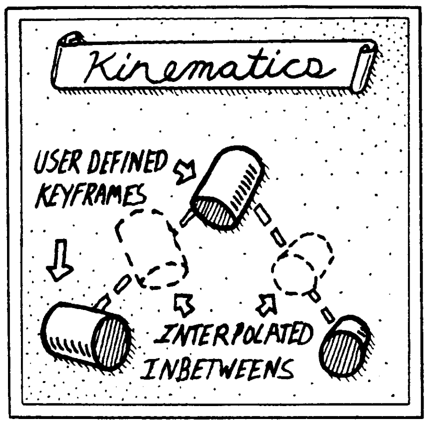


Figure 6: Kinematics

hierarchical object trees. The user creates motion by recording a series of keyframes and interpolating between them. The process of positioning such a figure in space can be frustrating--placement of the feet alone requires the trial-and-error setting of eighteen different joint angles (X,Y, and Z for each of the hips, knees, and ankles). More difficult is moving that figure's body while holding one of its feet still--the fixed hierarchical structure prohibits even that simple action. Complex character animation with these systems is rarely attempted. Even basic animation tasks--for example, a character walk cycle--are extremely difficult to implement. The reason--changing hierarchical joint angles is just not a natural way to position the limbs of a 3D figure.

Forward kinematic techniques have also been used procedurally to create realistic motion. Zeltzer[4] describes a system using kinematics to simulate the walk cycle of a human figure. While this approach is fairly successful at achieving realistic motion, it shares the problem of dynamic simulation--the animator has very little control over the figure's movements.

Inverse kinematics is probably the best existing method for manipulating a 3D character. It does not require angle settings for all of the joints on a limb. Instead the user can simply position the limb endpoint. The intervening joint angles between the end of a limb and its base are computed automatically. To animate a figure's leg, for example, the animator can interactively grab a foot and move it around on the screen. The knee joint angle is computed automatically. Girard and Maciejewski's PODA system demonstrates the use of inverse kinematics to aid the positioning of jointed figures for keyframing [5]. Their successful results are showcased in the animated film "Eurythmy" [6].

Of the available methods for computer animation, inverse kinematics seems to offer the most control without overwhelming the user in tedious detail. Unfortunately inverse-kinematic animation systems are a rarity; mathematical instability of the procedures and the difficulty of deriving single solutions have kept this technique experimental. The unavailability of these systems is not their only disadvantage. Even the best inverse-kinematics animation program does not offer the immediate feedback or flexibility of a hand-drawn flip-book. Three-D models are rigid; they do not "squash" and "stretch." Existing 3D animation systems, no matter how sophisticated, have a common limitation--they force the user to work in three dimensions to create something that is ultimately two-dimensional. A 3D animation system for use by character animators should not have this limitation. It should allow them to work, at least initially, in a more familiar medium.

2.3 Related Research

This thesis proposes the use of traditional character animation techniques as a means of producing 3D animation. I am unaware of any other research projects with the same goal. However, many animators incorporate the basic principles of hand-drawn animation in their work. In addition, some researchers have attempted to construct kinematic and dynamic simulation systems with built-in knowledge of those principles.

John Lasseter

The best computer animators strive to give their work the spirit and personality of hand-drawn cartoons. Of these animators, Pixar's John Lasseter is the most well-known--and arguably the most talented. Lasseter worked as a character animator for Walt Disney Studios on several feature films, including The Fox and the Hound and The Black Cauldron. While at Disney he participated in The Wild Things Test, a project combining hand-drawn characters with 3D computer generated backgrounds. However, Lasseter gained his reputation in the computer animation field for the work he did later at Pixar, the computer division of Lucasfilm. As the director of several short films, Lasseter established the standard for 3D character animation. One of those films, Luxo, Jr., has the "honor" of spawning the most derivative sequels in the industry's short history--for good reason. The two-and-a-half minute short is probably the finest example of traditional animation techniques applied to 3D computer graphics. In 1989 another Lasseter film, Tin Toy [7], became the first computer-generated short to win an Academy Award. These films demonstrate the overwhelming importance of character, story, and timing to the success of an animation.

How does Lasseter give life to his characters? They have a real screen presence surpassing that of most traditional cartoon characters. He has said that all it takes is knowledge of good filmmaking and basic animation techniques. Although Pixar's animation software is not available to the public, Lasseter claims that a standard kinematic keyframing system is sufficient to duplicate his techniques. His "secret" seems to be his willingness and ability to spend an inordinate amount of time to do 3D character animation the "right way". For example, some of the sequences in Luxo, Jr have no inbetweens whatsoever. That is, Lasseter explicitly defined the character's position at every point in the motion--thirty frames per second of screen time. Most computer animators would balk at the prospect of creating thirty keyframes for every second of screen time--they are too reliant on interpolation. Perhaps if 3D figures were easier to manipulate, more animators would be willing to spend the extra time. This thesis aims to make character manipulation more efficient--the idea is not to eliminate keyframing altogether, just to make it less of a crutch for animators.

Witkin and Kass

Andrew Witkin and Michael Kass's *spacetime constraint* system [8] takes the opposite approach to 3D character animation. It assumes that the so-called "principles" of animation are merely by-products of physically accurate motion. Squash and stretch, anticipation, and overlapping action are all observable in real-world phenomena. Witkins and Kass propose a dynamic simulation system in which 3D characters exert their own forces to accomplish user-specified goals. The user gives the character something to do; the system figures out how to do it, given the character's physical structure and available resources--muscles, springs, motors, etc. The result is computed motion that exhibits the principles of hand-drawn animation.

Spacetime constraints are probably the most effective available means for producing character animation automatically. To demonstrate their system, Witkin and Kass model the jumping desk lamp from Pixar's Luxo, Jr. Their results are comparable to Lasseter's; in addition, they achieve several interesting variations on the same jumping motion simply by changing the simulation's initial conditions. For example, they model the desk lamp with a base five times heavier than normal; as a result, the little lamp has to yank upward much harder to make the jump. A traditional animator would have to start over from scratch in order to make the heavy-base modification.

Unfortunately, spacetime constraints have not been incorporated into any commercial animation packages; the techniques are still relatively experimental. Also, in comparison with Lasseter's original, the Witkin/Kass desk lamp looks a little stiff and unnatural. Traditional animators tend to exaggerate the most appealing characteristics of a motion. The result is something that actually looks more realistic and natural than the physically correct motion. The spacetime constraint system cannot account for this tendency toward exaggeration. Future research may correct the problem; if so, spacetime constraints will offer future animators an exciting alternative to the traditional frame-by-frame approach. Not all animation is realistic, however, and a system offering frame-by-frame character positioning will always be valuable to those who want to do it "the hard way."

Scripting by Enactment

Since film/video animation is a two-dimensional medium, perhaps it makes sense to work with 3D objects via their 2D screen projections. For example, Delle Maxwell's "graphical marionette" project [9] attempts to animate a 3D figure by automatically matching it to video source material of a human "puppeteer" in a body-tracking suit. Recovering 3D information from video seems like the ideal way to script animation--how better to describe a motion than to enact it?

Maxwell's thesis describes a computer animation system that receives its input from 3D body-tracking hardware. The hardware records 3D motion by analyzing video of a human performer in a body-suit. LED's on the body suit identify the performer's major limbs and joint positions. As the performer enacts a motion, the 3D position of each LED is determined automatically. That information is used to calculate positional and rotational limb transformations, which are applied to the user's jointed 3D character.

Maxwell's ultimate goal is similar to my own--that is, to produce the motion of a 3D jointed figure from a 2D projection. However, her source material is video--she is confronted with an abundance of image-processing problems before 2D/3D conversion even becomes an issue. The task of this thesis is more straightforward--I am not attempting image-recognition. The user does the work of matching 2D skeleton projections to hand-drawn frames--a task surprisingly difficult for the computer but fortunately very simple for the user. Thus I am able to completely bypass the image-processing problem that has hindered Maxwell and others.

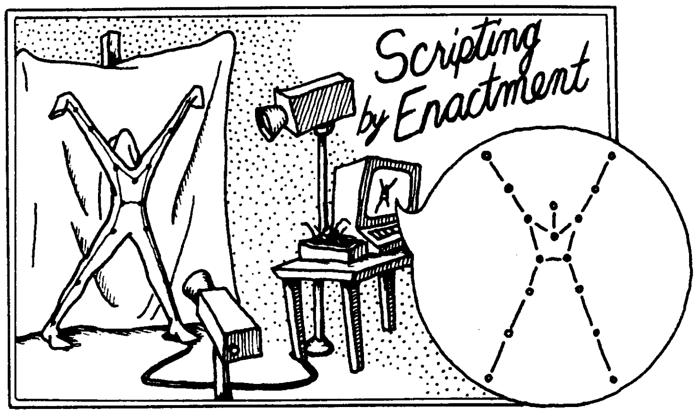


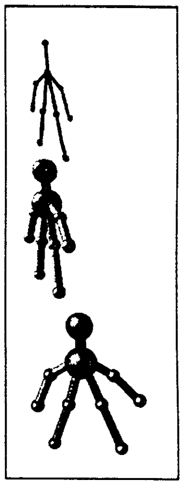
Figure 7: Scripting by Enactment

Scripting by enactment projects are a form of 3D *rotoscoping,* in which the movements of a live model are transferred frame-for-frame to the animated character. In many Disney films hand-drawn rotoscoping is used to perfect the movements of human characters. The graceful dancing sequences in Cinderella, for example, were created by tracing live film footage as each frame was projected onto the animators' drawing boards. Rotoscoping in 3D is more difficult, because the process is automatically performed by a computer. In addition, the task is more complex than simply tracing the silhouettes of filmed performers. A computer rotoscoping system must recover true 3D limb position and rotation data. This information cannot be recovered from a single camera view; a perpendicular two camera setup is required. Coordinating the two views is a major problem--the occlusion of lights, combined with differences in perspective and speed, often prevent the recovery of reliable 3D data. These problems, plus the awkward, expensive hardware requirements, have prevented the widespread use of computer rotoscoping.

2.3 Why Character Animation?

Why is actual character animation a better solution than those described above? Is drawing each frame by hand really the best way to create motion? Perhaps not, if the systems described above were fully reliable, easy-to-use, and readily obtained. Unfortunately, they are not. Scripting by enactment is a great idea, but it does require an awkward setup. Also, it's really only practical for the animation of real-world objects. If your character has six legs, two wings, and a tail, finding a live model to videotape is going to be pretty tough. Witkin and Kass' system is certainly interesting, and it may eventually allow a novice to rival the work of master animators. However, the system is still experimental--and for now, at least, their results pale beside Pixar's original.

