Applying Traditional Animation Techniques to Computer Animation

by

Eric J. Wessels

B.S. January 1993, George Mason University, Fairfax,
V

A Thesis submitted to

The Faculty of

The School of Engineering and Applied Science
of The George Washington University in partial
satisfaction of the requirements for the degree of Master of
Science

June 1998

Thesis directed by

Dr. James K. Hahn
Associate Professor of Engineering and Applied
S

Contents

Abstract

This thesis presents an approach to character animation that employs computer-based tools which build upon techniques used in traditional animation. Rotoscoping and spacing charts are techniques familiar to the animator and, thus, provide a comfortable working environment which he understands and which he knows has produced good results in cell-animation for decades. Rotoscoping generates the initial motion the animator wishes to express, and spacing charts allow the entire motion to be enhanced and to be expressed more fluidly from beginning to end. A computer system capturing these traditional animation techniques is presented which will reduce the labor intensive activities of the animator, enhance the final product and not burden the animator with unfriendly, or excessively technical tools to accomplish his task.

Chapter 1

Introduction

The creation of life-like character animation has been one of the most challenging tasks in computer animation. Humans are among the most important and interesting objects simulated using computer graphics, but they are also the most difficult to convincingly animate. Despite the difficulties in creating life-like character animation, traditional animators have used techniques for decades which have proven successful. While many approaches have been introduced in the computer animation field for the purpose of character animation, none allow the computer animator to work in a fashion closely resembling that of his traditional animation counterpart using the techniques that have proven successful for so many years. As a step in this direction, this paper describes an approach to character animation which allows the computer-based animator to use techniques closely analogous to those of his traditional animation counterpart. Furthermore, the tools presented in this thesis are designed to be intuitive enough that even traditional animation artists who are not savvy with current computer-based approaches can utilize this system.

Despite the variety of tools that have been introduced in computer animation for the purpose of character animation, few have been accepted by animators. The success of any new tool introduced to assist in character animation will depend upon whether the animator finds the tool helpful and less laborious to use than other tools that are currently available. This approach focuses upon enhancing procedures that have proven successful in producing expressive, believable animated characters by providing tools to the animator that are easy to use and reduce the amount of labor needed to accomplish the animator's tasks. Computer-based animation has had great success, nonetheless the tools would be of even greater benefit to the animator if they blended into the work background he recognizes and assisted in accomplishing his tasks in a less laborious and more intuitive manner than what is currently available. Thus, the motivation for the approach presented here is to benefit animators by creating computer-based tools that are designed to correlate to concepts in traditional animation that are known to produce good results and are well-understood. The approach focuses on providing motion control solutions which the animator can use in an intuitive manner to address various problems presented in animation. It attempts to present a technological solution that the animator can understand and a collection of tools for character animation that the animator will find easy to use because they are designed to correlate to concepts familiar to him. In the next section, existing methods for character animation are examined to help place into context the techniques developed for this thesis.

1.0 Background

Expressing human-like movement is one of the most difficult problems in computer animation. The specification of movements requires complex articulations involving up to 200 degrees of freedom. The difficulty is increased due to the fact that humans are sensitive observers of each other's motions and can easily detect erroneous movements. Even for simple models of the human figure, there are approximately 70 parameters that have to be specified for every frame of animation. As a result, much research has focused on ways to reduce the amount of specification necessary.

In this section, motion control techniques for articulated figure motion are examined. The section will begin with a description of the traditional animation techniques which are implemented in this system, namely rotoscoping and spacing charts. This is followed by a description of the evolution of computer-based techniques from low-level parametric keyframing to high-level procedural systems. Since spacing charts are used to create variations of motion, the reusability of motion in existing approaches is discussed. Finally, the drawbacks of current approaches are discussed and the proposed benefits of this approach are discussed thereafter.

In the production system pioneered by Walt Disney, keyframing is a technique where skilled animators design a particular sequence by first drawing "extreme" positions in the action - called keyframes. These keyframes are carefully thought out in regard to the action, dramatic presentation, interpretation of mood, reinforcement of story and scene composition. After the keyframes are drawn, the sequence is then passed on to less-skilled artists to fill-in the in-between drawings. To guide the in-between artists, the keyframe artists create spacing charts and motion arcs to plan timing and limb trajectories for the completed sequence. The importance of spacing charts is their ability to convey timing information which is essential for both the expressiveness and the realism of the character. Timing "breathes life" into the character. While very tedious, this process allows animators to convincingly animate the characters they have designed.

To supplement the keyframing technique, the Fleischer brothers pioneered a tool in the 1920's which is still widely used in character animation today, called rotoscoping. For human-like characters with arms and legs, it was an easy and relatively quick way to generate motion. In its purest form, it entails 'tracing over' live action with pencil sketches of the character. The real actor is simply copied over with the character. This form of rotoscoping can still be seen today in the *Tony The Tiger*™® commercials as well as the *McGruff the Crime Dog*™® public service announcements.

Disney animators, however, found that this pure-rotoscoping approach didn't recreate the illusion of life and that characters created in this fashion often moved in an unnatural and awkward manner. Animators chose instead to view rotoscoping as a guide for complicated movement, used for generating new sketches rather than simply drawing the character over the live actor. This technique, which could be dubbed half-rotoscoping, continues to be used in Disney pictures today. A film heavily employing

™® McGruff The Crime Dog is a registered trademark of the National Crime Prevention Council rotoscoping in recent years was *Who Framed Roger Rabbit*© which combined animation and live action.

The most revolutionary change in the last 20 years is the advent of computer-based animation. It has resulted in an explosion of new tools available to the animator. In the early 1980's, the computer became powerful enough to become an attractive option in the creation of animation. Computer-based animation presented a new set of problems with no direct analogy to traditional animation. Yet, it also presented some definite advantages. Lasseter in his famous 1987 Siggraph paper [2] described some of these advantages including how complex motion could be built in an additive manner using animation layers. His paper also presented solid examples of how the Disney principles could be followed in computer-based animation.

The first tools introduced for character animation centered around a motion control technique called parametric keyframing [3][4] which was a derivative form of traditional pose-planning. In parametric keyframing, tools are designed to allow the user to interactively manipulate low-level motion parameters such as joint angles and coordinates. To create movement, the user configures the rigid body for the articulations at "*key*" frames and relies on the system to generate intermediate frames. It is a notoriously time consuming task. At the same time, high-level interactions, such as body motion in its environment and interaction with other figures, must be explicitly

™® Tony The Tiger is a registered trademark of the Kellogg Company

© Who Framed Roger Rabbit is copyright of Walt Disney Company and Amblin Pictures expressed. In the end, the realism of the movement relies solely on the experience and talent of the animator.

Other motion control techniques have been introduced to abstract away from low-level joint angle manipulation and allow higher-level specification. They attempt to achieve realistic motion while minimizing the workload of the animator. Two of interest, because of their use for realistic character movement, are inverse kinematics and hybrid kinematic-dynamic techniques. The former is often incorporated in the latter.

In inverse kinematics, the user specifies an end-effector position, removing the necessity of specifying intermediate angles. This eases limb positioning and allows motion generation by encoding knowledge on how the joint angles change given the position of the end effector. In practice, however, these redundant joint angles are often specified to reduce solution space, thus better controlling the results. The strength of this technique is the ability to position end-effectors at precise coordinates, e.g., specifying exact footholds for a walking motion. Often inverse kinematics is used in conjunction with locomotion control techniques where new movements are interactively generated from a set of reference movements. These reference movements are based on walking and running gait patterns where an algorithm generates a movement based on some high-level specification of the animator. Believable legged locomotion has been achieved [18] with these *procedural* systems.

Hybrid kinematic-dynamic models [5,6,7,8,9] are procedural systems which combine kinematic constraints with physically-based modeling. In these systems, motion is expressed as sets of constraints that, when used in simulation, yield solutions which can be used to describe movement. The allure of these techniques is that they produce natural looking motion. Unfortunately, they are highly technical and require significant expertise on the part of the user to generate an intended movement. Additionally, while physically realistic, the resulting motion is typically not expressive and thus unappealing for character animation.

An alternative to procedural systems for high-level specification are techniques which capture motion from live actors. In performance animation, live subjects act out the parts of the characters while special hardware captures information necessary to describe motion. For humanoid characters, it can be an efficient means to capture realistic motion. Its drawbacks are its cost and the necessity for technically knowledgeable people to deal with the complexities of both hardware and software.

Whether animation is attempted by a traditional technique or a computer-based technique discussed above, reusing generated motion is an important issue. A drawback of both traditional and computer-based animation is that the motion created is not easily adaptable or reusable. If motion is slightly off, it often requires repeating the process from scratch. Some of the most recent computer graphics research has been aimed at addressing this problem. Researchers are working on techniques to adapt and blend existing motions into new motions[10,11,12,13] borrowing from other fields such as signal processing [10]. Current techniques work best when used to blend similar motions or to simply exaggerate motion.

1.1 Shortcomings of previous methods and proposed resolution

While the background reflects the computer's strong presence in the field of animation, there are clearly drawbacks to existing computer techniques. The tradeoff between high-level specifications and low-level parametric keyframing is primarily flexibility and control. While alleviating much of the workload of the animator, solutions utilizing motion control algorithms often require a programmer to implement new motions and their input is so technical and non-intuitive that they are not likely to receive acceptance by animators. In addition to these problems, the techniques are computationally expensive and automated approaches lack the essential expression and personality caught by the animator's eye. Another drawback is that motion is not easily adaptable which inevitably means more work for the animator. Often, the animator has to repeat the process from scratch if a motion is incorrect. Finally, to make computer tools more attractive to traditionally-trained animators, computer-based techniques should build upon rather than replace concepts the animator is familiar with. For the computer to continue to become an increasingly useful tool for the animator, it should build upon the same techniques that the animator has relied upon in the past to bring life to characters. Further, the usability of the tools should not be so technically complicated that they frighten away the animator.

The approach presented here attempts to remedy the problems mentioned above. The system described in the following chapters demonstrates how traditional animation techniques can be applied to computer animation to provide motion control solutions that are effective and can be used by the animator in an intuitive fashion. The collection of tools which compose the system illustrate the viability of this approach by giving solid examples of how easy-to-use and effective computer tools can be created based on these proven techniques. The system employs a rotoscoping technique familiar to the traditional animator but presents a method for adapting this technique to the computer. A method for employing spacing charts is presented to demonstrate how to bring animated characters to life by endowing them with personality through expressive and believable motion. The animator follows a well-defined process which begins by generating the initial motion with rotoscoping followed by the use of spacing charts to tweak the motion for desired results. Unlike many other tools, the system is relatively easy to understand so that the animator will not encounter a technical barrier preventing him from using it.

To appeal to both traditional-based and computer-based animators, all motion specification is done with graphical tools and requires minimal technical expertise on the animator’s part. Also, because refinement of a motion is an integral part of the process, the system presented offers editing features which allow the animator to revisit a motion and tweak it to a desired result.

Traditional animation encompasses a large collection of techniques that have been used for decades to bring cartoon characters to life. The traditional animation techniques employed here include rotoscoping, which offers a general solution to the motion control problems pertaining to animating a character, and spacing charts, which are used to describe the timing of the character for the purpose of enhancing the illusion of life. When used in combination, these techniques are capable of producing convincing and interesting character motion. Rotoscoping is used to generate the initial (rough) motion and then spacing charts are employed to tweak and enhance the movement. Implementing these techniques as computer-based tools is an important step in providing tools that are more intuitive to the animator. An overview of the system employing these techniques is given in the following chapter.

Chapter 2

System Overview

This thesis demonstrates the viability of applying traditional animation techniques to computer animation by employing two techniques from traditional animation, namely rotoscoping and spacing charts. As discussed in the introduction, rotoscoping is a technique for extracting motion from video footage and spacing charts are used to describe the timing of the character over a sequence. These techniques are implemented in this system with computer-based tools presented in the following chapters. These tools fall into three categories. They are three-dimensional rotoscoping, two-dimensional rotoscoping and motion editing. The tools are designed to be intuitive to use and to correlate well to the cell-animation techniques already understood by animators. The rotoscoping tools allow the animator to extract motion from video as well as other visual sources. They are designed to provide a general solution to the motion control problems associated with character animation. The motion editing tools are used to mimic spacing charts. The editing tools can be used to add timing, create sharper movements, or exaggerate motion for effect. When used together these tools can be an effective means for animating a character.

The pipeline of the system is illustrated in Figure 2.1. The system is specifically designed to be an independent pipeline component which can be integrated into an existing animation pipeline. Initially, a rigid body is created in a modeling program and then imported into the system. The animator generates a specific motion task by using the rotoscoping software to extract motion from recorded video footage of an actor. When he is finished generating the initial motion, he can then modify the character's motion path by mapping the character's motion to a curve. Next, he uses the system's editing tools to progressively tweak the motion until he is satisfied that his intent has been met. When he is satisfied with the motion, he then exports the model along with the motion data to a general purpose animation system.

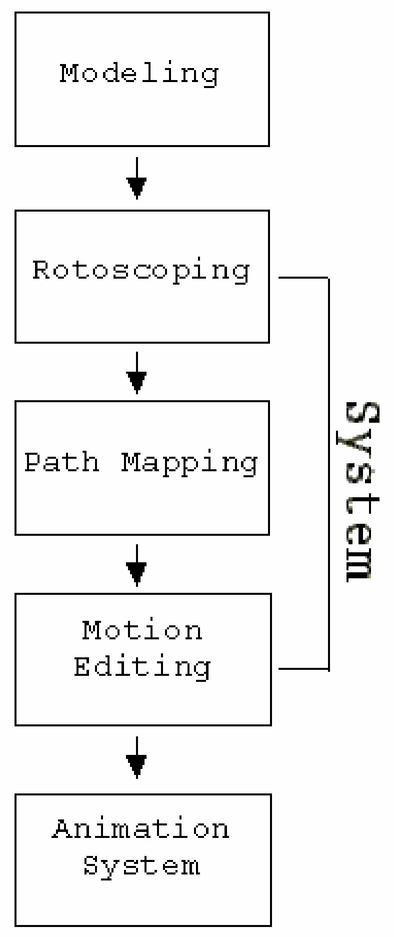


Figure 2.1 System Pipeline

Rotoscoping has been around in traditional animation since the 1920's. The motivation for using rotoscoping as a motion control technique is that it is widely used by traditional animators, simple to use, inexpensive and greatly accelerates the keyframing process. The system is designed to correlate closely to the cell-animation technique well understood by animators. In the original traditional animation technique, the animator projected a frame of film onto a transparent cell and then drew over the image of the actor with a character sketch. In this system, the animator follows a similar technique. The animator is given two windows each containing a video frame of the actor from a different vantage point. To indicate link orientations, the user traces line segments over the limbs of the actor. This is repeated for all limbs until complete stick figures appear in each window. Fortuitously, this is an easier process than the traditional animation technique because there is no need to *flesh-out* the character sketch. The stick figures are adequate for describing the necessary limb orientations.

Rotoscoping is an attractive alternative to other approaches. It provides the animator with an inexpensive way to generate motion which is more believable and can be specified much more quickly than with parametric keyframing. As described above, it requires very non-technical input, thus making it more attractive than most high-level, physically-based approaches. And unlike hardware-based motion capture, it is relatively inexpensive and doesn’t require complicated calibration.

The system presented here offers two rotoscoping tools. The three-dimensional rotoscoping software provides a general solution to the motion control problems associated with character animation. It is used to capture complex motions and requires multiple camera views of the recorded motion. Alternatively, the two-dimensional rotoscoping software is provided to extract motion from a single view. Instead of being limited to using video specifically recorded for rotoscoping purposes, the animator can also extract aesthetically correct motion from drawings and cartoons. The special tools developed to aid in this task increase the overall flexibility of the system.

The three-dimensional rotoscoping software employs solutions which produce good results even for complicated motion. As mentioned above, the animator traces over images of the actor in two camera views to indicate link orientations. This information is then triangulated and used to generate a true three-dimensional solution. This contrasts this software from earlier rotoscoping systems which appeared to derive link orientations directly from the camera view-planes and not in three-dimensional space.

Figure 2.2 shows a hardware/software pipeline for the three-dimensional rotoscoping software. The first graphic in the pipeline illustrates two cameras pointing at a common focal point. Each camera is perpendicular to a view-plane corresponding to the coordinate system of the real actor. In practice, this relationship is established by placing two strips of intersecting white tape on the floor to help position and orient the cameras. Next, the video recorded by the camcorders is compiled on an analog VCR deck. This analog video is then imported into the computer using a video capturing card. This card converts the analog video into digital video streams readable by the software. The animator then uses the video footage of the actor in the rotoscoping system to specify the motion. After the motion is created, the motion data can be edited or exported

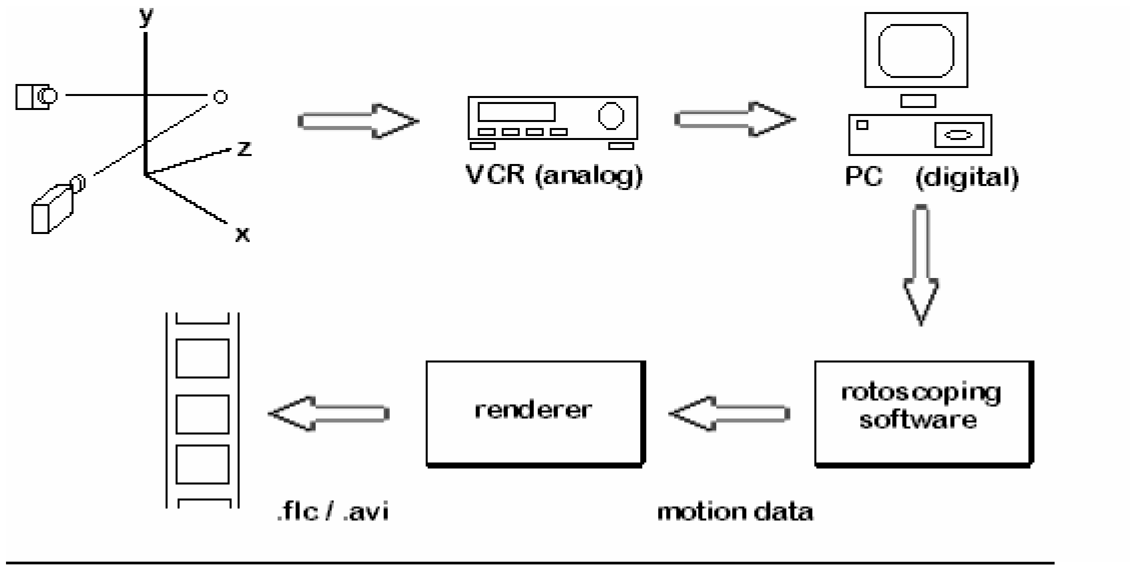


Figure 2.2 Hardware/Software Pipeline for three-dimensional rotoscoping

immediately to an animation system or low-level rendering package. The package used to render the images in this paper was the Persistence of Vision™ Ray Tracer, a freeware raytracing package available over the Internet.

The two-dimensional rotoscoping software provides tools for extracting motion from a single view. This allows the animator to extract motion from drawings or previously created animated work. In the case of drawings, the animator can draw keyframe sketches to plan the motion without detailing exact time-position information. The software presented here can be used to derive velocity directly from the character itself. Alternatively, the software can be used to derive motion from a video clip where only a single view is available. In limited animation, different parts of the character are animated separately and then put together for a composite for a particular sequence. This

™ software is Copyright 1991,1996 by the POV-Ray Team way transparent film cells can often be reused. The composite creates the variation. To maximize reusability, all the drawings correspond to a few simple camera angles. For this reason, the motion is conducive for capturing by this system.

Once the initial motion is captured, the animator has the option of mapping the motion to a path. The path is created interactively and footprints are generated for the animator to review. Mapping a character's motion to a path works best for simple gait motion, such as walking or running. The motion generated in the two-dimensional rotoscoping software is usually ideal for this type of mapping. With two-dimensional rotoscoping, the character tends to move in straight lines, so mapping the motion to a path is straight forward. For complex motion generated in the three-dimensional rotoscoping software, path mapping often doesn't make sense and is skipped. For example, mapping a complex motion, such as a martial arts routine, to a twisting path would result in movement that would be difficult to predict.

Finally, the animator can use the system's motion editing tools to mimic spacing charts. Spacing charts are used to add personality to the character or to enhance the realism of the character's movement. In this system, the motion editing tools allow the animator to create variations of a generated motion. Traditional animation artists know that the character's timing is the essence of the art. They endow personality to the character by making the character always walk or move with meaning - to never pause unless there is a reason for it - when the character does pause, it should pause as long as it can. Besides adding personality to the character, motion can be edited to enhance physical realism or to correct for errors resulting from the interpolation technique used to generate intermediate frames. Spacing charts are often used to model the effect of weight. Similarly with the motion editing tools, the motion can be adapted so that the character appears to be affected by gravity, to possess super-human ability, or to appear sluggish as he pulls a heavy weight.

The tools which constitute this system are discussed in detail in the following chapters. Described are the strengths of these tools, a description of their functionality, their usage by the animator, and how they were implemented. The final chapter discusses what has been accomplished with this system as well as the direction of future work.

Chapter 3

Three-dimensional rotoscoping

This chapter covers the three-dimensional rotoscoping software written for this system. Its purpose is to capture the motion of an actor from recorded video. The goal of the 3D rotoscoping software is to provide a general solution to the motion control problem associated with character animation. Described below are the strengths of three-dimensional rotoscoping, a comparison to earlier attempts to implement these systems, and a description of this system.

The overall system described in this thesis actually employs both three-dimensional and two-dimensional rotoscoping software. In this chapter, the features of three-dimensional rotoscoping software are discussed, and the features of the two-dimensional rotoscoping software are discussed in the following chapter. The two-dimensional software is similar to other early rotoscoping systems in that it doesn't provide a general solution. Nonetheless, it is included because it has special tools to extract motion from a single view.

3.0 Overview

As described in the introduction, motion control for character animation is a difficult problem. Many computer scientists have developed solutions to address the problem. The drawback of most of these techniques is that they offer a technical barrier for the potential user, namely the animator. Many require very mathematical input or require a programmer to hard-code new motion tasks.

The strengths of the system described here are that it is intuitive to use, it accelerates the keyframing process and it correlates to a cell-animation technique well understood by animators. And unlike hardware-based motion capture, it is relatively inexpensive and calibration isn't very technical. For these reasons, this computer-based rotoscoping technique should provide an attractive motion control solution to character animation.

The ability of this software to produce a general solution to the motion control problem contrasts this system to earlier computer-based rotoscoping systems. In earlier systems, used in commercial animation in the 1980's, the motion captured appeared to be planar. Instead of providing a three-dimensional solution, they seemed to capture two-dimensional motion and apply it to a three-dimensional character.

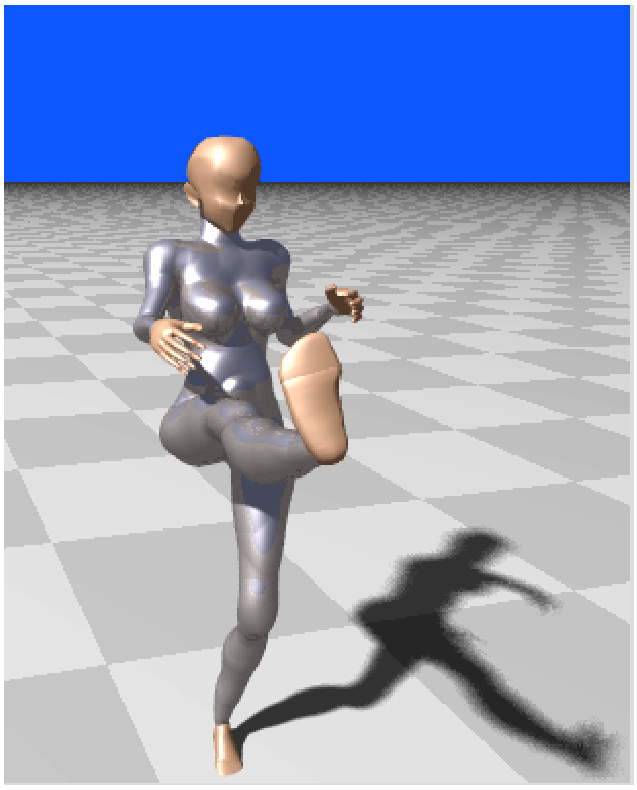


Figure 3.1 Motion from a martial arts routine captured with the three-dimensional rotoscoping software.

The reason the motion in earlier systems was planar appears to be the manner in which link rotations were solved. Simply solving rotations based on how the actor's limbs appear on the camera view-planes will not produce accurate results. For this reason, only specific types of motion could be captured. Usually, figures moved in straight lines and rotations only appeared to be specified for a single axis. In contrast, to correctly solve for link rotations, the information that is extracted from the camera views must be triangulated and then solved in three-dimensional space.

The methodology used in this software takes the two-dimensional information extracted from the camera views and combines it into three-dimensional information before solving link rotations. This yields a true three-dimensional solution. There are difficulties in constructing three-dimensional data in this way. Most importantly, the information is distorted by perspective. Also, certain limbs may be obscured during the recording. Therefore, some guess work is required on the part of the animator entering information.

The technical description of how link orientations and object positions were derived is detailed in the following sections. In general, the process is as follows. Link orientations are solved by transversing the link structure of the articulated figure and solving for the rotations that describe one link frame relative to the next, based on the information extracted from the camera views. An analytical solution is initially calculated, however an iterative search for a solution must be used for complex orientations. To extract position, bounding volumes must be calculated for both the configured articulated figure and the actor as he appears in the camera frame. This will help by relating the coordinates; but, due to the effects of perspective, the relation is not exact.

Before the animator can begin working with the motion in the software, the motion must be performed by an actor and recorded. In the recording session, two cameras are used. The position and orientation of the two cameras must follow certain criteria. The cameras must be at right angles, aimed at a common focal point and equidistant to the focal point. If these conditions are violated, then the data triangulated by the software can be adversely affected. Fortunately, there is a simple means to calibrate the cameras. Tape can be placed on the floor to calibrate the orientations of the cameras. Two intersecting lines of tape are placed on the floor perpendicular to each other. From the view of the camera, the two lines of tape should appear horizontal and vertical. The intersection of the lines of tape should be centered in the cross-hairs of the cameras since it is the focal point. Finally, the position of the cameras should be measured to ensure that they are equidistant to the focal point. Once the analog video is recorded, it must be converted to a digital format and imported to the computer. Fortunately, low cost video capturing cards are readily available and software, called codecs, can be used to digitize and compress analog video into digital video streams.