Although traditional animation is probably the most time-consuming way to make a film, the initial stages of motion development are actually quite efficient. A character animator can often rough out several seconds of action in an afternoon. This may seem like a lot of time to devote to three or four seconds of footage; however, by the time those four seconds make it to film, at least four other people will have spent a week inbetweening, inking, and painting the final frames. In comparison the animator's single afternoon seems mercifully short. That extra week of detail work contributes very little to the meaning or character of the sequence. After all, the whole point of the thumbnails is to quickly, efficiently summarize the scene. This thesis adapts that initial, most creative stage of character animation for use in a 3D environment.



3. Approach

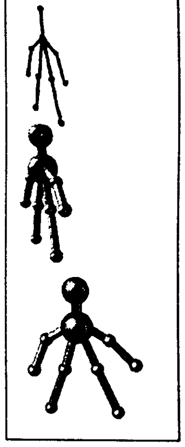
Perhaps three dimensional control is unnecessary to describe the motion of a jointed figure; an interactive projection of the figure may suffice. This chapter presents an approach to transforming 2D stick-figures into 3D jointed models. The explanation is divided into four parts. The first is a description of this system's representation of 3D models. The second section explains how such a model can be positioned interactively over a hand-drawn image. The third section describes in detail the algorithm used to manipulate 3D models via their 2D projections. In the last section, a method is proposed for attaching polygonal objects to the moving skeletons.

3.1 The 3D Model

This research proposes a new tool for animating 3D models. In order to fully understand its implementation, one must first be familiar with the construction of the 3D models themselves. Computer models are very different from cartoon characters; models don't "squash" or "stretch." Their bodies are made up of rigid pieces, which link together like the limbs of a plastic doll. That analogy is appropriate, because 3D figures are positioned in the same way as are dolls--by bending their limbs at the joints formed by the intersection of two rigid body parts. In this respect, a 3D character can be envisioned as a simple stick figure with a suit of objects attached to it. The objects themselves don't change shape--they just rotate with respect to the other body pieces.

Extended Figure 1: In the upper-right corner of the next thirty pages you will find three images of a jumping figure. The top image is hand-drawn by the animator. The middle image is the system's attempt to match that sequence in 3D. The lower image is the same 3D sequence as seen from the front. By slowly flipping the pages from back to front, you will see the figures execute their jump.

In order to describe the motion of a 3D figure, it is unnecessary to see its hands, feet, torso, etc. A simple stick-figure representation will suffice. For this reason I have separated the 3D character description into two structures--the *costume* and the *skeleton*. The costume is the set of rigid polygonal solids that normally constitute a 3D character--its clothes, arms, legs, shoes, hat, etc. The stick-figure, on the other hand, is the character's structural description. It isn't a visible part of the character; rather, it is a symbolic representation illustrating the figure's limbs and joints.



Although the skeleton is nothing more than a series of 3D points connected by segments, it's actually the most important part of the whole animation process. One premise of this thesis is the idea that 3D character motion can be completely represented by stick-figures--a user should be able to animate characters of great complexity simply by describing the motion of their skeletons. If so, the conversion from 2D to 3D is greatly simplified. That process no longer involves image or shape recognition; it is just a problem of assigning depth information to a 2D stick-figure. Using skeletons instead of full figures is also beneficial to the artist. With this system an animator does not have to draw "real" characters, with flesh, clothing, facial expressions, etc. All that is necessary is a thumbnail stick-figure indicating the position and angle of each limb. The system can't model complex attributes like baggy clothes anyway--it can only pose the 3D skeleton and then fit the "costume" of objects onto it. A simple stick-figure is easy to draw and provides the system with the limb angles necessary to position a 3D jointed figure. Thus, the representation of 3D characters as nothing more than costumed stick-figures simplifies the animator's job as well as the computer's.

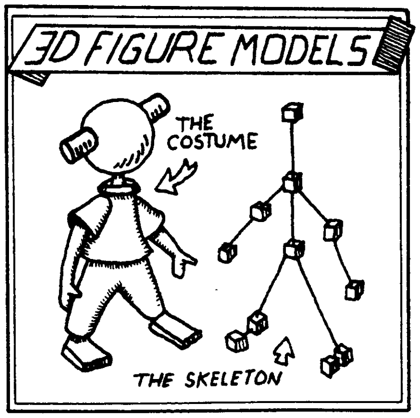
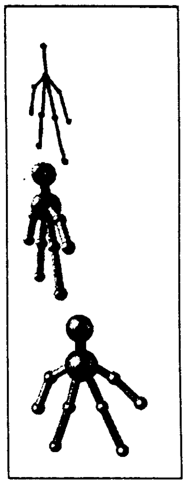


Figure 8: 3D Figure Models

Usually 3D jointed figures are constructed as object hierarchies. In a hierarchical model, the most stable part of a character--usually the torso-- is designated as the root node of a tree. The figure's arms and legs form descending branches of the tree. An arm, for example, forms at least a three level hierarchy below the torso, consisting of the shoulder, elbow, and wrist. Transformations applied to one level of the hierarchy automatically affect lower levels. Thus, rotating an arm at the shoulder automatically causes the forearm and hand to swing along with the upper arm. The hereditary nature of such transformations is generally considered to be the greatest advantage to tree structures. If the hand didn't move with the rest of the arm, creating any kind of animation would be tremendously difficult. Ninety-nine percent of an animator's time would be spent keeping the character's limbs attached to each other.



However, hierarchies can be extremely problematic--often for the very same reasons they are useful. With a typically constructed 3D figure, bending the legs upward from the hips is very simple maneuver. It requires changing a single rotation variable. However, getting that same figure to bend *over* at the hips, keeping its legs rigid, is usually much more difficult. These two operations, which seem almost identical, are very different with respect to the figure's fixed tree structure. The first operation--bending a leg--requires the transformation of one branch of the tree. The second--bending over--requires that the *whole tree* revolve around a single branch. Techniques for rerouting the structure of a tree hierarchy are fairly common, but they have yet to be incorporated into a conventional animation system.

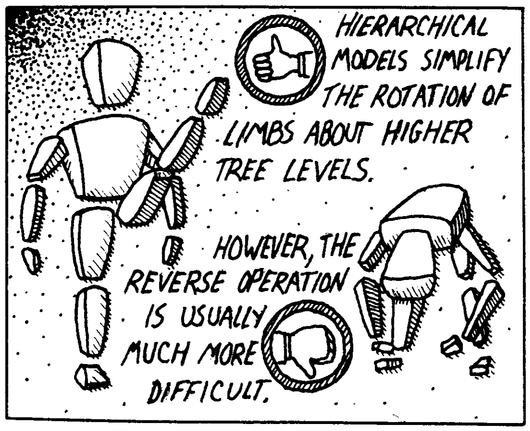
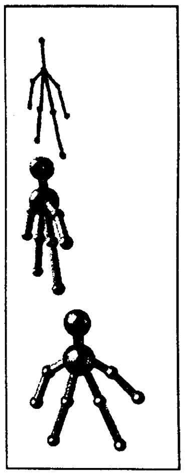

HIERARCHICAL MODELS SIMPLIFY THE ROTATION OF LIMBS ABOUT HIGHER TREE LEVELS.
HOWEVER, THE REVERSE OPERATION IS USUALLY MUCH MORE DIFFICULT.

Figure 9: The Problem with Standard Models

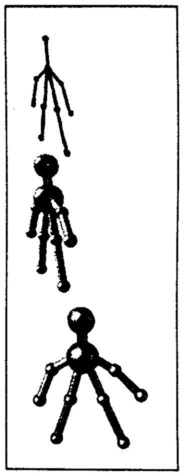


The two figure manipulations described above seem quite similar--they should be similar operations on an animation system. The simple skeletal representation I have chosen for my 3D figures provides that flexibility, although perhaps at the expense of structural heredity. The 3D skeleton is not hierarchical--it's merely a list of points and segments. The concept of "changing the shoulder angle" is not relevant to such a representation--joint angles are determined by the relative positions of limb endpoints. Also, changing the position of a shoulder does not guarantee that the elbow and hand will move along with it. While this loss of transformation heredity may seem unfortunate, in the context of character animation it is not so important.

This unusual figure representation has been chosen because it is more applicable to the requirements of character animation. Traditional animators do not think about shoulder or elbow angles; they just draw what looks right. This system must adapt that "what you see is what you want" mentality. Perhaps if a drawing does "look right," then it can be interpreted by the system as an actual 2D projection of the character. That is the premise of this thesis. By interactively aligning the true projection of a 3D skeleton with a hand-drawn estimation, the animator can easily create frame-by-frame motion sequences.

3.2 The Interactive Projection

The above description suggests that the animator's hand-drawn frames are really nothing more than guides for positioning the character skeleton--and that's exactly what they are. In this system, hand-drawn figures are not really "converted" into 3D models; there is no image processing or shape recognition performed. Hand-drawn animation merely provides a flexible, extremely simple means of "roughing out" motion. The animator could easily use digitized film or video as an alternate motion source. Character animation is used in this system because it can be used to create motion "from scratch." This system is selfcontained. Also, following the basic principles of traditional animation insures that the motion will look reasonably three-dimensional. The 2D/3D "conversion," if it can be called that, occurs as the user interactively positions the character skeleton over the drawings. The process is really a form of 3D rotoscoping, only without the need for two camera views.



This system allows a 3D skeleton to be positioned interactively onscreen as though it were a 2D rubberband stick-figure. Instead of changing nine different joint angles to position a figure's hand, the animator can just grab the hand with the cursor and drag it to the right place. For every frame in an animation sequence, the animator must position the skeleton's limbs over the corresponding drawn lines. Although this operation is essentially 2D, the skeleton is actually being positioned in 3D. How is the system able to manipulate a 3D figure as though it were flat? It uses elementary trigonometry combined with knowledge of the figure's fixed 3D structure.

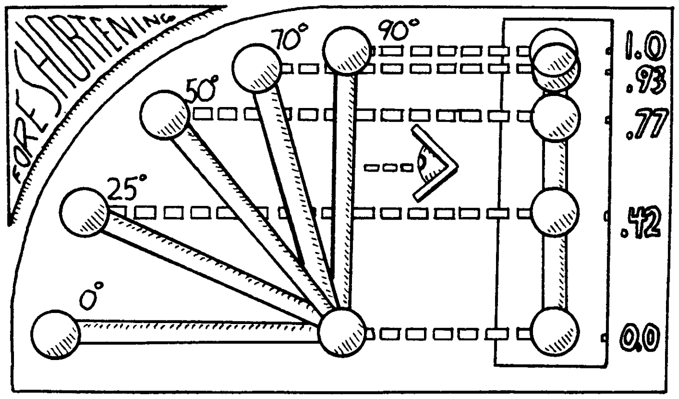
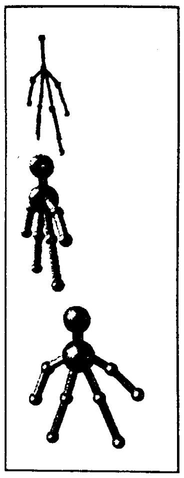


Figure 10: Foreshortening

3.3 The Reverse Projection Algorithm

The heart of this system is a routine that performs a reverse geometric projection. Given a set of connected 2D points--a stick-figure--the system reconstructs the 3D skeleton. Obviously a 2D projection alone does not contain enough information to reconstruct the 3D model from which it was made, especially if the "projection" is only a hand-drawn estimation. In fact, any one 2D stick-figure maps backward into an infinite number of 3D models, because the stick-figure has no depth information. The main problem, then, is recovery of the correct depth values for these 2D points.