After the performance is recorded, the animator can begin working with the three-dimensional rotoscoping software. As mentioned in the introduction, rotoscoping in traditional animation has been around a long time. The goal of the end user view for the interface in this software is to allow the animator to enter information in a fashion similar to cell-based rotoscoping as well as provide the functionality expected by computer animators using parametric keyframing systems. The following things must be done to generate motion data for one keyframe:

- The animator enters stick figures over the captured video to approximate the pose of the actor.
- The software triangulates the inputted data and solves for rotations.
- The animator tweaks the results as needed.
- The software calculates the character's position based on the apparent position of the actor.
- The animator ensures the character's footholds are correct.

In its purest form, cell-based rotoscoping involves tracing over the actor in the film cell with a cartoon drawing. Similarly, the animator using this three-dimensional rotoscoping software traces over the actor with a stick figure. Instead of projecting a frame of film onto a cell (the classic procedure in traditional rotoscoping), the animator is given two windows each containing a video frame of the actor from a different vantage point. To indicate link orientations, the user simply enters line segments into the windows which trace over the limbs of the actor. This is repeated for all limbs until complete stick figures appear in each window.

Of course the animator will want immediate feedback after entering data. This is because it is useful to monitor the results throughout the process to catch small problems before they become large problems. Unfortunately, it is often difficult to give the animator real-time results when problems require a large number of computations to solve. To minimize the amount of time the user has to wait for an answer, the software offers a compromise. Because some solutions are easier to generate than others, when configuring the articulated figure, this system doesn't always use the complex solution. For example, it is much simpler to solve rotations for a man walking in a straight line than for a complex martial arts routine. In fact, the motion of the walking man is nearly planar and doesn't require a full three-dimensional solution. In this system, the user is given a quick analytical solution based on his input. If the results are satisfactory, then he can move on to the next keyframe. Otherwise, the application can iteratively search for a solution.

After the software has solved for the link rotations, the animator examines the results in the rendering window, see Figure 3.2. Typically, the result will match the stick figures, however, slight manual adjustments may be needed. Many things can affect the results including camera setup, perspective and how well the stick figures approximated the actor's pose. The animator will usually want to examine the articulated figure from different vantage points to determine what corrections should be made. The animator can tweak the results by selecting limbs on the articulated figure and then manually rotating them with the mouse.

In addition to tweaking the figure's pose, there are times when manual input is required. While entering stick figures is a simple means of data entry, some angles can't be described in this manner. For parts of the articulated figure which are centered about the vertical axis, i.e., the chest, head, and pelvis, vertical line segments give no clue as to the rotations about the vertical axis. In this situation, the animator must manually orient these parts before drawing the stick figures. Also, the exact position of the actor is difficult to derive from the video footage. The view volume in which the actor performs is greatly distorted by perspective. A scale must be established between the actor in the image and the articulated figure. This can be accomplished by calculating bounding volumes for the configured articulated figure and the actor. This must be done at each keyframe after the articulated figure has been configured. Since the real actor will move back and forth in his volume throughout the sequence his size will continually change relative to the articulated

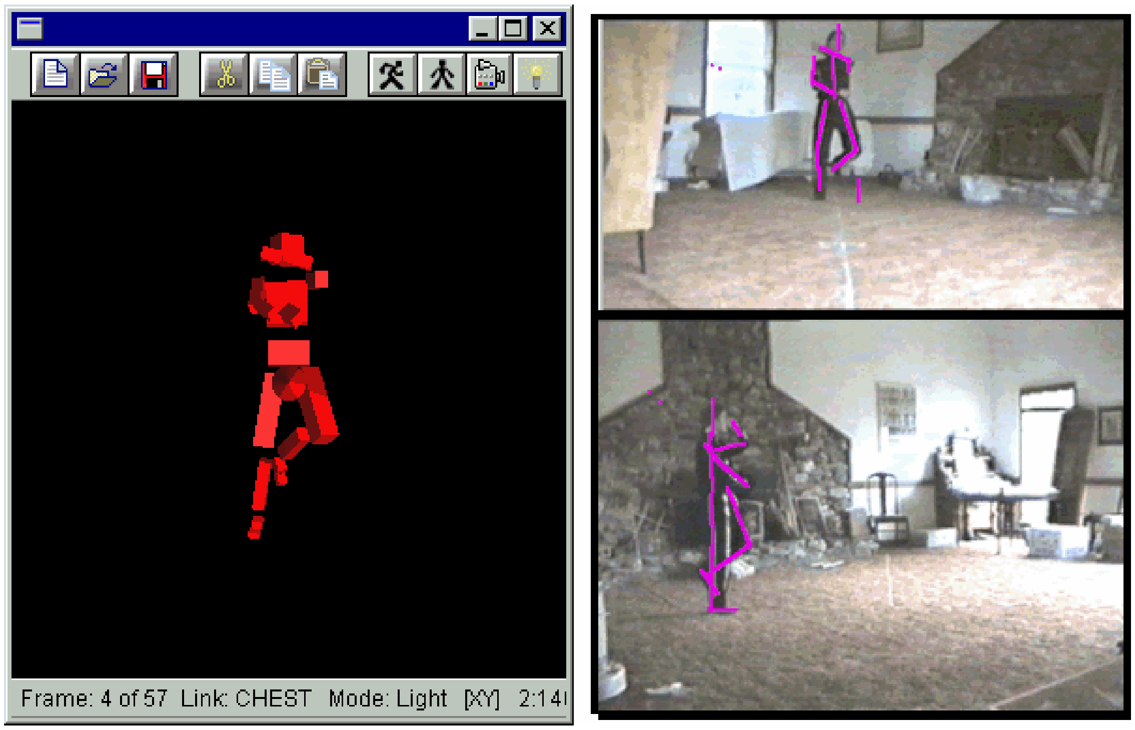


Figure 3.2 The interface for the three-dimensional rotoscoping software. On the left, the animator can review the results and make manual adjustments in the rendering window. On the right, the user enters stick figures to approximate the pose of the actor.

figure. A special edit mode is provided to help the animator fine tune position. The articulated figure is shown in a multiple exposure fashion so the animator can inspect the footholds from one keyframe to the next. In traditional animation, it is well known that for gait motion each leg goes through a support phase and a transition phase. For the support phase, the foot of the support leg should remain fixed in place. Otherwise, the character will appear to slide. By examining the foot of the support leg for a multiple exposure, the animator can detect sliding. Sliding of the support leg can be fixed by simply re-positioning the character so that the foot remains in place until the leg enters its transition phase.

Once the articulated motion has been configured for one keyframe with respect to link orientations and position, and the animator is satisfied with the results, then the process is continued for the other keyframes in the sequence. The rendering window offers a playback mode enabling the animator to review his work at any time.

The utility of traditional rotoscoping has been known for years. Nonetheless, there are some advantages to computer-based rotoscoping compared to its traditional cell-animation counterpart. First, there is no need to *"flesh-out"* the character in detail. The animator only needs to extract information about the character's pose which can be accomplished by drawing simple stick figures over the character. There is no need to create complicated drawings. Properly placed lines suffice for the required data entry. Also, the traditional animator using cell-based rotoscoping techniques has to be concerned with issues of shading, perspective, and depth perception. Fortunately, these can be ignored in the computer-based technique. For these reasons, the computer-based rotoscoping could potentially be a faster process than its traditional animation counterpart. The current drawback is the time involved in finding solutions for difficult orientations using an iterative technique. Extracting three-dimensional information from two-dimensional images is an active area of research and is not trivial. The details of how position and rotations were extracted for this software are discussed in the following sections.

3.1 Rotations

In the introduction, a distinction was made between pure-rotoscoping and half-rotoscoping. In pure-rotoscoping, the live actor, as seen in the film cell, is traced over with the artist’s character. While in half-rotoscoping, the actor was used as a reference for creating new sketches. In this system, the input is analogous to pure-rotoscoping. The animator extracts link orientation by tracing over the actor in the window frames. For each keyframe, the animator must enter a series of line segments. Each line segment corresponds to a link in the articulated figure. When all line segments are entered, we have a ‘stick’ figure in each camera window which mimics the pose of the real actor.

The next step is for the software to solve for the correct configuration of the articulated figure. This is the configuration which matches the apparent orientations of the line segments entered by the animator. The general technique for configuring the link structure requires constructing line segments in the view frame and then solving for the angles of rotation which relate a particular link frame to the frame of it’s predecessor. View frame coordinates are constructed by relating the coordinates of a vertex from one camera window to the coordinates of a vertex in the other, see Appendix A.

Solving for orientations in this way seems simple enough. Surprisingly, it often requires an iterative technique. There are well known analytical solutions for finding the angles of rotation necessary to describe the orientation of a line segment. However, a complication arises because these solutions typically use a scheme of rotations less general than those used in articulated figures. As seen in Appendix B, a two angle rotation scheme gives a unique two angle combination for any orientation and therefore is used as a general recipe. However, rotations are lost around the third axis. While a three angle rotation scheme provides a more general description of orientation, it can’t be

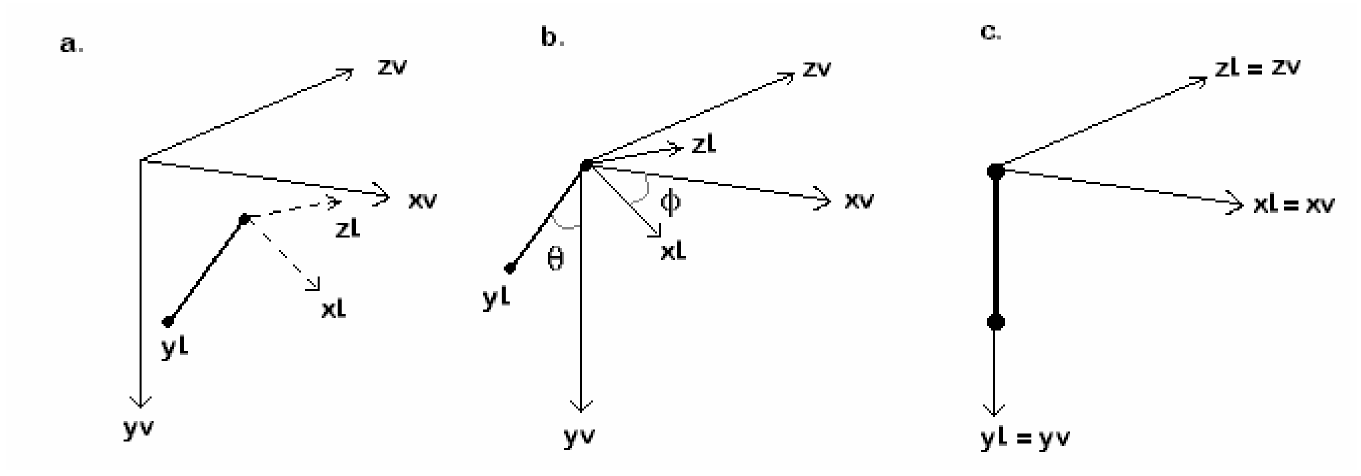


Figure 3.3 Angles between a Link Frame and the View Frame. In (a) is the line segment representing a link orientation. In (b), the segment is translated to the origin of the View Frame, then find ϕ and θ. In (c) the line segment is checked. It should be coincident with the View Frame axis if ϕ and θ are correct.

solved for a unique solution analytically. For one orientation, there are hundreds of possible three-angle combinations. *Consequently, a two angle rotation scheme must be used to solve for the orientation and the result is later stored in a three angle structure.*

As mentioned in the previous paragraph, the link orientation will be solved using two angles. The two angles will be referred to as ϕ and θ in the following discussion. Since the link structure uses a three angle scheme, ϕ and θ may refer to different axes when moving from one link to the next. To find the rotation angles ϕ and θ which relate one frame of reference to another, (See Figure 3.3) start by constructing a line segment in the view frame from the animator's input. Then treat the line segment as a direction vector representing a particular axis in the frame associated with the link. The axis depends on what the Euler angles ϕ and θ represent, see Appendix B. Assume the y-axis is a yaw-roll rotation in this software. Translate it to the origin of the view frame, and then rotate the line until the link frame coincides with the view frame. When the angle between y-axis$_{\text{linkframe}}$ and y-axis$_{\text{viewframe}}$ is close to zero, then the rotation angles don't need to be computed anymore. This gives the link-to-view transform. The equations to solve the rotation angles are given below.

$d1 = \sqrt{((z_1-z_0)^2 + (x_1-x_0)^2)}$ $dph = \sqrt{((z_1-z_0)^2 + (x_1-x_0)^2 + (y_1-y_0)^2)}$

$\phi = \tan^{-1}((y_0-y_1), d1/dph)$ $\theta = \tan^{-1}((z_0-y_1)/d1, (z_0-z_1)/d1)$

For this particular application, two additional considerations must be taken into account. First, the rotations needed actually relate the current link frame to the frame of the previous link. As mentioned in appendix A, the orientation of a particular link is actually the result of multiplying all the control matrices from the root of the link structure to the link. Second, while a two angle rotation scheme is used to solve for the orientation of a particular link, the link structure as a whole uses a three angle rotation scheme. This requires an iterative solution, as opposed to the analytical one described above. This leaves two remaining problems that must be solved. First, the technique that uses a line segment (described in view frame coordinates) to give rotations relative to the frame of the parent link, must be adapted. Second, an analytical solution may not always suffice. If it doesn't, what can be done?

To adapt the first technique, find the rotations necessary for the base vectors of the current link to become coincident with those of the frame of the previous link. This is done by taking the technique above and making it even more general. In Figure 3.4, the direction vectors Q_0 and Q_1 are seen in relation to themselves and the y-axis. Q_1 is the y-axis for the link frame whose orientation must be to solved, and Q_0 is the y-axis for the

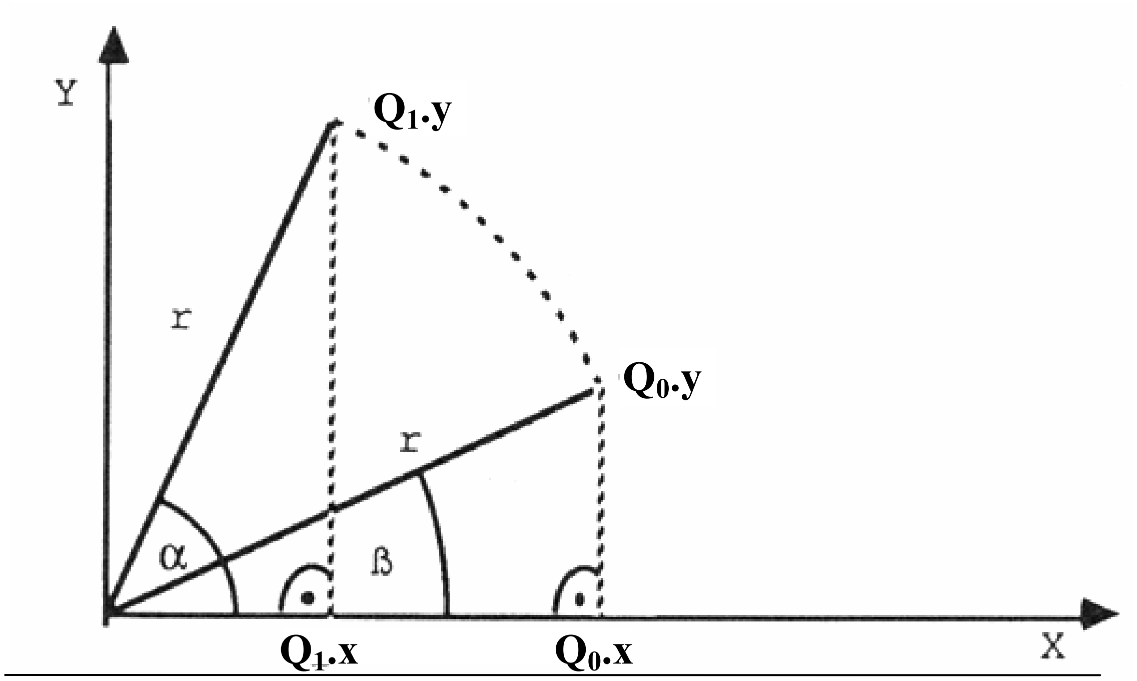


Figure 3.4 Relating one Link Frame to Another. The angle α relates an axis from the link's frame to that of the parent link. The angle β is the total rotations for that axis from the root of the link structure to the parent link.

link frame which immediately proceeds it in the tree. As mentioned in Appendix A, rotations accumulate as the link tree is transversed in a top-down fashion. Consequently, the angle β is actually the combined y-axis rotations from the root of the link tree to the link frame immediately proceeding the frame of the link that must be solved. And since Q_1 represents the y-axis of the current link frame, the angle α is the y-axis rotation angle which relates the current link frame to the previous one in the tree, i.e. Q_1-Q_0. To find the relative angle α use the addition theorem given below.

$$\cos(\phi-\theta) = \cos(\phi)\cdot\cos(\theta) + \sin(\phi)\cdot\sin(\theta)$$

$$\sin(\phi-\theta) = \sin(\phi)\cdot\cos(\theta) - \cos(\phi)\cdot\sin(\theta)$$

Essentially, this is the same technique as before but the equations solve for relative angles versus absolute angles about the axes of the global system (the view frame in this case). In each link frame, these equations solve for two intermediary angles which would make the current frame of reference coincident to that of it’s parent.

Since the link structure as a whole uses a three angle rotation scheme, the above analytical solution only works if all preceding link frames were solved, where ϕ and θ related to the same axes. For example, one link may be solved for a yaw-roll combination, while another is solved for a yaw-pitch orientation, while yet another is solved for a roll-pitch combination. For completeness, all two angle rotations are allowed with a hard-coded preference for each link, i.e. default settings. However, the analytical solution will only work when angles can be incrementally increased, so they must be summed up to a grand total. Remember, this was the merit of the two angle scheme. For this to work with the link structure, the path from the top of the tree to the link in question must use identical combinations. Such as, a series of yaw-roll combinations. Once a different combination is found, e.g. a yaw-roll preceded by a yaw-pitch, the analytical solution doesn’t work. This is intuitive, because now the rotations about three axes are known and as mentioned before three angle (Euler) combinations aren’t unique recipes for orientations.

There are a couple of ways to check and see if an analytical solution will work. As described above, an algorithm can be written which traces a path from the root of the tree to every link frame analyzing the combinations. Or, simply try the analytical solution and check the answer. To check the correctness of the solution found, construct a coordinate transform and see if the calculated answer matches the actual line segment entered by the animator. Figure 3.5 illustrates what shall be accomplished. The D

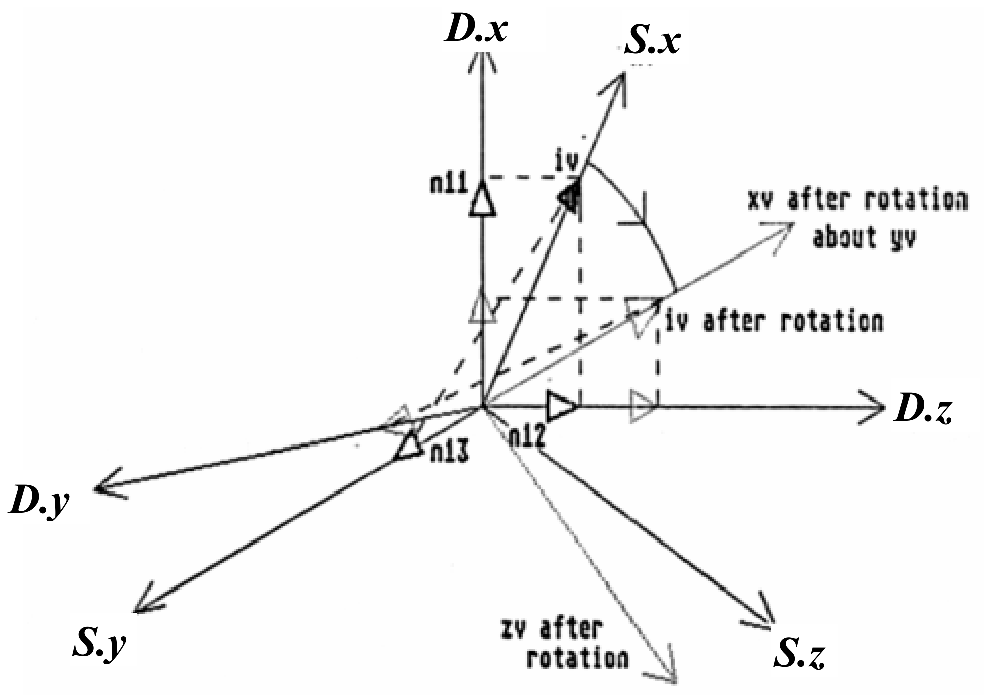


Figure 3.5 Rotation of Base Vectors using Direction Cosines

(destination) is the line segment originally entered by the animator, in view frame coordinates. While S (source) is a new non-initialized line segment pointing down the y-axis in the view frame. S is of equal magnitude to D, where S=<0, |D|, 0>. Multiply S by a transform based on the calculated angles, ϕ and θ, and check to see if the result is coincident with D, or within a certain tolerance. The necessary transform is described below.

If the coordinates of the vertices of an object are known in one reference frame and need to be known in another, all that is needed is a simple coordinate transform. Since, as far as rotations are concerned, there is always a linear relation between two sets of coordinates, a matrix of general terms can be constructed. The elements n11, n12, etc. are specific to the relative orientation of the two reference frames. The elements are simply the cosines of the angles between the axes of the reference frames, commonly called direction cosines. The element n11 is the cosine of the angle between x_{link} and x_{view}, n12 is the cosine of the angle between y_{link} and y_{view}, etc. After multiplying S by the coordinate transform check to see if the result is coincident with D. If it is, then no iterative solution is needed and the angles in the link's data structure can be stored. If the line segment is not coincident after the transformation, then the angles must be found with an iterative technique. This is typically the case when there is a mixture of rotations, e.g. yaw-roll followed by yaw-pitch. The iterative solution is simple but time consuming. It involves gradually incrementing the direction cosines for ϕ and θ until a coordinate transform is found which properly rotates S into D. It is a search algorithm. On an Intel Pentium processor running at 133mhz, it can take a minute to configure an articulated figure consisting of seventeen links.

3.2 Position

After configuring the link structure, the final step is to use the apparent position of the real actor, in the view frame, to place the virtual actor in the world frame, see Appendix A. As simple as the required information is (a single point!), two things complicate the definition of the transform. First, as mentioned before, there is no predefined relationship between the coordinates of a vertex in the view frame (the real world) to the coordinates of a vertex in the world frame. The correspondence that will be established needs to be adjusted at each keyframe. Second, the effects of perspective when extracting motion must now be taken into account. In deriving link orientation the effects weren't severe enough to require a correction. However, the effects of perspective are more noticeable when attempting to extract an exact position.

3.2.1 Bounding Volumes

Ignoring perspective for a moment, assume that both window frames are at right angles. This reduces the view-to-world transform to a simple problem of magnitude. If the ratio between the length of the base vectors in the view frame and the length of the base vectors in the world frame can be estimated, then the solution is a simple scaling transform. To do so, compare the area of the real actor in the view frame to that of the fully configured virtual actor in the world frame. This will require calculating a bounding volume for the link structure and estimating a bounding volume for the real actor in view frame coordinates. When calculating the bounding volumes, the articulated figure must already be configured. This is necessary so that the analogous volumes are compared between the real actor and the virtual actor.

3.2.2 Perspective

When link rotations were found, it was assumed that the base vectors in the view frame were perpendicular to each other. However, since the window coordinate systems used to construct the view frame are based on camera footage, vertices in these coordinate systems are actually based on perspective projections, versus orthographic. Thus, the view frame is actually distorted by the combined perspective of both window frames, i.e. camera views. In a typical diagram of a viewing frustum, the near and far clipping planes would relate to the observable range of the real actor. Unlike a typical

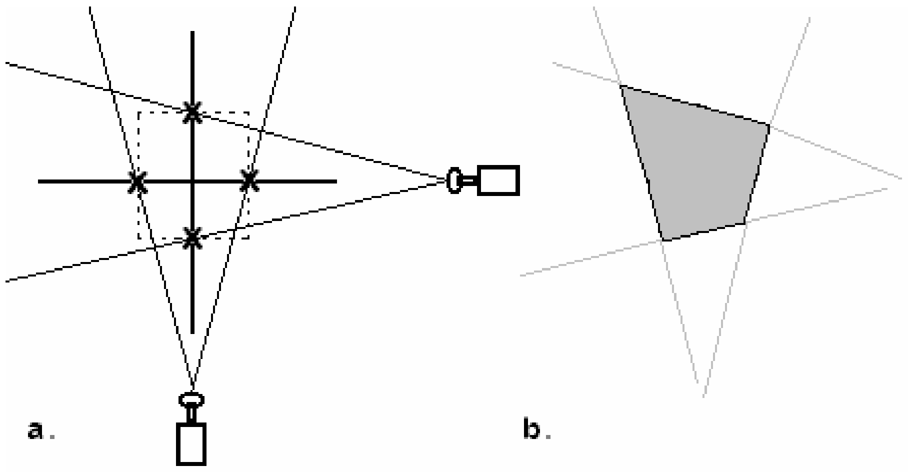


Figure 3.6 Rotoscoping System Frustum. In (a) is the combined field of view of the cameras. In (b) is the shape of the frustum in the xz plane.

frustum, the geometry of the space which is constructed from multiple cameras appears sheared when drawn using ray projectors. In Figure 3.6, the shape of the frustum as captured by this system is illustrated (in the zx plane only) based on the combined projections. The angles of the ray can be derived by comparing the observable field to the distance of the cameras.