How does the system recover depth information that is not present in the animator's hand-drawn frames? By comparing those frames to the known 3D model and its *true* projection. Wherever the animator has drawn a limb shorter than its actual length, the system assumes that the limb is *foreshortened*--that is, tilted toward or away from the user instead of lying directly within the screen plane. Animators always incorporate foreshortening in their drawings--it's an attribute that gives their characters solidity. This system relies on that user knowledge in order to recover 3D angle information from flat thumbnail sketches. In fact, the system can determine the exact angle of rotation for a limb simply by comparing its drawn length with its true length. The two measurements are used as the scalar values in a dot product:

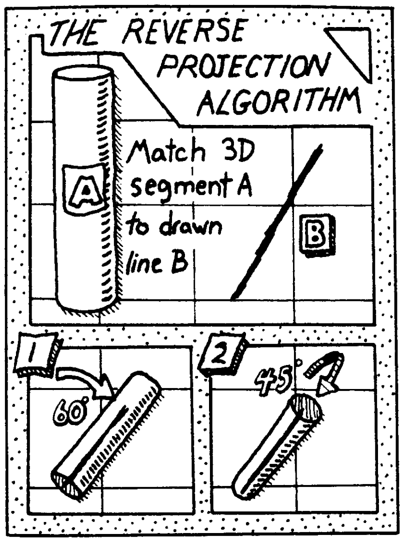
$$\alpha = \cos^{-1}\left(\frac{L_{drawn}}{L_{measured}}\right) \qquad \text{Equation 1}$$

where L_{drawn} is the length of the hand-drawn limb, $L_{measured}$ is the limb's true length, and α the angle at which the limb is tilted toward or away from the viewer. In-screen limb rotations--i.e., rotations occurring within the viewing plane--are determined by measuring the planar angle between the hand drawn limb and the line formed by the true limb's 2D projection onto the camera plane. The equation for measuring the angle between two 2D lines is:

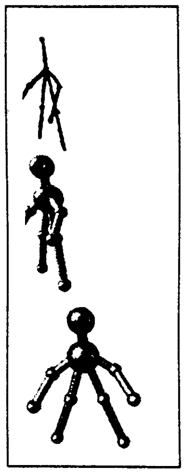
$$\alpha = \cos^{-1}(a_1a_2 + b_1b_2) \qquad \text{Equation 2}$$

where the normalized line equations are $a_1x + b_1y + c_1 = 0$ and $a_2x + b_2y + c_2 = 0$.

Figure 11 illustrates the use of this algorithm. Segment A is a 3D cylinder, defined by the user. Segment B is a hand-drawn line indicating the desired rotation of the bar. Using Equation 2, the bar is aligned within the screen plane

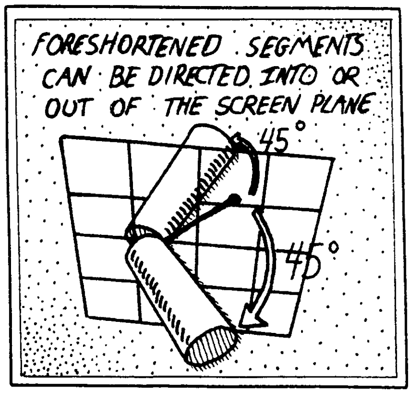

THE REVERSE PROJECTION ALGORITHM
Match 3D segment A to drawn line B
A
B
1
60°
2
45°
Figure 11: Reverse Projection

by rotating it 60 degrees around the Z axis. Notice that the drawn line is substantially shorter than the 3D bar, indicating foreshortening. Using Equation 1, the system determines that the bar is rotated 45 degrees either toward or away from the viewer. Figure 12 shows the two possible final positions of the bar.



Obviously there are major holes in this argument, if the above algorithm is to be used to convert entire jointed figures. For example, in order for the above equation to work, one of the segment endpoints must already be fixed in space. Also, deciding whether the limb is tilted into the screen or out of it can be a major problem. These problems are substantial--it may not yet be clear how an entire 3D figure can be converted. Nevertheless, this simple algorithm--determining the rotation and 3D position of a limb through estimation of its foreshortening--is the primary basis for my 2D to 3D conversion. The rest of this chapter deals with the questions just raised and others involving the conversion of an entire figure.

A requirement for using the above algorithm is that one of the limb's endpoints be fixed--that is, one joint's position must already be known in order to determine the two possible positions of the other joint. How can this algorithm possibly be of any use for posing a figure with ten to twenty joints, all of their positions unknown? If all of the figure's joint positions are really unknown, then it's true--this algorithm can't be used. However, if even a single joint *is* fixed in space, or just constrained to moving within a defined plane, this algorithm can be used


FORESHORTENED SEGMENTS CAN BE DIRECTED INTO OR OUT OF THE SCREEN PLANE
45°
45°
Figure 12: Ambiguous Orientation

recursively to convert an entire jointed figure. A visual example may help to explain the process more easily than text alone.

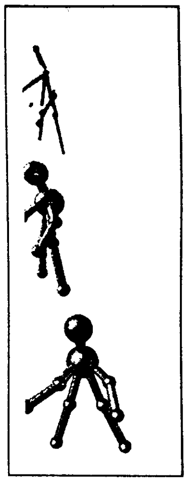


Figure 13 shows a series of linked segments, all of the same length. These segments have been created in 3D by the user; therefore their joint positions are known in x, y, and z. Of course, the illustration is only a 2D projection of those segments; the view has been selected by the user as one appropriate for animation of the object. Now, suppose the system is told to position those segments in 3D so that they match the second drawing in Figure 13. Node 1 is in the same 2D position in both drawings; it is assumed to be fixed in the third dimension as well. Using Equation 1, the system can determine that Segment A is tilted 45 degrees into or out of the screen plane. For now just assume that all foreshortened segments tilt inward. The x,y,z position of Node 2 is therefore established. With Node 2's position known, the same calculation can be applied to Segment B; it is found to be tilted inward 10 degrees. In addition, equation 2 determines that Segment B is rotated 60 degrees within the screen plane itself. Node 3 is now fixed. Following the same algorithm the system is able to convert the remaining segment.

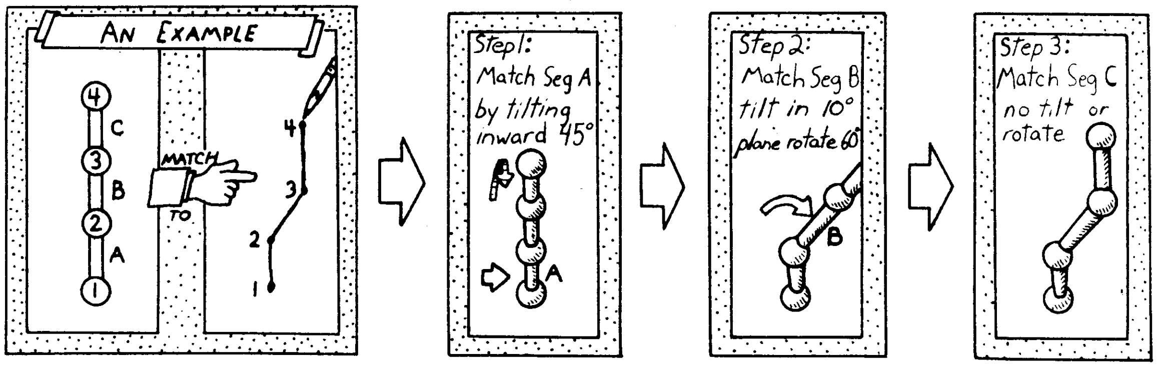
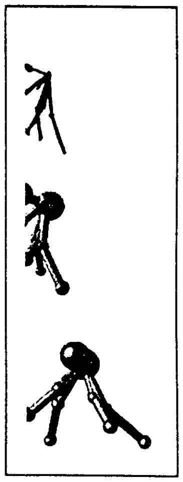

AN EXAMPLE
4
C
3
B
2
A
1
MATCH
TO
4
3
2
1
Step 1: Match Seg A by tilting inward 45°
A
Step 2: Match Seg B tilt in 10° plane rotate 60
B
Step 3: Match Seg C no tilt or rotate

Figure 13: An Example of the Reverse Projection Algorithm

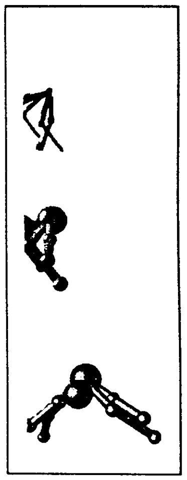
The above example assumes, for the sake of simplicity, that foreshortened limbs tilt toward the viewer. Obviously foreshortened limbs are just as likely to tilt away--how is the system supposed to decide which direction is correct? One flat drawing really doesn't provide enough information--however, by looking at the previous frames in the sequence, it is easy to make an educated guess about the limb's orientation. If a limb has been pointed in the same direction for multiple frames, then it is likely to remain that way. Therefore, each time the animator advances to a new frame, the 3D figure's limb orientations are copied from the old frame. On the screen limb orientations are differentiated by color; segments pointing toward the viewer are brighter than those pointing away. If a limb is pointing the wrong way, the animator can reverse it by pressing a key. Thus, the problem of foreshortened limb orientation has been solved by a combination of automatic frame inheritance and manual user control.



The foreshortening algorithm described above can be used recursively to handle most 2D/3D conversions. However, there are several exceptions which require special treatment. They are examined in the following examples.

What happens if the hand-drawn line is actually *longer* than the original 3D limb? Excluding the effects of visual perspective, there is no way to rotate a 3D segment so that its 2D projection appears longer than the segment itself. In this case the system assumes the longest possible limb projection, which is directly parallel to the viewing plane. Remember that when actually operating this system, the animator must interactively position the figure's 2D projection. Each skeletal limb is positioned by hand over the corresponding hand-drawn limbs. In this case, when the drawn limb is impossibly long, the skeletal node will simply stop moving when the limb has reached its maximum length.

Sometimes an animated figure does not have any fixed points. The limbs of a running man, for example, are continuously in motion. As his feet hit the ground, his whole body rocks back and forth, leaving no "anchor" points for the conversion algorithm described above. In such cases the animator is required to restrict the movement of at least one joint on the character, so that the system will have a reliable 3D starting point. Restricting a joint's movement does not mean that the point must be immobile. It means that the system must be able to determine the point's true 3D position, given only a 2D screen coordinate. A logical way to implement this restriction is to "lock" the point to a specific 3D plane. For example, the running character in Figure 3 can be constructed with its torso locked to the XY plane. The two torso endpoints are free to move parallel to the viewing plane, but their Z values are fixed. With the 3D positions of those two points established, the system can work outward to convert the rest of the figure using the foreshortening algorithm.



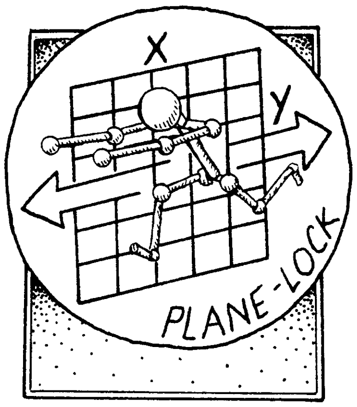
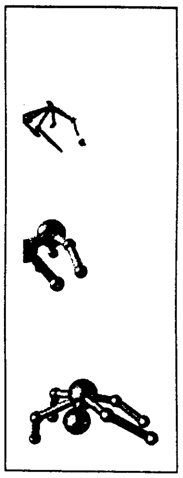


Figure 14: Plane-Lock

The ability to lock a point to a plane can also reduce errors due to imperfect drawing. Variance in the length of hand-drawn segments can cause a figure's limbs to jitter back and forth noticeably--the foreshortening algorithm incorrectly converts those length differences into angle changes. However, if the limb has been locked to movement within a specific plane, those changes will be ignored. Plane-anchored points are the first ones converted to 3D; therefore they take precedence over the normal, free-floating points. By locking the most important, stable parts of the character, the animator can minimize the errors caused by imprecise drawing.

Care must be taken not to overuse the plane-locking feature. If a skeleton has too many restrictions on its movement, it may be impossible to satisfy all the constraints. This often happens when the user plane-locks both endpoints of a single segment. Figure 15-1 shows a segment locked between two planes. Suppose the animator wants to align that segment with the hand-drawn line in Figure 15-2. The 3D segment formed by intersecting point A with plane A' and point B with plane B′ is much longer than the limb's fixed length. The system is caught between matching the limb to its corresponding hand-drawn line, placing the endpoints on their respective planes, and keeping the limb length fixed. How does the system resolve the opposition of these constraints? In this implementation, limb length is the only absolute constraint; the system will not position a joint so that its attached limbs are distorted. With that rule in mind, the system does its best to satisfy the other constraints. In the case of a segment with two plane-locked points, the system can probably lock one of the points, getting as close to the other plane as the fixed limb length will allow. Figure 15-3 shows that solution.



3.4 Attaching the Costume Objects

Once a skeleton's 3D pose has been established, the system can similarly position the figure's object costume. This operation breaks down into one of simple 3D object transformation. When the user first constructs a 3D figure, polygonal solids may be placed over the lines and nodes of the figure's skeleton. Each time an object has been correctly positioned, it can be symbolically "attached" to the corresponding skeleton limb. Thereafter, when the skeleton limb changes position or rotation, the object or objects attached to that limb will move also. The user is

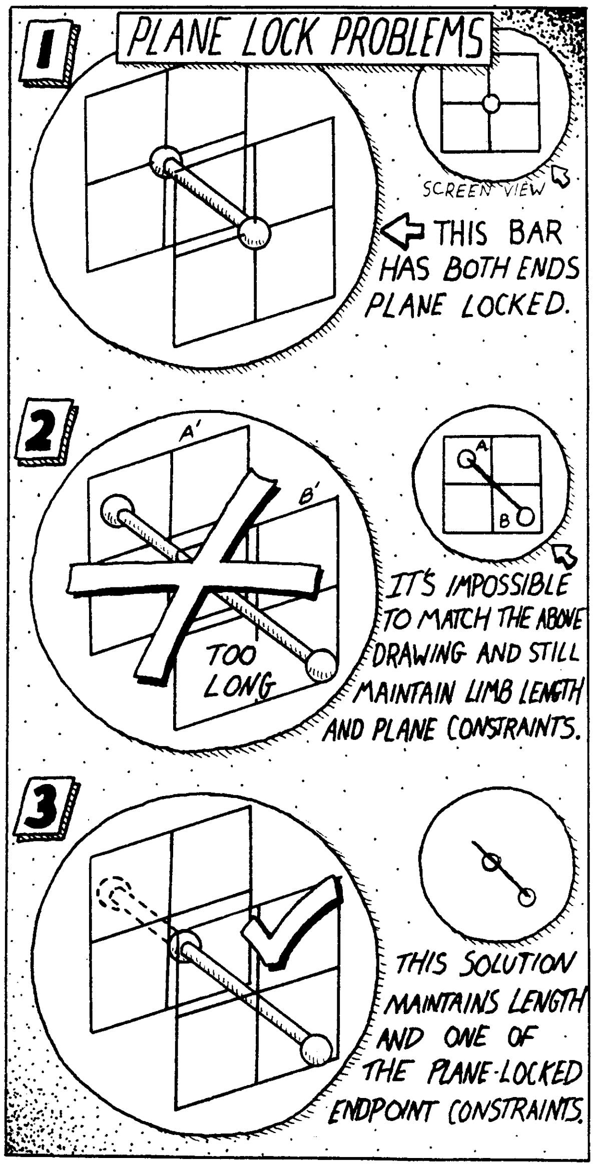
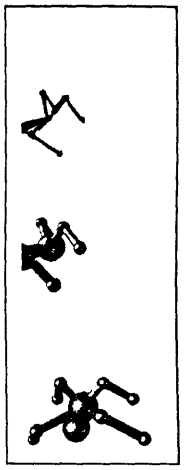

PLANE LOCK PROBLEMS
1
SCREEN VIEW
THIS BAR HAS BOTH ENDS PLANE LOCKED.
2
A′
B′
A
B
TOO LONG
IT'S IMPOSSIBLE TO MATCH THE ABOVE DRAWING AND STILL MAINTAIN LIMB LENGTH AND PLANE CONSTRAINTS.
3
THIS SOLUTION MAINTAINS LENGTH AND ONE OF THE PLANE-LOCKED ENDPOINT CONSTRAINTS.

Figure 15: Plane-Lock Problems

essentially posing the entire character by dragging points on the skeleton.



The algorithm used to effect these object transformations is very simple. By measuring how much a limb has changed from its starting position, the system knows how to transform the limb's attached objects. The changes are computed as standard 4x4 matrices, which can be applied to those objects. For each new frame in an animation, the system computes the transformation matrices for all of the limbs on a figure. Those matrices are then used to pose the costume of objects.

The transformation matrix for a single limb is computed as follows. The limb's initial and current position are represented as 3D vectors. The 2D/3D conversion algorithm insures that they have the same length; they differ only in XYZ rotation and position. Initial 3D rotation and position values are assumed to be zero. In the first frame, attached objects are already in the right place, so no action is needed to position them. The problem is to determine the XYZ translation and rotation for the current frame. The following equations are used to recover those unknowns:

$$\theta_{initial} = \tan^{-1}(a_1) \qquad \text{Equation 3}$$

$$\theta_{current} = \tan^{-1}(b_1) \qquad \text{Equation 4}$$

$$\theta_{final} = \theta_{current} - \theta_{initial} \qquad \text{Equation 5}$$

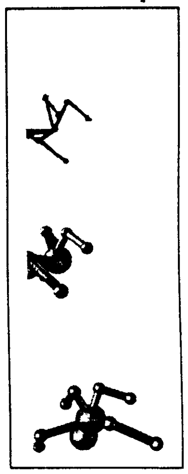
$$\phi_{initial} = \sin^{-1}(a_2) \qquad \text{Equation 6}$$

$$\phi_{current} = \sin^{-1}(b_2) \qquad \text{Equation 7}$$

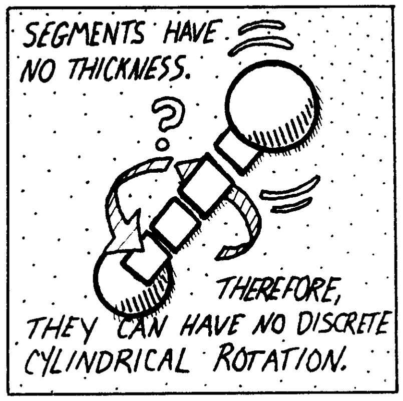
$$\phi_{final} = \phi_{current} - \phi_{initial} \qquad \text{Equation 8}$$

$$\text{translation}_{x,y,z} = \text{current}_{x,y,z} - \text{initial}_{x,y,z} \qquad \text{Equation 9}$$

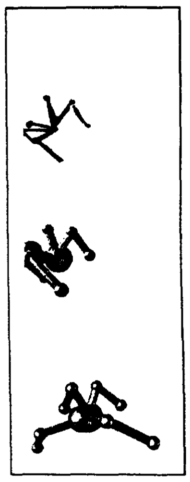
where **A** = a_1**i**+ a_2**j**+ a_3**k** and **B** = b_1**i**+ b_2**j**+ b_3**k** are the limb vectors for the initial and current frames. The variables $\mathbf{current}_{x,y,z}$ and $\mathbf{initial}_{x,y,z}$ are the tail points for those vectors. θ_{final} and ϕ_{final} are the spherical rotation angles that describe the difference in orientation between the initial and current limb frames. These equations correctly position the object relative to the current limb.



The above equations leave unspecified the transformed object's cylindrical rotation--that is, the rotation of the object about the axis formed by the limb. The system really has no way of knowing or computing that value--it's a problem inherent in the simple skeleton representation. The skeleton's limbs are composed of line segments, which by definition have no cylindrical rotation. For many costume objects this is not necessarily a problem--often a character's arms and legs are simple cylinders, which look the same at any cylindrical rotation. However, attaching asymmetrical objects to a 3D figure's limbs may produce unexpected changes in the object's rotation. If this happens, the animator can control the object's rotation manually using keyframes. This solution is admittedly far from perfect; future work may concentrate on finding a way for the animator to indicate cylindrical rotation during the drawing or interactive projection phase.