There are at least two ways to help counteract the effects of perspective with minimal effort. The first solution is to construct bounding volumes for each keyframe, this relates the scale of the real actor to the figure, it also helps to minimize the accumulation of errors. Since the real actor will appear smaller when close to the far plane and larger when close to the near plane, the bounding volume will grow and shrink as the actor moves back and forth within the camera view. Unlike the real actor, the bounding volume for the link structure retains a fixed size for a given configuration regardless of placement in the world frame. Therefore, the view frame should be normalized to the object frame, i.e. real actor to virtual actor. As mentioned above, this only requires scaling. Once the view frame coordinates are scaled properly, the world frame coordinates can be used for positioning the object’s frame. This yields a good approximation for position. The second solution is to attempt to construct an inverse perspective transform based on the relationship between camera distance and field of view. This involves finding the ratio between the distance of the camera (to the focal point) and the length of the observable field. Unlike the other solution, this requires that measurements be entered into the system for each run. Unlike the other technique this method corrects vertices immediately as they are entered by the animator. Consequently, line segments used to specify link orientations can be corrected along with position.

Figure 4.1 Sketches of keyframe drawings

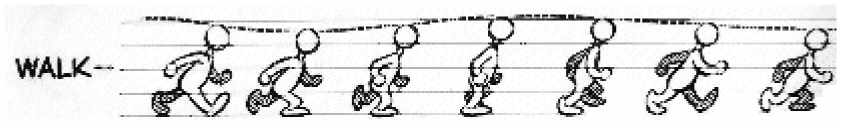
Chapter 4

Extracting motion from a single view

The two-dimensional rotoscoping software extracts planar motion from images. Such a system can be used to capture simple motion, but does not provide a general solution to three-dimensional motion capture. The strengths of the software presented here relate to the novel ways that it can extract motion. It assumes that motion will be captured for a single view. As a result, it provides tools that aid in this process. Unlike the general purpose rotoscoping system described in the last chapter, this software provides solutions for specific problems.

4.0 Overview

There are two primary uses of this software. It can capture motion from artists' drawings and it can capture motion from previously recorded video, e.g., movies, cartoons, etc. In traditional animation, the master animator typically draws the extreme poses of the character performing a task and allows another animator to draw the in-



between poses. As seen in Figure 4.1, these extreme poses are often spaced apart for clarity. As a result, they give no direct indication of the character's relative position from one keyframe to the next. Typically, this information is conveyed with spacing charts and motion arcs. Fortunately, even without this additional information, this software is still useful. Using only the drawings seen in Figure 4.1 as a guide, this software can derive the locomotion parameters by examining the character itself. Alternatively, the software can be used to derive motion from a video clip where only a single view is available. This is the case with previously recorded video. For the best results, the actor or character should be moving across the field in a direction nearly parallel to the screen. Occasionally in movies, we will see actors move across the field in this manner. More often this type of motion can be found in cell-animation cartoons where limited animation is used.

In limited animation, different parts of the character are animated separately and then put together into a composite for a particular sequence. This way transparent film cells can be reused. The composite creates the variation. To maximize reusability, all the drawings correspond to a few simple camera angles. Typically, characters in limited animation move parallel, perpendicular or at a 45-degree angle relative to the screen plane. For this reason, the motion is conducive for capturing by this system. When the character moves parallel to the screen, the motion can be derived directly. However, when the character's motion path is not parallel to the screen, then more work needs to be done, as described below.

One of the main tools of the system is to correct for foreshortening. Foreshortening occurs when a character's path of motion is not parallel to the screen. Foreshortening affects angles in the capture process. Later in this chapter, the technique used to counter the effects of foreshortening is discussed in detail.

Like all the ideas presented in the thesis, the end user view is important. It is preferable to allow the animator to enter all data using the mouse. If information is simple to specify, then the process as a whole seems less intimidating. As always, it is important that procedures are well defined and follow a simple step-by-step process.

To begin, the animator must create a new project in the software. To simplify the process the animator should set certain parameters which best suit the motion trying to be captured. The animator will follow the steps listed below in creating the project.

- The animator selects a static bitmap or a bitmap stream.
- The animator chooses the form of interpolation he wishes to use.
- The animator chooses whether a velocity curve or a trajectory curve should be created.

In the case where an artist's sketch is used, the keyframes may all appear on the same page. If a single bitmap with keyframe drawings will suffice, then a static bitmap should be used. However, if a video clip is necessary, then a bitmap stream must be selected. Sometimes the artist will compile his drawings on a flip-book. In this case, he can create a bitmap stream from previously scanned drawings.

When setting parameters for the new project, the animator must choose which form of interpolation he wishes to use. The acceptable forms for interpolation are linear, B-spline, or Catmull-Rom. Linear interpolation does not produce smooth transitions, so it is rarely used. Catmull-Rom can be useful for motions involving sharp movements. However, the B-spline form is typically the best. The cubic B-spline form provides the smoothest transitions and is usually the best choice for animation. In the three-dimensional rotoscoping system, all motion data is interpolated with this form.

The animator has a choice between two techniques for extracting the character's position. He can specify a trajectory curve or a velocity curve. If a video clip of a cartoon or a movie is being used, then a trajectory curve is the best way to describe position over time. In the editor, the animator can simply create points that mark the position of the character on the image. These marks must be added at uniform intervals, however they don't need to be specified for each keyframe. With this method, we know the character's precise location relative to time. The velocity curve is useful when tracking the position of characters that aren't positioned on the image relative to time. A velocity curve measures distance traveled over time. Instead of storing position, the change in position between keyframes is stored. The keyframe drawings in Figure 4.1 illustrate a character walking. However, the character is spaced for clarity and not for position. In this

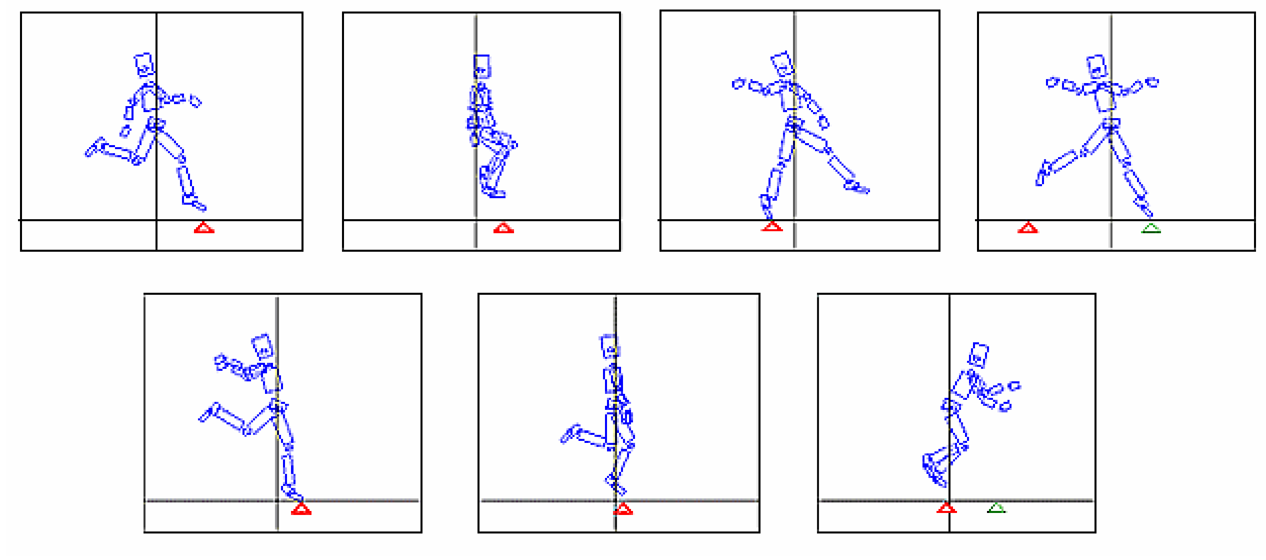


Figure 4.2 Deriving velocity by examining the character's gait over several keyframes

example, the relationship between position and time is not obvious. Fortunately, there is a technique which can be used to extract velocity my examining the character drawings. The following paragraph describes how an animator uses this technique. Later in the chapter, the mechanics behind this technique will be described in detail.

The animator can enter data for the velocity curve, by simply placing a tracking arrow underneath the character's support-leg at each keyframe. In Figure 4.2, we see multiple frames of the character. Each keyframe is displayed in it's own rendering window. The characters are centered in the windows giving no hint to the actual position. In each window we see a tracking arrow pointing to the character's support-leg. By comparing the tracking arrows from one frame the to the next, we can approximate how much distance the character has traveled. This technique works best for gait-like movement where the character has at least one leg in contact with the ground. Finding the character's vertical position relative to the floor is simple. The animator drags a line representing the floor to the character's lower foot by using the mouse.

To capture link orientations for a particular frame, the animator draws a series of line segments over the current image. Each line corresponds to a different link in the articulated figure. When all the line segments are entered, a completed stick figure will appear over the image. The rendered figure, in the rendering window, will be completely configured to match the pose in that view.

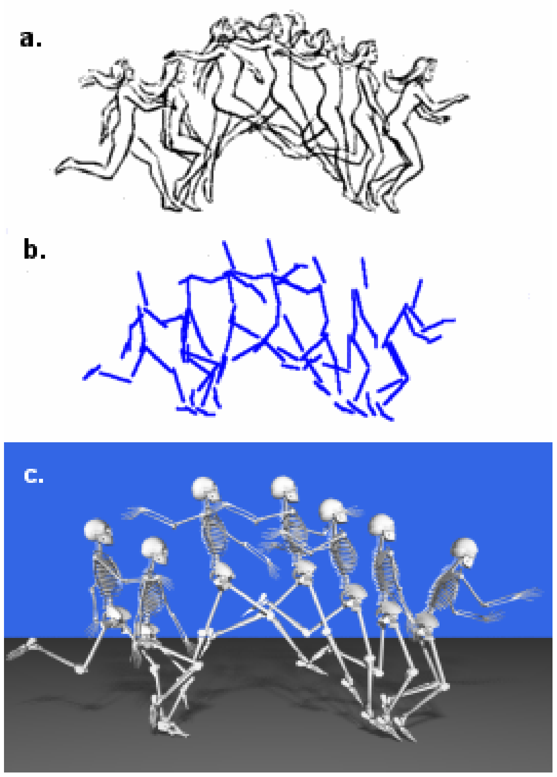


Figure 4.3 Data Entry. In (a) the artist's sketches are shown. In (b) the animator enters stick figures to mimic the poses of the actor. In (c) the motion data is applied to the articulated figure.

Another useful tool allows the animator to create new windows to import images. This gives the animator the option of entering rotations for different axes. In practice, rotations can only be specified around the x and z axes. There is no means to enter pitch (see Appendix B) rotations. The three-dimensional rotoscoping software should be used for this. Since only a two-angle rotation scheme is used, all angles are incremental along the link structure. Every combination is unique, so no iterative solution is needed and all editing is real-time.

The final step for the animator, after he/she has configured the articulated figure for each keyframe and is satisfied with the results, is to map the motion to a path. Mapping a character to a path works best for simple gait motion, such as walking or running. The planar motion generated in this two-dimensional rotoscoping system is ideal for this type of mapping. With this technique, the character tends to move in straight lines, so mapping the motion to a path is straightforward. Using the character's velocity for each keyframe interval, the distance traveled along the path can be found for each time step. From the animator's point of view, he must create a two-dimensional curve and specify a scaling factor. The creation of the curve can be done interactively with the mouse. Setting a value to relate scale is a heuristic means to specify the length of the figure's stride relative to the total arclength of the curve.

The remaining sections in this chapter will discuss in detail how velocity is extracted from character drawings, a method for correcting foreshortening, and how a character's motion is mapped to a path.

4.1 The Foreshortening Problem

The two-dimensional rotoscoping scheme works best when no pitch rotations are necessary. This is the case when the actor is moving in a straight line, e.g. jumping or walking. This technique produces motion similar to that seen in the commercials in the

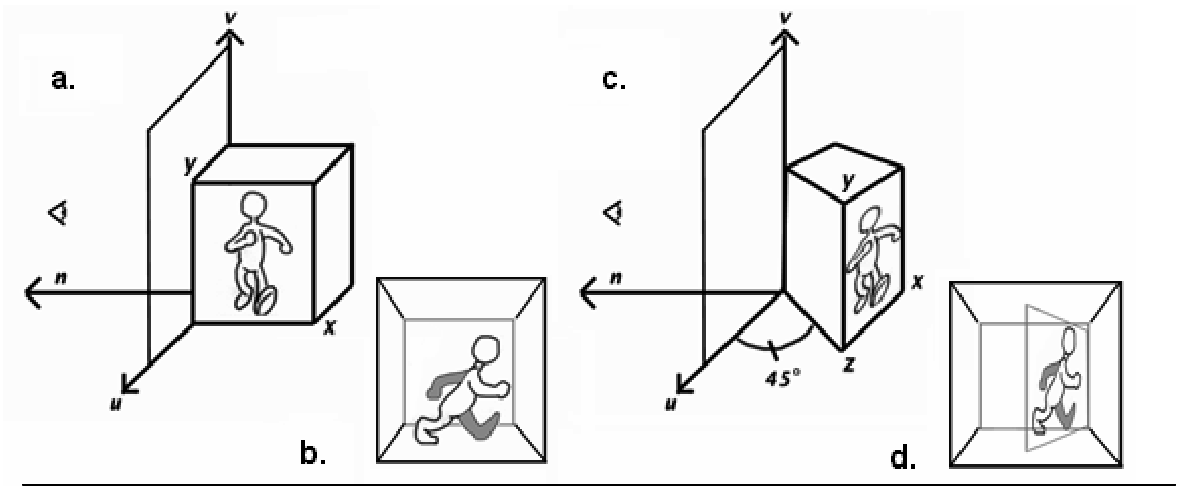


Figure 4.4 Foreshortening Problem

80's which first used rotoscoping as a means of motion capture for 3D animation. The movement tended to be planar, with the characters moving in straight lines. A potential use of this technique is for extracting motion from previously recorded video. Unfortunately, there exists a major obstacle when using video that has not been specifically recorded for rotoscoping. Usually the camera angle is not exactly parallel to the path of motion, even for a simple motion. This is important to the no pitch requirement described above. What will typically be seen is the actor moving in a line at an arbitrary angle until he is foreshortened to our line of sight as seen in Figure 4.4. To help visualize the problem, attach a frame to the character in the illustration. This allows better visualization of the character's orientation relative to the camera's projection plane. The

system of the synthetic camera is given as u, v and n. Screen coordinates are given as u and v. And the unknown depth coordinate is n. In Fig. 4.4.a the character's frame is coincident with that of the synthetic camera. In Fig. 4.4. b the character is seen as it appears from the view plane of the camera. Since the character's frame is coincident with that of the camera, no foreshortening occurs. In this situation, the joint angles extracted will be correct. In Fig. 4.4.c the character's orientation indicates a 45° pitch rotation relative to the frame of the synthetic camera. Here the character's system is not coincident with that of the camera. In Fig. 4.4.d the character is seen as it appears from the view plane of the camera. In this situation, foreshortening does occur because the systems are not coincident. The image needs to be projected back into the view plane of the synthetic camera, i.e. the uv plane, otherwise the extracted joint angles will be incorrect. In other words, convert Fig. 4.4.d to Fig. 4.4.b.

4.2 The Foreshortening Solution

Fortunately, the problem can be corrected and the image can be projected into our camera view. To fix this problem, project the line segments, the 'stick' figure, back into the desired view plane. Once this is accomplished, then the correct angles can be derived for link orientations.

Initially, the user enters an estimation of the figure's orientation. This can be accomplished by rotating a cube to indicate the character's apparent orientation, see Figure 4.5. All adjustments must be stored in order on a stack, so that a transform matrix can later be constructed.

The angles of rotation that relate the character's frame to that of the synthetic camera are given here as λ, β and α. Where λ is the u-axis rotation, β is the v-axis rotation and α is the n axis rotation. First estimate λ, β and α in order to project the

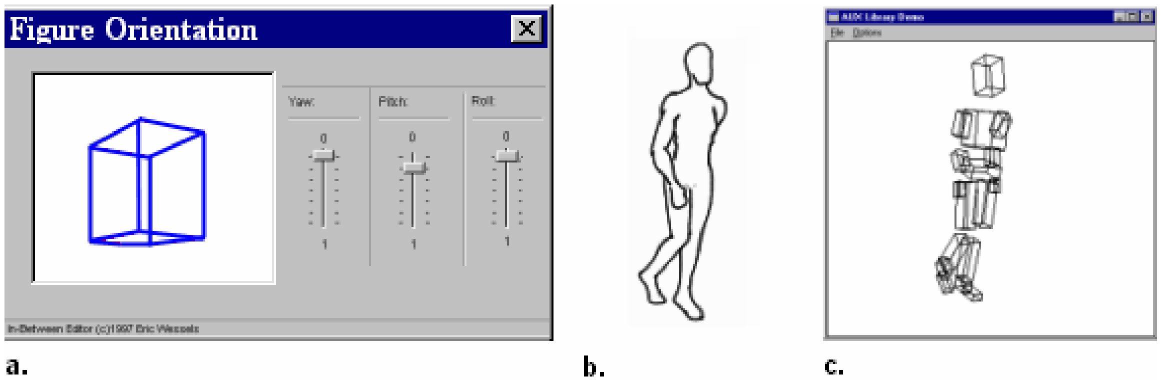


Figure 4.5 Correcting Foreshortening. User orients cube with slider controls (a) to match the character orientation in the original sketch in (b). In (c) is the fully configured character.

image of the character into the camera view plane in a way that makes the character's path of motion appear to be parallel to the camera plane. In other words, the character is rotated into the uv plane.

There are two cases to consider in solving the problem. In the first case, either γ or β equals zero. This is the trivial case since the unknown n coordinate is not needed, u and v are derived from the image. In this simple case, the screen point <u, v> is re-projected with the transform below, where either λ or β is zero. Because the cosine of zero equals one, the following transform stretches the character's image along a single axis.

$\mathrm{Rot2D}(-\alpha, \langle u, v \rangle)$ /* where α is the n axis rotation */

$u' = u \cdot (1/\cos(\beta))$ /* screen u, where β is the v axis rotation */

$v' = v \cdot (1/\cos(\gamma))$ /* screen v, where γ is the u axis rotation */

In the second case when both γ and β are nonzero, solve for n. Start by creating transform matrix M consisting of negative user rotations. As mentioned above, these rotations are placed on a stack as the user orients the cube. Then, iteratively solve for n. At each iteration, construct a vector made up of the screen u and v and a next guess for n. Then, multiply the vector by M and check if the n component of the vector equals zero. If it does, then the segment has been successfully rotated back into the uv plane. [The algorithm is given in pseudo-code in Appendix C.] Once the stick figure is re-projected, the extracted angles can be checked for correctness by comparing the articulated figure against the image.

4.3 Deriving motion from the character itself

In gait motion, an articulated figure doesn't move forward at a constant velocity. To do so, will typically result in unconvincing motion, most noticeably an undesirable sliding of the support leg. To illustrate a correct behavior, Figure 4.6 shows a character taking a step. The various diagrams in the Figure 4.6 illustrate how to recreate gait movement by examining the character itself. Our figure begins with his feet a stride-length apart, right leg forward and left leg behind. The right leg is supporting the weight and is fixed in position. This leaves the left leg free to swing forward. As the left leg crosses over the right, the stance straightens, but the character covers very little distance. Then, as the left leg steps down, the character falls forward. The results from the shift in weight of the right leg to the left. At this moment, the forward velocity reaches its peak during this step sequence.

After the link structure has been configured for orientation, gait movement can be recreated by extracting motion parameters from the character itself, as seen in Figure 4.6. In Fig. 4.6.a a character is taking a step. In Fig. 4.6.b a diagram which describes the motion path of the character's foot during a walk is seen. In the support phase, the foot drags along the ground as the character moves forward. In the transfer phase, the leg follows an arc path as it crosses over the support leg. In Fig. 4.6.c the motion parameters that can be derived from the character's poses are seen. The $\mathbf{y_i}$ values represent the displacement from the object's origin to the floor at each keyframe. The character's forward progress is measured as displacements between keyframes. For example, the variable $\mathbf{\Delta s_{1\text{-}2}}$ is the displacement between keyframe one and two. $\mathbf{\Delta s_i}$ versus $\mathbf{s_i}$ is used because the accumulated distance versus a position measurement is desired. Note that $\mathbf{\Delta s_{2\text{-}3}}$ is larger than $\mathbf{\Delta s_{1\text{-}2}}$ demonstrating that velocity is not constant and therefore a curve is needed to represent it. Now, a velocity curve should be created to capture the behavior by measuring the displacement of the character's support leg between keyframes along a principle axis.

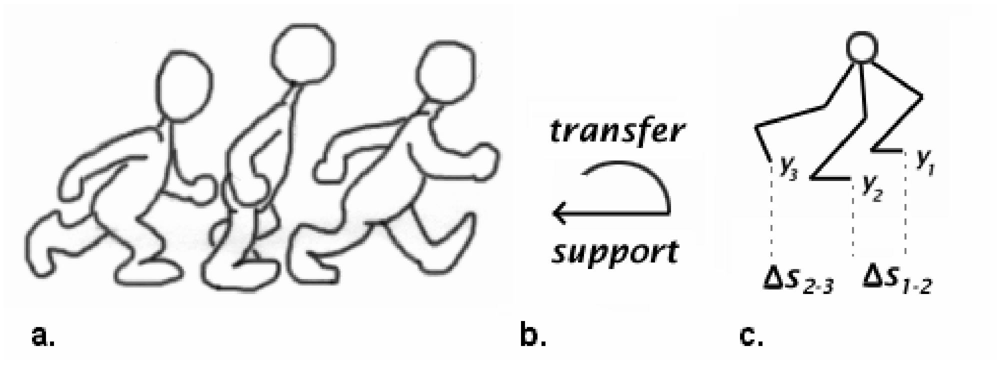


Figure 4.6 Recreating Gait Movement From Character Sketches

4.4 Path Generation

This section describes how to correctly map a character's motion to a trajectory path specified by the animator. The technique described here works best when the character's initial motion is simple. A mapping must be created between the character's velocity **V(t)** and a path given as **Q(u),** for as many cycles as needed to transverse the path. The velocity curve simply measures the change in a character's position from one keyframe to the next. Since **V(t)** represents distance traveled over time, it directly corresponds to the arclength **s(u)** of the trajectory curve **Q(u)**; **s(u)** is given below.

$$\mathbf{s(u)} = \int_{u_0}^{u} \left[\left(\frac{dx}{du}\right)^2 + \left(\frac{dy}{du}\right)^2 + \left(\frac{dz}{du}\right)^2 \right]^{1/2} du$$

The way to solve **s(u)** is with the forward differencing method given in the equation below. The forward differencing method finds the length of **Q(u)** by moving along the curve in small increments which are added up. The total of these increments for time **u** is given by the arclength function **s(u)**.

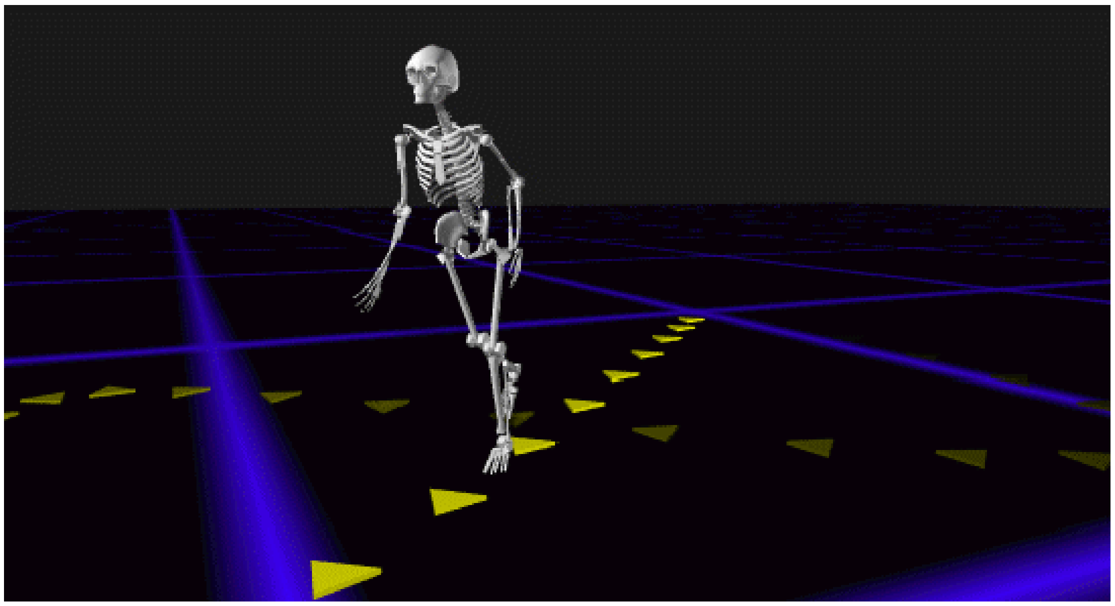


Figure 4.7 Motion Path

$$s(u) = \sum_{j=1}^{i} d_j = | Q(u_j)-Q(u_{j-1}) |$$

Iteration along **Q(u)** accumulates arclength **s(u)**. The arclength **s(u)** relates to **V(t)** because **V(t)** is also an accumulator, namely it is the distance traveled by the character. If the total length of **s(u)** is greater than the total length of **V(t)**, then the character's motion will map multiple times onto **Q(u)** and thus the motion must be cyclical.

Based on how many cycles the character has already completed and the current value of **V(t),** the world frame position of the character, called **x(t),** can be found rather easily. To find the new character position, given the trajectory curve **Q(u)** and the original character position **x(t)**, solve for the curve parameter **u**. This equation takes on the form of $\mathbf{u} = \mathbf{A^{-1}(x(t))}$, which unfortunately is not an analytical function. Thus, a numerical integration routine is needed. The routine is simple enough to implement using the forward differencing method, however it is potentially costly for real-time solutions.

An alternative to an integration solution would be to use a mapping. In chordlength reparameterization [16], a mapping is created so that the knot spacing is proportional to the distance between points. In other words, $\mathbf{Q(u)}$ is reparameterized so that the knot values are evenly spaced. This requires creating a curve which maps $\mathbf{u}$ to $\mathbf{x(t)}$. For this technique to work well, knots and control points are proliferated so that the velocity is roughly constant along the portions of the curve between knots. Fortunately, this can be implemented with a B-spline and yields good results.