SEGMENTS HAVE NO THICKNESS.
THEREFORE, THEY CAN HAVE NO DISCRETE CYLINDRICAL ROTATION.
Figure 16: Cylindrical Rotation

Using the techniques described above, this system allows animators to manipulate 3D figures as if they were 2D "rubberband" skeletons. The algorithm automatically computes foreshortening when a rubberband line is too short, and it prevents the animator from positioning rubberband lines that are too long. However, even with built-in constraints on the figure's movement and knowledge of the figure's fixed structure, the system is likely to find several "correct" 3D interpretations of any single 2D projection. Sometimes a figure looks perfect from the initial angle but is unrecognizable from another. Also, hand-drawn sequences may appear realistic even when the drawn frames vary wildly in scale and limb length. Such variances can destroy the system's chances of correctly interpreting the drawings. For this reason the technique described above is very much an interactive one. The user must be able to recognize and fix mistakes made by the automatic conversion algorithm. Of several possible solutions to a frame conversion, the user must help the computer identify the correct solution.



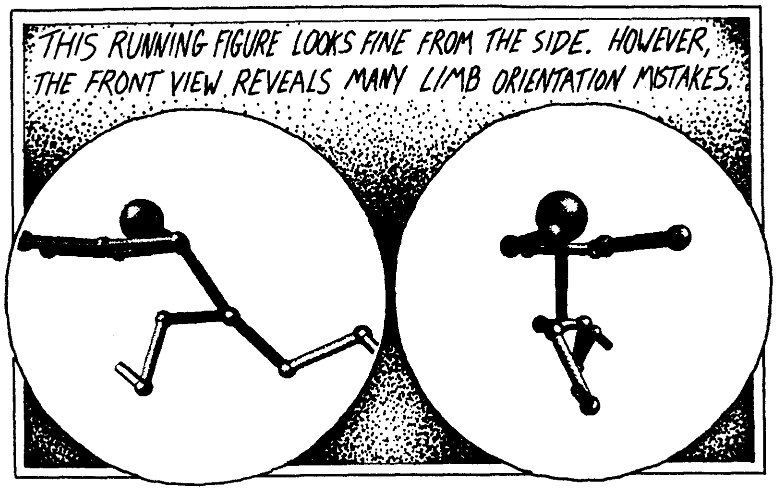
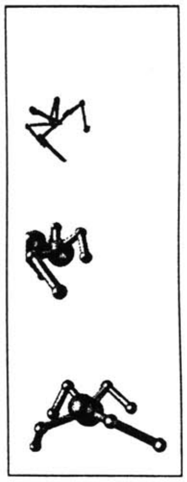

THIS RUNNING FIGURE LOOKS FINE FROM THE SIDE. HOWEVER, THE FRONT VIEW REVEALS MANY LIMB ORIENTATION MISTAKES.

Figure 17: Orientation Problems

Luckily, animation is never an automatic process. The editing decisions necessitated by the above problems are not too much to ask of an animator who is used to drawing hundreds of frames per animation sequence.

4. Program Design and Operation



The results of this research are a set of animation tools which have been installed within the author's animation program *new*. *New* is a keyframe-based animation system, written in C for the Hewlett Packard series 9000 computers. The system provides for the animation of both 2D and 3D object types, including flat polygons, bitmaps, typography, hand-drawn cels, 3D polygonal solids, light sources, and cameras. It is intended for use by non-programmers and has a user-friendly interface design, requiring very little keyboard input. The program uses the author's *BadWindows* windowing system, which allows direct access to the Hewlett-Packard's high level graphics functions. The system's integration of 2D and 3D animation, as well as its existing tools for hand-drawn animation, make it an excellent base platform for this research. This chapter begins with a description of the 3D skeleton object and its associated tools for figure creation. After that, the system's cel objects are introduced, with an explanation of how they can be used to create thumbnail flipbooks. The next section describes how these two object types are used together to produce 3D animation. The last part of this chapter describes how these tools have been incorporated into a keyframing environment and how keyframing may be used for detail animation.

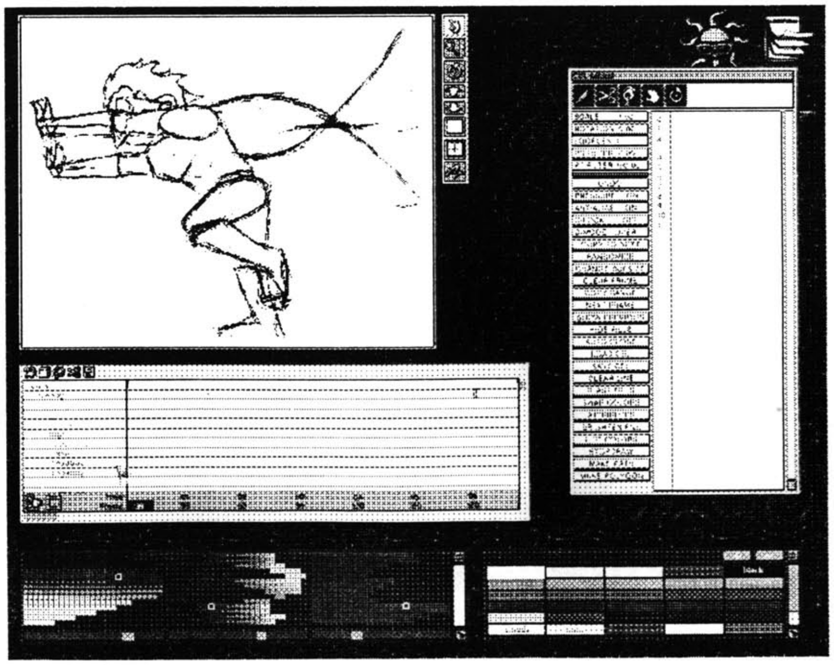
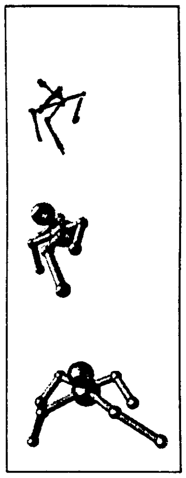


Figure 18: ***New*** **Interface**

4.1 3D Skeleton Construction



The 3D skeleton structures introduced in Chapter 3 are implemented within the animation system as an object type. In order to create a 3D figure for animation, the user selects "skeleton object" from *new*'s object menu. On the screen appears a 3D viewport, with a large vertical grid at the origin. Using the Hewlett-Packard's knob box input device, the user can position and rotate the grid in 3D. In addition, the grid's dimensions--width, height, and resolution--can be set via the skeleton object's onscreen control menu. This grid is the construction tool used to specify points on the character's skeleton. By positioning the cursor at one grid point and dragging it to another, the user can create 3D segments. A point gravity function allows new points to be connected easily to already existing skeleton nodes; as the segment rubberband line passes near a node point, it will lock on to that point until the cursor has moved further away.

Once a skeleton node has been created, it can be repositioned using the *move* mode. In move mode the cursor appears as a tiny outstretched hand. Clicking on a skeleton node and moving the cursor causes the node to be repositioned at the intersection of the cursor and the construction grid. Points can be cut from the skeleton using *edit* mode. In edit mode the cursor becomes a tiny pair of scissors, which can be positioned over any part of the skeleton. Clicking on a segment or node causes it to be deleted. Points and segments can also be destroyed from the *skeleton control window*.

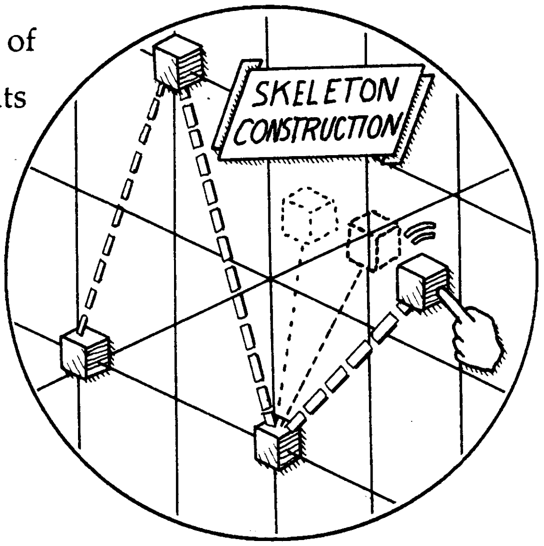
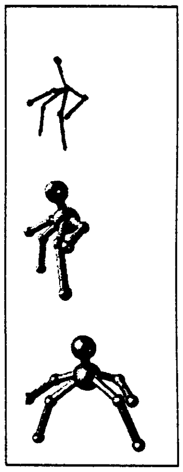


Figure 19: Skeleton Construction

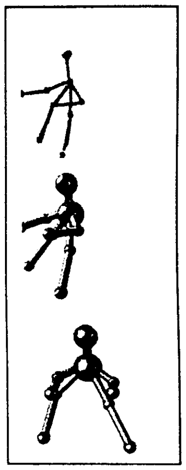
The *skeleton control menu* has a numbered listing of all the skeleton's points and segments. By clicking on the listing for a particular point or segment, the user can see a menu of operations which can be applied to it. Plane-locking is one of the operations which may be selected from this menu. Choosing Plane-lock for a particular point restricts it to the plane currently occupied by the construction grid. *Segment Hookup* is another menu option, applicable to segments. Segment Hookup allows the user to select another 3D object and symbolically "attach" it to the chosen segment. Once attached, that 3D object will maintain its position relative to the segment, translating and rotating with the segment as it moves. An entire costume of objects may be built for a character, by positioning one or more objects over each skeleton segment and symbolically linking them together.



Using the tools described above, the animator may construct a 3D skeleton with any number of points and segments. The finished model can then be viewed from any 3D angle. Sliding the mouse back and forth pivots the camera viewpoint around the model. In this way the animator may find a suitable angle for drawing the thumbnail flipbook.

4.2 Flipbook Creation with Cel Objects

The program *new* contains an extensive set of tools for creating traditional frame-by-frame character animation. The user interface for these tools bears a very close resemblance to the traditional animator's pegboard light-table. The current drawing is displayed on-screen, with previous and following drawings barely visible in a lighter color. This "onion-skin" effect helps the animator coordinate character movements from one frame to the next; it also makes inbetweening very simple. The animator's input device is a Wacom cordless pressure sensitive graphics tablet; in conjunction with custom antialiased line drawing software, it is an adequate substitute for a pencil or brush. Pressure-sensitivity in the stylus is mapped to varying line width, giving lightly sketched lines the appearance of a hard-lead pencil. On the other hand, varying the pressure over the length of a stroke will result in a pen-and-ink or paintbrush effect. Drawings are recorded and stored in the system as a list of individual pen strokes. Each time the screen is refreshed, the HP's line-drawing hardware redraws every line in the image--even for complex characters this operation takes only a fraction of a second, usually less time than it would take to redraw a bitmap. This stroke-based image representation is far more memory efficient than storing each drawing as a raster bitmap. The animator can work on sequences with hundreds of drawings without approaching the machine's memory limit.