As a final step, footprint generation should be provided to allow the user to evaluate the result. The location of each footprint can easily be calculated from the data by looking for ground plane contact with the foot at the keyframes. The clock values and positions where penetration occurs should be saved. To find unwanted foot penetration from interpolation, the same technique is used but iteration along the clock is in smaller time-steps. The mapping of $\mathbf{V(t)}$ to $\mathbf{Q(u)}$ can then be used to determine the location of footprints along $\mathbf{Q(u)}$. The orientation of the footprints can be derived by computing the tangent curve of $\mathbf{Q(u)}$.

Chapter 5

Motion Editing

As mentioned in the introduction, the reuse of previously generated motion is one of the leading areas in computer graphics research. Due to the complexity of computer animation, considerable time and expense is invested in creating quality motion. Ideally, this motion could be adapted to generate similar motions without repeating the entire process. This chapter covers various motion-editing tools written for the purpose of adapting and enhancing previously generated motion. The tools relate to time editing, fine-tuning transitions and operations on motion curve data. They don't attempt to redefine the motion task. Instead, their primary purpose is for motion enhancement. They modify the 'look' of how a task is performed rather than modify the task itself. Their primary focus is to provide an equivalent to a traditional animation technique called spacing charts, described in the next section.

5.0 Overview

The animator can use the software described here to create variations from a previously generated motion. The desired effects may be to add personality, model the effects of weight, create sharper movements, or simply exaggerate motion. For example, imagine that the animator wishes to create an animation of a female lounge singer who sways as she walks across the stage. He can improve the caricature by exaggerating hip movement and modifying timing so that she appears to bounce each time she swings her hips.

5.1 Mimicking Spacing Charts with a Timing Curve

With regard to acting, timing has been referred to as the essence of the art. By specifying timing information the animator can enhance a motion for dramatic effect. There are times when a character should freeze in his tracks or jump out of a pose. Timing gives life to the character by giving it personality. Traditional animators rely on instinct to know how the character should act. They have their own technique to describe exact speed over the dozens of film cells that they must create. They use spacing charts.

Spacing charts are simply time-position patterns for actions. Typically, the traditional animator has a specific action in mind along with the number of frames the action should take. He not only encompasses speed but also acceleration. Additionally, charts may be required for individual limbs to model secondary and overlapping actions.

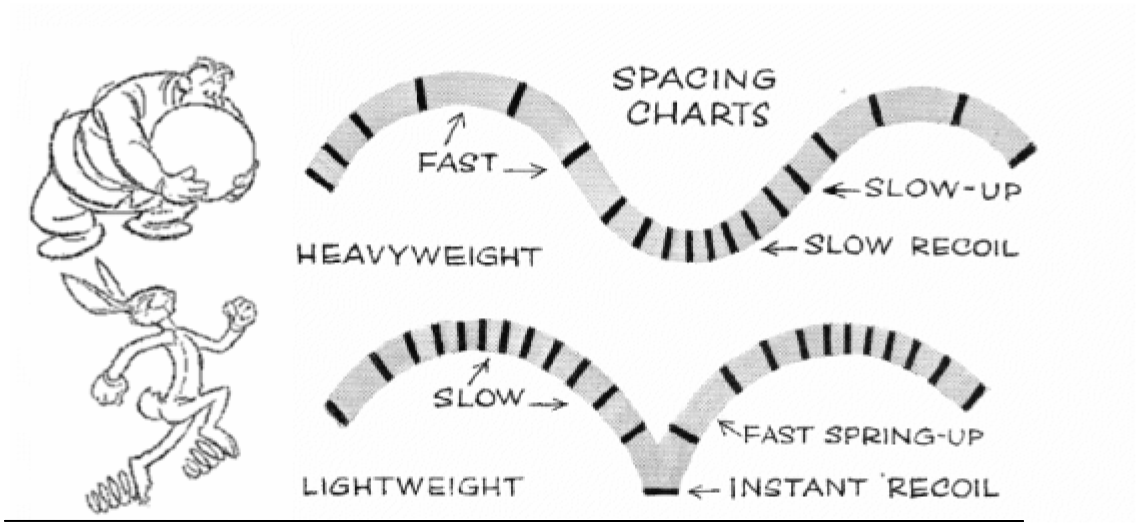


Figure 5.1 Spacing Charts

Besides personality and dramatic effect, spacing charts can be used to model the effects of weight. In actions such as walking, running and jumping, time can be accentuated at key points in the motion to model weight. When time is accentuated at the recoil phase, it will appear that the character is heavy and cannot seem to get off the ground. If the transition phase is accentuated, the opposite effect is created. The character seems to float.

Clearly, a level of control similar to spacing charts would be invaluable to the computer animator. A timing curve can be created to help model spacing charts. This curve is used to reparameterize the motion curves that collectively describe the overall motion. Timing curves can be modeled with various forms from S-curves to cubic B-splines. In early work, S-curves were commonly used for simple transitions such as slow-in/slow-out. The more flexible Cubic B-spline form allows for more complicated mappings. Badler et al [14] describe a reparametrization technique using B-splines for scripting. In their approach keyframes are mapped to specific time values.

There are two requirements for the timing curve in this system. First, the input needs to be simple enough so that the animator can enter timing information through a simple interface. Since the animator's input is interactive and heuristic, Badler's method is not appropriate. Exact time values are not known. The second requirement is that the timing information entered for one keyframe will not greatly affect the overall shape of the curve beyond the neighboring keyframes. In other words, localized editing should truly be local. In Badler's method, one large interval will greatly distort the shape of the curve.

In this system, the following method is used to create the timing curve. For each interval between two keyframes, a control point is initialized to a slope of one. This gives an initial identity mapping where the output clock (the spline value) is nearly identical to the input clock (the parameter). To increase the speed at a particular interval the slope is increased and the curve is re-normalized. In contrast, the slope is decreased to slow down the clock. Because each interval has a limited effect on the overall length of the curve, no one interval can dominate the sequence clock. The strength of this method is the input is simple and well behaved. As will be seen below, this simplifies the interface used by the animator.

The layout of the interface for creating a timing curve can be fairly simple. The user can adjust one control to select the desired keyframe. Then, another control can be adjusted to increase or decrease the slope of the timing curve at the keyframe. Decreasing the slope of the timing curve achieves the effect of making the sequence appear to decelerate at the current keyframe. Alternatively, increasing the slope of the timing curve achieves the effect of making the sequence appear to accelerate at the current keyframe.

The cubic B-spline form was used to model this curve. One problem that remained was ensuring that acceleration and deceleration occurred at the precise moment that the animator intended. The cubic B-spline form was used to reparameterize the motion curves, but it became apparent that a second curve was needed to reparameterize the timing curve. These forms are often used to represent motion curves because of their C^2 continuity that results in smoother curves. A drawback of this form is that the spline doesn't actually pass through control point values. As a result, the time at which acceleration and deceleration occurs is only approximate. To improve the precision of the timing curve a second curve must be employed to control the knot spacing on the timing curve. While this improves the precision of the timing curve, this type of information is difficult to specify in an intuitive manner.

A technique for graphically defining knot spacing using a plotted graph is discussed next. While this technique may be intuitive for a computer scientist, it is unlikely to be appealing to computer animators. One of the goals of future work will be to improve the user-friendliness of the interface with respect to specifying knot spacing.

In the graph, see Figure 5.2, boxes are plotted along the timing curve to represent the knot values at the keyframes. These boxes are labeled with the number of the keyframe which they represent. The animator can look at the placement of the knots relative to the shape of the curve and determine if the object is accelerating at the correct time. If it is not, then he can adjust the knot placement up and down along the curve to indicate the desired position of the keyframe knot. The input is simple and can be done interactively with a mouse. However, the visualization is mathematical and needs to be improved for the animator. For example, assume the animator wants the motion to decelerate as it enters keyframe two, then accelerate as it leaves keyframe two and enters keyframe three. First, the user will select the keyframe and reduce the slope using a graphical control. Then he will examine the plotted curve. To accomplish what he wanted to do, he will position the box (knot) for keyframe two into the portion of the curve with the lowest slope between keyframe two and keyframe three. The effect will be that the motion decelerates as it enters keyframe two and accelerates as it leaves it.

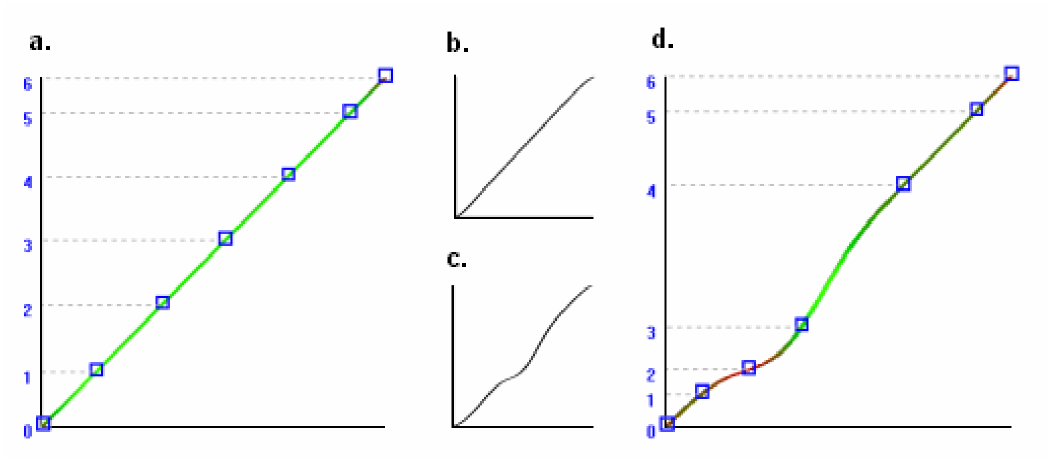


Figure 5.2 The Timing Curve. In (a) a default timing curve is seen. In (b) the knot spacing curve for the timing curve is shown. In (d) is the timing curve after the user has decreased speed at keyframe two. In (c) the knot spacing curve for (d) is seen where the user has moved the knot into the slowest portion of the curve.

5.2 Fine Tuning Spatial Transitions

Serendipitously, the technique described above can be used for a purpose other than manipulating sequence timing. It can also be used to fine tune limb velocity throughout a sequence. While Cubic B-splines yield smooth transitions, they do have a very visible effect on the motion as a whole. Since knot values don't exactly match the original control point values, the peak amplitudes of the motion curves are noticeably reduced. The results in animation are mostly noticeable in quick movements where characters seem to move in a sluggish manner. For example, the character may appear to casually swing his arm when he is supposed to throw a punch. To some extent, this effect can be counteracted by re-normalizing the motion curves until the knot values are closer to the original control point values. However, it doesn't completely counter the effects. Additionally, re-normalizing all control points may adversely effect in-between frames.

An alternate, more localized solution can be used. Instead or re-normalizing the whole curve, the curve can be re-normalized only for critical control points. From there velocity can be modified for the in-betweens. The modification of velocity is done at strategic points where we wish to move from one knot value to a next very quickly. The difference between this and a timing curve is that we do not wish to speed up the character's clock, only specific spatial transitions.

This type of precise fine tuning is most beneficial for motion correction. However, this technique does not directly correlate to any existing idea in traditional animation. Thus, it will be difficult to implement this tool in such a way that the animator would be encouraged to use it. The best implementation of this technique would be in an algorithm designed to look for problems in the interpolated motion.

5.3 Exaggerating Motion

The final editing feature implemented by this system simply multiplies the spline value by a scalar at each time-step. This operation provides a simple way to exaggerate or reduce movement around a particular axis. This can be useful for decreasing stride or increasing hip swing. This simple means of editing can be surprisingly useful when globally applied to all angles around a particular axis. When walking or running in gait motion, the movement of the hips triggers a chain balance reaction which tilts the chest, shoulders and head. As a result, this simple form of multiple-link editing produces good results. For the same reasons, the overall stride of the gait can be modified in a similar fashion. Eventually, all motion editing will be grouped into a single application specifically designed to visualize motion data. The challenge will be to visualize data in such a way that it is understandable by the animator.

Chapter 6

Conclusion

Creating computer-based tools usable by the animator is challenging. The approach presented in this thesis demonstrated that believable character motion can be accomplished by building off traditional animation techniques already familiar to the animator, namely rotoscoping and spacing charts. In addition, these computer equivalents offer advantages to pre-existing computer-based techniques. Some of the advantages of computer-based rotoscoping are that it offers an improvement to low-level parametric keyframing in that humanoid motion can be described more quickly and with more accuracy. It is more attractive than hybrid kinematics-dynamics systems because the required input for the software is easy to understand and relates to an established traditional animation technique. And finally, the tools needed for rotoscoping are generally available and could potentially be used by anyone with a video camera and a computer. The advantage of working with spacing charts is that a precise description of timing can greatly enhance a motion. Editing tools which mimic spacing charts are provided to allow the user to attain the level of refinement desired in a finished product.

The implementation of spacing charts is unique because it focuses on the precise re-parameterization of motion for purely heuristic purposes. An example of such a purpose would be to endow the character with personality.

Television viewers can frequently watch programs which are dedicated to presenting the labor-intensive, tedious work involved in producing a traditional or computer-based animated film. These programs cause one to wonder why so much time and the efforts of so many are necessary to accomplish such a task. Hopefully, the approach presented here is a viable shortcut to some of this activity. Nevertheless, developing the system and performing the mechanics necessary to operate and test it demonstrate the need for future improvement. Primarily, these improvements should focus on diminishing labor intensive activities.