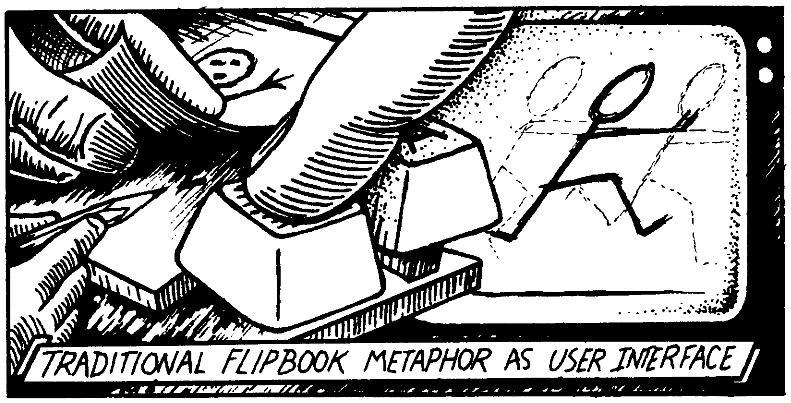
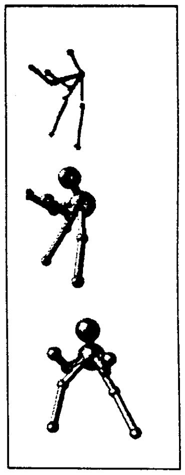


Figure 20: Skctch Layers in *New*

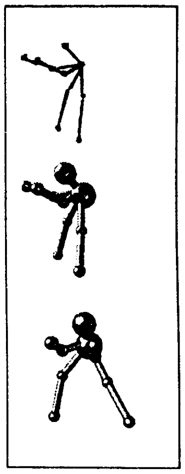
This system adopts the traditional process of developing a motion sequence, that of quickly sketching and flipping between drawings in a scene. On the computer, this process has been preserved and even improved. An animator using *new* is able to work quickly, sketching with one hand and flipping "pages" with the other by tapping keys on the keyboard. This method is preferable to pencil and paper, because the animator can hold down the keys and scan continuously through an entire scene. With paper it is only possible to flip back and forth between four or five drawings at a time.

Cel layer objects in *new* can be used to develop finished, painted character animation. However, for the purposes of this thesis they are only needed as thumbnail sketch flipbook tools. Using a cel setup like the one described above, the animator can quickly rough out the motion of a stick-figure. *New* is capable of simultaneously displaying 2D and 3D objects; therefore, any important 3D information, such as background scenery or the initial 3D skeleton, may be displayed while the animator is drawing. This feature is especially useful for establishing a camera position prior to drawing. The 2D/3D conversion algorithm needs a 3D camera specification from which to work backward; choosing one beforehand is better than trying to pick a 3D camera setup that will match a set of existing drawings. By creating a skeletal figure and viewing it from several angles, the animator can decide which is best suited for the series of drawings that will follow.



A finished set of flipbook drawings should correspond pretty closely to what the animator wants the final motion to look like. However, the animator does not have to draw every single frame in a sequence; only the "keyframes" are absolutely necessary. The intervening frames can be automatically computed by interpolating along splines passed through the model's joint nodes once the 2D/3D conversion has been completed. Care should be taken not to overuse the spline-inbetweening, or else the animation will lose its spontaneity.

Creating a set of flipbook drawings that can be used as reliable 2D projections is not easy. Even very experienced animators find it hard to correctly account for the perspective effects of a true 3D camera. Many hand-drawn attempts at 3D motion may seem realistic by themselves but look horribly distorted when compared to the real thing. Therefore, the animator must plan each sequence carefully, simplifying the character design as much as possible and picking a camera angle that will minimize perspective distortions. The ideal camera angle is one that offers a clear, unobstructed view of the character's whole body throughout its entire range of motion. If the character's whole body is in motion, the animator must remember that some point on the figure's body will have to be locked to a plane in order for the 2D/3D conversion to occur. If possible, that point should be the most important one on the character, so that it will have the best chance of being placed correctly. If the most important part of the character is not in the right position, others are likely to be even farther off. Keeping these factors in mind, the animator should be able to create an flipbook sequence that will translate well into 3D.



4.3 Translating 2D Sketches into 3D Motion

Hand-drawn stick-figure information is transferred to the animator's 3D skeleton via the interactive projection method introduced in chapter three. Although the on-screen image is a projection of a 3D skeleton, the animator is able to manipulate it as though it were a set of linked 2D rubberband lines. The animator steps through the sequence of flipbook frames, in each one positioning the interactive 3D figure to match the drawn figure. The system remembers these individual poses by creating a new copy of the 3D skeleton structure for each frame. Thus, for every frame in the animator's flipbook there exists a matching 3D skeletal model. The idea is similar to claymation, in which a single character may actually consist of dozens of nearly identical clay models, each with a slightly different pose or expression. However, a computerized system employing this idea has the distinct advantage that new models may be created instantly by copying the previous pose. In addition, the computer is able automatically calculate any number of inbetweens for every user-defined frame. The timing of a scene may be altered smoothly by inbetweening various segments of the motion.

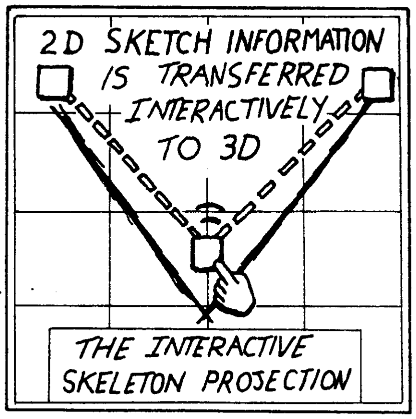
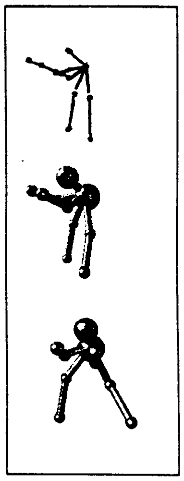


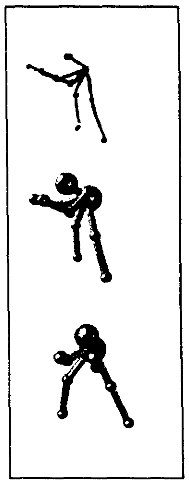
Figure 21: Interactive Projection

As the animator positions the skeleton nodes for each frame, the system is continuously running the 2D/3D conversion process. Each time a new cursor location is read from the graphics tablet, the conversion algorithm updates the selected joint position and then determines whether the position is a valid one. If the system cannot figure out any way to move the current joint to the new position, it will come as close as it can. This effect is noticeable in the system when the animator attempts to drag a limb farther than its maximum length. Once that limit has been reached, the limb will cease to follow the cursor; the animator will know to move back a little. Also, manipulating one node may affect several others on the skeleton--for example, dragging a knee node may cause the ankle and foot nodes to slide with it. These visual cues are a beneficial side effect of the interactive conversion; they can be helpful in determining how the program is interpreting the 2D projection.



Color is used in the interactive projection to indicate the direction of foreshortened lines. Limbs directed toward the viewer are displayed as red lines; those directed away from the viewer are purple. Since most foreshortened lines can be interpreted as pointing either toward or away, the animator is responsible for setting their directions. Usually this is not difficult, because limb orientations for new frames are automatically copied from the previous frame. However, the animator must always remain conscious of possible mistakes in limb direction. If a limb is pointing the wrong way, it can screw up the conversion of the whole figure. Determining a limb's 3D orientation can be a challenge for odd camera angles. In order to maintain a clear understanding of the figure's true 3D position, the animator may wish to have two 3D viewports on the screen at the same time. The two viewports are given radically different camera angles. Hence, one may used as a reference to double-check limb orientations. A limb which is ambiguous from one camera angle may stand out clearly from another.

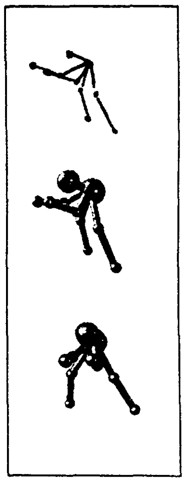
Positioning the 3D skeleton is essentially the last step in the animation process. Once all the frames in a flipbook sequence have been transferred to 3D, the animator just has to check for mistakes and smooth out noticeable aberrations in the movement. Sometimes an animation which looks excellent in skeleton form will not work when the character's full costume is displayed. The animator must remember to provide ample space for thickly modeled body parts; otherwise, the finished animation may be plagued by unsightly object intersections!



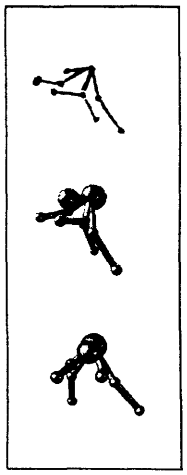
4.4 Keyframing and Detail Animation

All of the above tools are part of the animation program *new*, a keyframing system. Yet the animation process described so far has very little to do with keyframing. How do these two animation methods interact? Keyframing has a number of potential uses within the context of this research--the most valuable of these is for *detail animation*. Detail animation is any kind of motion that cannot be easily expressed as part of the simple character skeleton--for example, facial expression, finger movement, and cylindrical rotation around a skeleton limb axis. Objects can be attached to a 3D skeleton and still exhibit standard keyframed motion. For example, a pair of eyes may be programmed to blink and look around while attached to the "head" limb of a 3D figure. The hierarchical nature of *new*'s object world allows this parallel animation of objects to occur. A grouped object, by definition, follows both its own instructions and those of parent objects above it.

Keyframe animation also becomes important when one considers producing a finished animated short film. Jointed figures are usually not the only source of movement in an animation. Camera pans, scenery, and props are better animated with keyframing than with the technique proposed by this thesis. In fact, for this research to have any practical application at all, it must function within an environment that supports other methods of animation. *New* is ideal in this respect, because it supports so many different types of animation--2D and 3D keyframing, cel, and even procedural animation. The ultimate goal of this research has been to extend that list with a set of tools appropriate to the animation of complex jointed figures. Those tools, though limited in their application, enable certain types of 3D animation to be easily produced that would otherwise be very difficult.



5. Implementation

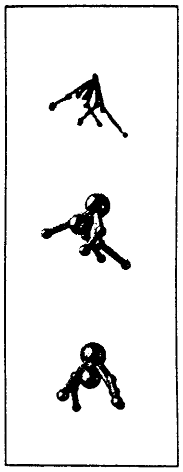


This chapter discusses the actual implementation of my 2D/3D conversion algorithm presented in Chapter 3. The procedure, called *InstantConversion*, poses a fixed-length 3D skeletal node structure based on its initial position and a single set of 2D projection points. *InstantConversion* is aptly named--it is executed every time the user makes a change to the 3D skeleton, yet the conversion runs quickly enough for the entire process to be interactive. The code is written in C for the Hewlett-Packard series 835 workstations. They each have a RISC-based processor, two 24-bit high resolution frame buffers, and an extensive 3D graphics library. Several of these library routines are used in the low-level coding of the 2D/3D conversion algorithm--therefore, porting this code to another machine would require that it have similar features.

5.1 *InstantConversion*

The conversion code is actually broken into two hierarchical routines, of which *InstantConversion* is the higher level. *InstantConversion* manages the segment-by-segment positioning of the 3D skeleton; it is the "director" function, deciding which segments have changed and in what order they should be converted. The function takes as arguments a 3D skeleton structure and a set of 2D points. The skeleton structure is the character's initial position; the 2D points represent the desired pose as seen from the current camera angle (see Figure 22). The function's job is to return a new skeleton structure, whose connectivity and limb lengths match the initial figure, but whose pose matches the 2D point set.

InstantConversion is performed in three steps. In the first, the target skeleton is created and "cleared"--its points and segments are all marked as unconverted. Then, a reference 2D projection is created from the supplied 3D skeleton, using the current 3D camera model. The result of this projection is a set of 2D points very much like the one supplied to the function. By measuring the differences between these two projections, the system can determine how to transform the limbs of the 3D figure. Obviously, 2D points that do not change at all are easy to convert; in such cases the system assumes that the 3D point is stationary. That point's X, Y, and Z values are copied directly from the initial skeleton to the target, and the new node is marked as converted.



In the second step of figure conversion, a recursive loop is used to find and transform segments that have only one of their endpoints locked down. Assuming there is at least one stationary point in the figure, this loop is guaranteed to find a valid segment on its first iteration. Conversion of that segment, by definition, will lock down its free endpoint. It follows that any other limbs attached to the endpoint will be eligible for conversion on the next trip through the loop. As each segment and node is converted, it is flagged. Only when every limb has been marked will the system exit the loop.

Suppose that none of the skeleton points are stationary--the loop described above will have no starting point. In this case plane-locking becomes important. Plane-

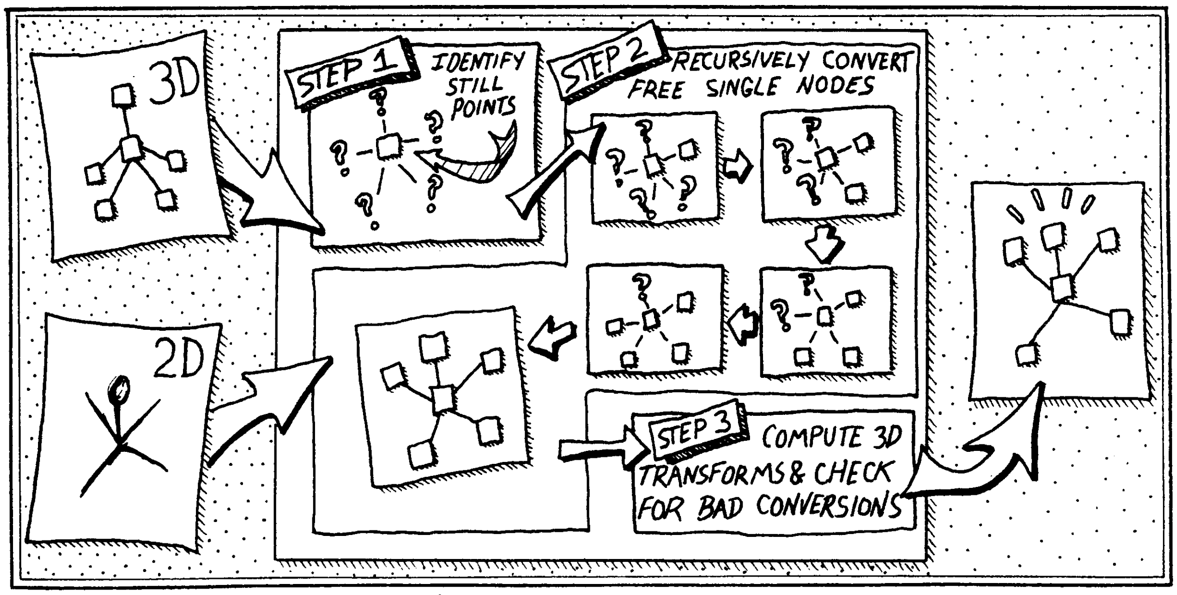
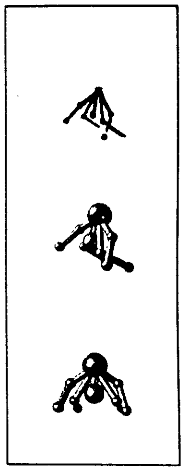

3D
2D
STEP 1
IDENTIFY STILL POINTS
STEP 2
RECURSIVELY CONVERT FREE SINGLE NODES
STEP 3
COMPUTE 3D TRANSFORMS & CHECK FOR BAD CONVERSIONS

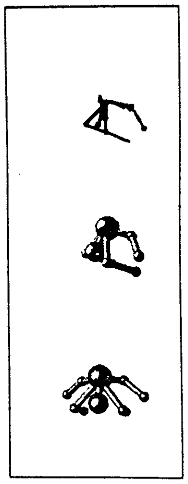
Figure 22: ***InstantConversion***

locked points can be converted at any time; the 2D projection coordinate combined with node's specified plane-equation is always sufficient to locate it in three-space. Any segment with a plane-locked endpoint will be automatically converted as soon as it is encountered in the loop. Plane-locked nodes thus have a higher priority than ordinary ones.



The third and final step in *InstantConversion* is a cleanup operation. It is possible for both endpoints of a segment to be locked down separately, without the segment itself being sent to the line conversion routine. This happens when a segment's two immediate neighbors get converted before the segment itself. In such cases both nodes get marked, but the segment itself does not. This situation usually results in the segment having the wrong length; since the endpoints are positioned independently of each other, there can be no attempt to maintain the fixed limb length. The "true" solution to this problem would require a retransformation of all the neighboring segments. The system would have to find a compromise that preserved limb lengths, while staying as close as possible to the given 2D projection. This problem rarely occurs, however, so a simpler, though technically incorrect, solution has been implemented for the present. The system assumes that the segment length, though invalid, is not really that far off. No attempt is made to change the length or point positions. Instead, the system computes the appropriate 3D transformation matrix for the segment's attached objects. The "broken" segment has the wrong length, but its 3D orientation is still valid. If the mistaken length is close to its correct value, the viewer is unlikely to see a problem. If larger errors do occur, the 3D skeleton may be too distorted to use anyway--the "correct" solution would require such a radical departure from the desired 2D projection as to make it useless. This "solution" relies on the assumption that the animator's drawings are reasonably accurate. If they are not, this technique is not going to work very well anyway.

5.2 *NewConvertLine*



The second procedure described in this chapter is the actual low-level segment conversion routine, *NewConvertLine*. This routine performs the conversion of a single line from 2D to 3D. Given the limb's initial 3D coordinates, a pair of 2D destination points, and one 3D endpoint, *NewConvertLine* determines the unknown 3D endpoint's position. The technique used to do this is introduced in Chapter 3. However, the actual 2D/3D conversion code differs somewhat from the algorithm explained previously. The algorithm presented in Chapter 3 is a simplified version--it does not take into account the distortion effects of a 3D perspective camera. Perspective is an important visual cue, especially when working with 3D computer graphics. Therefore it is essential that the tools developed in this thesis work with perspective cameras. The fundamental assumption of Chapter 3's conversion algorithm holds true for any camera model--a 2D line maps backward to at most two 3D lines, given the camera position and one of the 3D endpoints. However, for perspective cameras the equation used to determine the line's angle of rotation becomes much more complicated than a simple dot product. That angle is no longer simply a function of the hand-drawn line length. With a perspective camera, the onscreen length of a 3D line depends not only on its foreshortened angle but also its distance from the viewer. A segment tilted toward the viewer will appear longer than the same segment tilted the same angle away from the viewer--perspective adds length to the first example and subtracts length from the second.

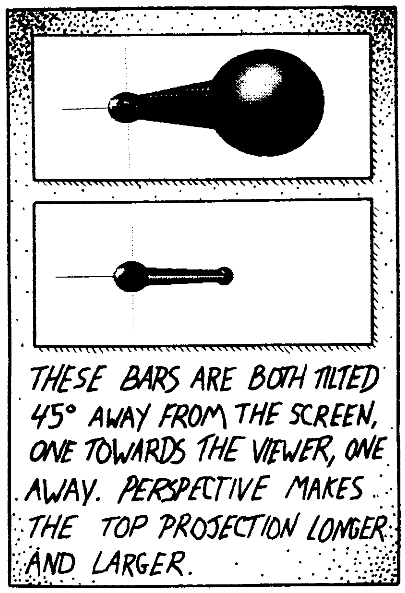
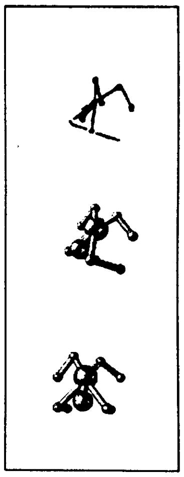

THESE BARS ARE BOTH TILTED 45° AWAY FROM THE SCREEN, ONE TOWARDS THE VIEWER, ONE AWAY. PERSPECTIVE MAKES THE TOP PROJECTION LONGER AND LARGER.

Figure 23: Perspective Effects

The addition of perspective cameras complicates the reverse projection problem. Fortunately, thanks to Hewlett-Packard's graphics library, that problem has a simple solution. The Hewlett-Packards have a 3D transformation engine that makes conversion between coordinate systems easy and efficient. For example, the *transform_point(WORLD_TO_VDC)* command automatically computes the 2D projection coordinates of any 3D world point. Even better, the *transform_point(VDC_TO_WORLD)* command will return a valid 3D point, given a 2D coordinate and a depth value relative to the screen. If (x,y) is the screen coordinate of the unknown limb endpoint, the 3D solution lies somewhere on the line passing through (x,y) perpendicular to the screen. Using calls to the *transform_point* function, it is possible to calculate the equation of that line. Two of those calls, with arguments (x,y,z_1) and (x,y,z_2), will return a pair of 3D points on the desired line. Any values can be assigned to z_1 and z_2, as long as they differ. Using the equation for a line through two points, it is trivial to locate the unknown limb endpoint. One simply has to find any points on the line that are distance d from the given endpoint, where d is the limb's length. The problem breaks down into a standard quadratic formula.



$$v_x = x_2 - x_1 \qquad v_y = y_2 - y_1 \qquad v_z = z_2 - z_1 \qquad \text{Equation 10}$$

$$x = x_1 + v_x t \qquad y = y_1 + v_y t \qquad z = z_1 + v_z t \qquad \text{Equation 11}$$

$$at^2 + bt + c = 0 \qquad \text{Equation 12}$$

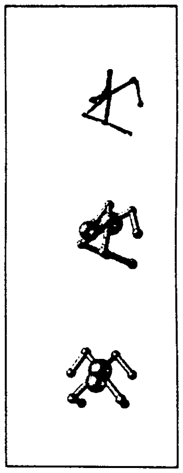
$$a = v_x^2 + v_y^2 + v_z^2 \qquad \text{Equation 13}$$

$$b = 2 \times (v_x x_1 + v_y y_1 + v_z z_1) \qquad \text{Equation 14}$$

$$c = x_1^2 + y_1^2 + z_1^2 - d^2 \qquad \text{Equation 15}$$

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad \text{Equation 16}$$

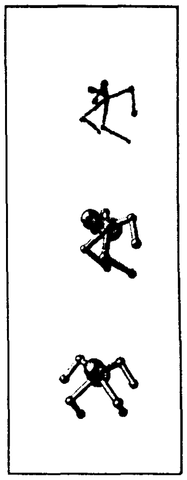
The value **t**, when put back into Equation 11, will give the correct x,y,z values for the limb endpoint. In these equations the 3D endpoint has been translated to the origin, to make calculation easier. The 3D line is translated by the same amount. The 3D points resulting from these equations must naturally be offset by the same amount, to account for the translation.



Notice that two solutions can exist, or a single solution, or none at all. These cases result from three specific situations that commonly arise while running the system. The existence of multiple solutions has already been discussed. There can be two answers, because the limb can be in two different positions and still match the drawing. If the segment is pointing *toward* the viewer, the correct solution is the 3D point closest to the screen plane; if the segment is pointing *away*, then the answer is the more distant point (assuming that the limb points in the direction of its "free" node). When only one solution exists, the limb lies directly within the screen plane; there is no foreshortening on the line. Finally, if Equation 16 has no solutions, the line is too far away. The system must compromise by again giving the limb no foreshortening and putting it within the screen plane.

NewConvertLine is called repeatedly as *InstantConversion* runs through its recursive loop. The two procedures are both relatively short, but together they accomplish the main task of this project--2D/3D figure conversion.

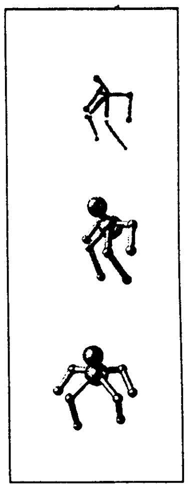
6. Results



This chapter discusses the relative merits and drawbacks of the tools presented in the preceding chapters, as compared to more conventional systems. In addition, an animation sequence produced with the system is presented and analyzed. Initial tests of the system have been largely successful. The proposed method of manipulating 3D figures seems well-suited to the existing work practices of traditional animators. However, the system is far from perfect. It relies too heavily on the animator's drawing ability, and conventional operations such as rotation and translation are not integrated very well with the interactive figure positioning tools. These problems were relatively unexpected--they became evident when using the system for real animation production. Although they do not prevent the use of this system in a production environment, these problems certainly point the way for future research.

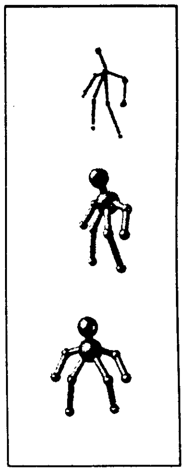
Probably the most successful elements of this system, in ease-of-use terms, are the preparatory animation tools for 3D skeleton construction and hand-drawn flipbook development. The construction-plane works very well as a means of building a 3D skeleton. The ability to position it numerically is especially useful for maintaining symmetry of the figure; in this respect it is quite similar to the construction plane approach of many computer-aided design systems. One could imagine using this tool in an architectural software package. Also, the construction plane has dual functionality--it is used to constrain point movement to specific 3D planes. Although users of the system have not encountered any problems with this tool, one person familiar with more sophisticated 3D rotation schemes suggested that quaternion rotations [10] could be used to position the plane. Quaternion rotations are supposedly more intuitive than simple roll, pitch, and yaw.

The hand-drawn animation tools used in this thesis have been employed in the production of traditional animation in the Visible Language Workshop for nearly two years; they are reliable and easy to use. The flipbook format is simple enough for an eight-year old to learn in a few minutes; yet the same tools can be used to produce finished, professional quality character animation. All of the 2D animation for the short film *Grinning Evil Death* [11] was produced with this system. The only real problem with these tools is that they do not provide enough computer assistance--there is no automatic inbetweening or cel cleanup. The animator is still required to draw every frame of an animation sequence. For the purposes of this thesis, however, the lack of automation is not a major problem. The animator is only drawing stick-figures, so inbetweening is easy and cleanup unnecessary. Also, once the sequence has been converted to 3D, automatic inbetweening *is* available.

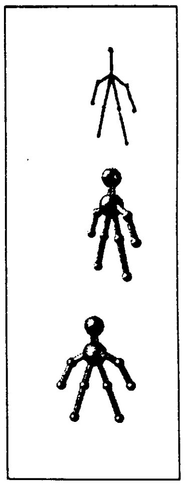


The only substantial problems with this system occur during the 2D/3D conversion stage. This isn't really surprising, given the novelty of the conversion technique and its reliance on user drawing skill. Test animations performed with the system reveal a tendency of the user to misjudge limb lengths in thumbnail sketches. As a result, the conversion routines often assume foreshortening that is not intended by the artist. A figure's limbs will swing unpredictably in and out of the picture plane. These errors are not usually evident from the initial viewpoint; they show up when the sequence is seen from a camera angle other than the drawn one. The animator should therefore draw his or her animation from the same angle that it will eventually be seen. Of course, that is not always possible, particularly when a moving camera is desired. In such cases the only real solution to the problem is trial-and-error editing of the motion. The animator should pick two representative camera angles and edit with multiple viewports onscreen. Thus the user can assure that any changes will look good from both eye points.

The other problem with the 2D/3D conversion is the lack of conventional figure positioning tools. Occasionally it is desirable to manipulate an arm or leg with simple 3D rotation. Or the animator may wish to slide the entire figure some distance along the Z axis. In such cases standard 3D transformations would be more appropriate than the thumbnail sketch metaphor presented here. Fortunately, the 3D skeleton format used by this system does not prohibit the use of conventional methods; the next step in this research will be the incorporation of standard 3D and translation capabilities with the interactive skeleton projection.



The problems with this system do not prevent its use as an effective animation tool. The system is especially useful for the full-body animation of complex jointed figures, since that type of motion is so difficult to produce otherwise. Extended Figure 1 (at the top right of the page) is an example of a jumping humanoid figure produced with this system. By slowly flipping the pages of this thesis from back to front, you will see the standing figure crouch, leap into the air, and land. The top image is the initial set of thumbnail sketches for this motion. The middle image shows the motion applied to a 3D figure constructed of spheres and cylinders, and the lower image is the same motion seen from a different camera angle. This example was selected because it would be difficult to duplicate on another system. During the crouching and landing stages of the sequence, the character's body moves relative to its feet--ie, the feet hold still, even though the legs are moving. As explained in chapter two, standard systems would prevent this action--their hierarchical models do not allow figure movement relative to end nodes. Extended Figure 1 is a full-body animation; at some point in the sequence every node on the figure moves from its initial position. In order to produce this motion, the character's torso was locked to the vertical plane traveling along its path of movement. In other words, the ball representing the character's body was allowed to move up and down along the jumping path, but not side to side. The 3D version of this sequence matches the hand-drawn flipbook very well, especially from the initial angle. The front view reveals the problem described above, the animator's tendency to misjudge limb length. Mistaken foreshortening causes the character's left leg to kick outward; the problem occurs because the sketched version of the leg has been drawn slightly shorter than normal. From the initial angle the problem is not really noticeable; it only becomes apparent when the scene is viewed from another angle.



7. Conclusion

The basic principles of character animation are founded on the knowledge of 3D motion. The technique presented in this thesis incorporates those principles into a 3D computer graphic animation system. The system allows hand-drawn stick-figure flipbooks to be used as guides for positioning the interactive projection of the user's 3D model. The result of this work is a simple, intuitive interface for the manipulation of 3D figures. In video tests the tools have proven successful for a variety of animation tasks, including complex full body motion. The animation process developed as part of this research is not meant to replace existing techniques; it is designed to be used in conjunction with standard methods to increase the tools available to the computer artist, and to expand the range of applications appropriate to computer animation.

This project has great potential for future research, particularly in the area of automatic sketch-recognition. The ideal system would be able to generate a 3D moving figure automatically, given only a set of rough flipbook drawings. That system is clearly a fantasy, at least for the immediate future. However, several of its appealing features are definitely grounded in reality. For instance, image-processing techniques, particularly energy minimization, could be used to aid the animator in positioning the nodes of the interactive 3D projection. A system employing such techniques would automatically place skeleton nodes by allowing them to "migrate" towards areas of the source drawing with maximum energy--ie, corners in the stick-figure. It is doubtful that the system would be 100 percent reliable, but even a small amount of computer assistance would be beneficial.

The emergence of superior graphics capability in personal computers is making computer animation a viable form of expression for a majority of their users. The inevitable explosion in the number of amateur films is likely to bring about a change in the standards by which they are judged. Flashy graphics will no longer impress--the visual and technical sophistication that distinguishes the very best films today will soon be commonplace. Style and content will take precedence over photorealism. New animation tools are needed, ones that encourage artist-computer interaction over automation. This project has resulted in one such set of software tools, which establish traditional character animation as a practical and efficient user interface for computer animators.

Bibliography

[1] Thomas, F. and O. Johnston. (1981). *Disney Animation: The Illusion of Life.* New York, Abbeville Press.

[2] Lasseter, J. (1987). *Luxo, Jr.*, computer animation produced by Pixar.

[4] Zeltzer, D. (August 1984). *Representation and Control of Three Dimensional Computer Animated Figures*, Ph.D. Thesis, Ohio State University.

[5] Girard and Maciejewski. (1985). *"Computational Modeling for Computer Animation of Legged Figures"*, Computer Graphics, Vol. 19, No. 3.

[6] Girard and Maciejewski. (1989). *Eurythmy,* computer animation produced at Ohio State University.

[7] Lasseter, J. (1987). *Tin Toy,* computer animation produced by Pixar.

[8] Witkin, A. and M. Kass. (August 1988). "Spacetime Constraints", *Computer Graphics.* 22(4): 159-168.

[9] Maxwell, D. R. (1983). "Graphical Marionette: A Modern-Day Pinocchio", Master's Thesis, Architecture Machine Group, MIT.

[10] Shoemake, K. (1985). "Animating Rotation with Quaternion Curves", Computer Graphics, Vol. 19, No. 3.

[11] Sabiston, B. and M. McKenna. (1990). *Grinning Evil Death,* computer animation produced at the Media Laboratory, MIT.