The approach described in this thesis can greatly reduce the time needed to create believable character motion. However, the implemented system can benefit from improvements in some critical areas. The process of bringing live video into the computer could be better streamlined. Computing solutions for link configurations should be sped up so the animator does not have to wait. And the amount of data entry could be reduced by taking advantage of spatial locality.

The process of capturing, importing and converting video data into a suitable format is laborious. To begin, cameras need to be positioned and oriented properly to ensure good results. After recording the motion, the camcorders' footage must transferred to a conventional VCR recording deck. Then, the video is imported into the computer with a video capturing device. Unfortunately, low-end video capturing devices record the videostreams in an uncompressed format, making the files large. Thus, it is necessary to convert each stream to an 8-bit compressed format with a commercial codec to save space. Next, the streams need to be synchronized before exporting the frames as individual images readable by the system. Because codecs typically average frames for improved compression, the camera flash used during the recording session for the purpose of synchronization can be lost. If this occurs, finding the correct starting points in the streams is needed so that the video can be exported as sequences of images. Clearly, simplifying this to a one or two-step process would be helpful.

Another aspect subject to improvement is the process of extracting the motion from the images in the three-dimensional rotoscoping system. As described in chapter 3, the user must enter stick figures to approximate the actor's pose and then wait while the software calculates a solution. The attractiveness of the system would be greatly improved if the time associated with both these actions could be reduced. For example, a ten-second sequence of a complex motion, e.g., a martial arts one-step, can take up to two hours of user time. If we assume a sample rate of five keyframes per second, the configuration for the entire sequence will require an hour of computation time due to the iterative solution. Users typically want real-time results.

With regard to extracting motion from the images in the three-dimensional rotoscoping system, there may still be an analytical solution that has been overlooked. As mentioned in chapter 3, the analytical solution doesn’t work because the technique mixes combinations of rotations along the link structure. When solving for rotations, the line segment is always rotated into a particular axis based on the angles being sought. The problem appears to arise because the link segment doesn’t represent the same axis from one link to the next. But perhaps, an intermediary coordinate transform may fix the analytical solution. This could make the solution real-time.

Finally, drawing stick figures with a mouse can easily consume one hour, given fifty keyframes. Drawing the stick figure in each view results in one hundred stick figures total. Because the actor’s body changes little from one keyframe to the next, it may be advantageous to allow the user to enter a stick figure for every five or so keyframes and then let the software interpolate the in-betweens. The user could then adjust the in-betweens, so they will be correctly posed. For the parts of the actor which are moving slowly, this could dramatically reduce the amount of input required by the animator. If the animator’s work for fifty keyframes could be reduced from two hours to 20-30 minutes, the attractiveness of this rotoscoping approach would significantly improve.

Motion capture using specialized equipment, i.e., electromagnetic trackers, will always be more accurate than the rotoscoping techniques described here. However, the low-cost nature of this system makes it an attractive alternative for animators working for smaller companies with modest budgets. Other techniques presented here for editing motion proved workable but require more development to further ease usage. For example, the most useful editing tool, to imitate spacing charts, offers a high-level of precision but needs a less intimidating interface for non-technical users. Perhaps the best argument for the overall utility of this system is that it produces good results. And while some tools in the system still need to be refined for usage, the principles behind most of the tools and the input they accept is straight forward.

Appendix A

Frames of Reference

A.0 Description

The terms coordinate system and frame are fairly interchangeable. Imagine a set of axes permanently attached to an object so that when the object moves the axes move with it. In a Cartesian coordinate system, positions are specified along the x, y & z axes with the point <0,0,0> referring to the origin of the system. The whole constitutes a frame of reference to track the motion of various objects. Often when an object moves it is easiest to keep track of what is going on by following the motion of the frame of reference attached to the object.

Probably the most common use of this construct is for coordinate transforms. Coordinate transforms are used to relate the coordinates of vertices from one frame of reference to another. For example, in a flight simulator it is necessary to constantly transform objects into the world coordinate system to coincide with the view of the

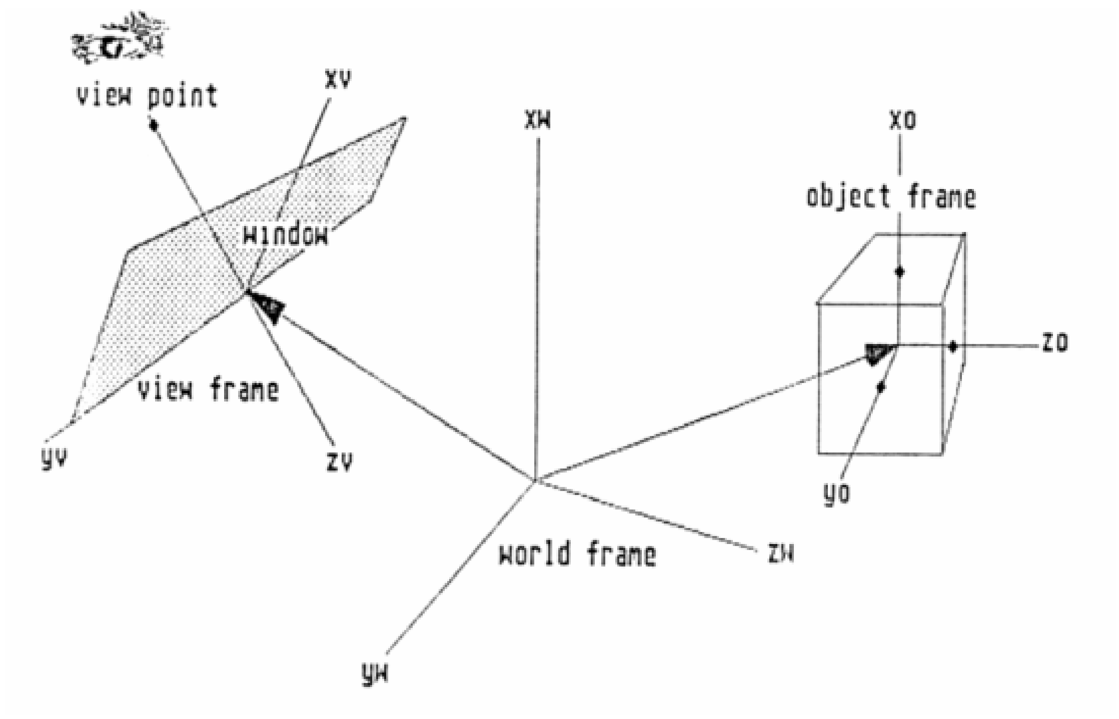


Figure A.1 Frames of Reference

cockpit. This would require a coordinate transform between the observer's frame and the world frame. By contrast, the other type of transform, called a geometric transform, usually describes what happens when the object itself is moved inside a single reference frame, e.g. a cube spinning around it's center.

In the rotoscoping system described in this paper, there are two top-level frames of reference. First observe the actor perform in a space called the view frame. This coordinate system is the real world it is actually constructed from the viewpoints of two cameras at right angles.

The other frame is the world frame. This the top-level frame of reference for all the objects in the world. In this hiearchial construct, each object has it's own frame called an object frame. Since our objects are link structures, the hierarchy proliferates with individual link frames.

These two top-level frames are treated independently. There is no predefined transform which shall correlate coordinates of vertices from one frame to the other. Fortunately, the only time when there is a need to correlate coordinates between the two frames is when the position of an object in the world frame must be determined from the camera footage. In this case, the coordinate from the view frame will have to be normalized into the world frame.

A.1 The View Frame

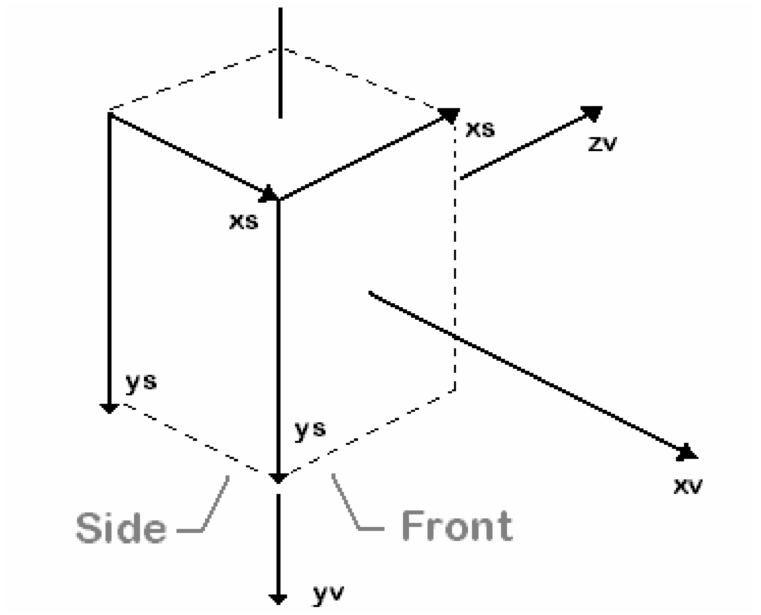


*Figure A.2 The View System***.** The View Frame is constructed by combining coordinates from the camera view planes, called Window Frames.

Typically in 3D graphics, objects are brought to the screen through a series of coordinate transforms. Objects are placed in the world coordinate system (world frame) and transformed to the view of the world as seen by the user. This reference frame of the observer is typically called the view frame. To get from the view frame to the screen, make a projection onto a plane, called the view-plane, of the object which is to be displayed.

In a rotoscoping system the opposite process must be followed, the view-planes go to the view frame. Begin with two cameras set up at right angles to each other and aimed at a common focal point. This allows two fixed views of the real world. The contents of these views may change as a function of time, but the position and direction of each camera is constant. Now track movement in each view relative to a fixed coordinate system.

A.1.1 The Window Frame

Since the animator works with these views in a raster display, these fixed 2D coordinate systems are referred to as window frames. Designate one as ‘side’ and the other as ‘front’. A window frame relates to the fourth quadrant in a Cartesian plane system where all coordinates are positive.

A.1.2 Constructing the View Frame

In order to construct a view frame, a method of combining window frame coordinates to create view frame coordinates must be used. In other words, combine the coordinates of a vertex in one window frame with those of the other in such a way as to yield a correct view frame coordinate. Fortunately, the right angle relationship between the two window frames makes this easy. For every vertex the user enters in the ‘front’ window frame, a corresponding point is specified in the 'side' window frame. The view frame coordinate is simply (X_{front}, $(Y_{front}+Y_{side}) / 2$, X_{side}). The view frame is clearly an imaginary construct but it plays an important role as a frame of reference. In Figure A.2, the relationships between the window frames and the view frame is illustrated.

A.2 The World Frame

As mentioned before, it is convenient to track the motion of objects by comparing their frames of reference to a global system. This common space, inhabited by all the objects, is called the world frame. It is important that all objects are transformed into this common space in order that their spatial relationships may be defined.

A.3 The Object Frame

Each object in the world frame has it's own local coordinate system called an object frame. The process of transforming coordinates of vertices from an object frame to the world frame is known as the object-to-world transform. Since the motion of human-like characters must be described, a hiearchial structure which consists of many smaller objects is needed.

A.4 The Link Frames

An articulated figure is a structure which consists of a series of rigid links connected by joints. The link structure is represented as a whole with an object frame.

The smaller component objects are link-joint instances, referred to here as links. Each link has it’s own coordinate system, typically centered at the joint, called the link frame.

A.4.1 Degrees of Freedom

An articulated figure greatly differs from a simple rigid body in the amount of information required to describe the object, as well as how this information is specified. An unconstrained rigid body has six degrees of freedom, three translational and three rotational. In an articulated figure where the rigid links are fixed, or ‘glued’, at the joints (translational constraints), only the three rotational degrees of freedom are considered. The number of degrees of freedom of the entire link structure is the number of independent variables necessary to specify the state, also called the configuration, of the figure. For example, to represent one keyframe of a man running may require 51 different angles of rotation to be specified in the link structure. The configuration of the articulated figure could be represented in state space with a 54 element state space vector, 51 elements for rotation plus an additional 3 elements to translate the object in the world frame.

A.4.2 Transformation Matrix

Constructing a transformation matrix for each link is also more involved than that for a simple rigid body. The transformation matrix for a particular link is the concatenation of all matrices along the path from the top of the link structure to that link. Because transformations are cumulative, the structure must be transformed in a top-down fashion using transforms which relate the frame of reference of the current link to that of the one immediately proceeding it in the tree. Thus, the transformation of a particular link is in fact a series of coordinate transforms starting at the top of the link structure and ending at the link. It is important to remember that the order of matrix multiplication is critical since the rotational components of the transforms are non-commutative. This representation is both intuitive and conducive for 'gluing' of geometric data to the link skeleton which must be done later.

Appendix B

Rotations

B.1 Schemes of Rotations

In discussing the configuration of the articulated figure, two schemes of rotations should be mentioned. One is used when solving for the angles of rotations which relate one frame of reference to the other. The other is used in the link structure as a general description of orientation. Both ways can be used to describe the orientation of a 3D object. And both use rotations relative to the base vectors of an object's frame. The major difference between the two schemes is that one uses two angles to represent orientation and the other three.

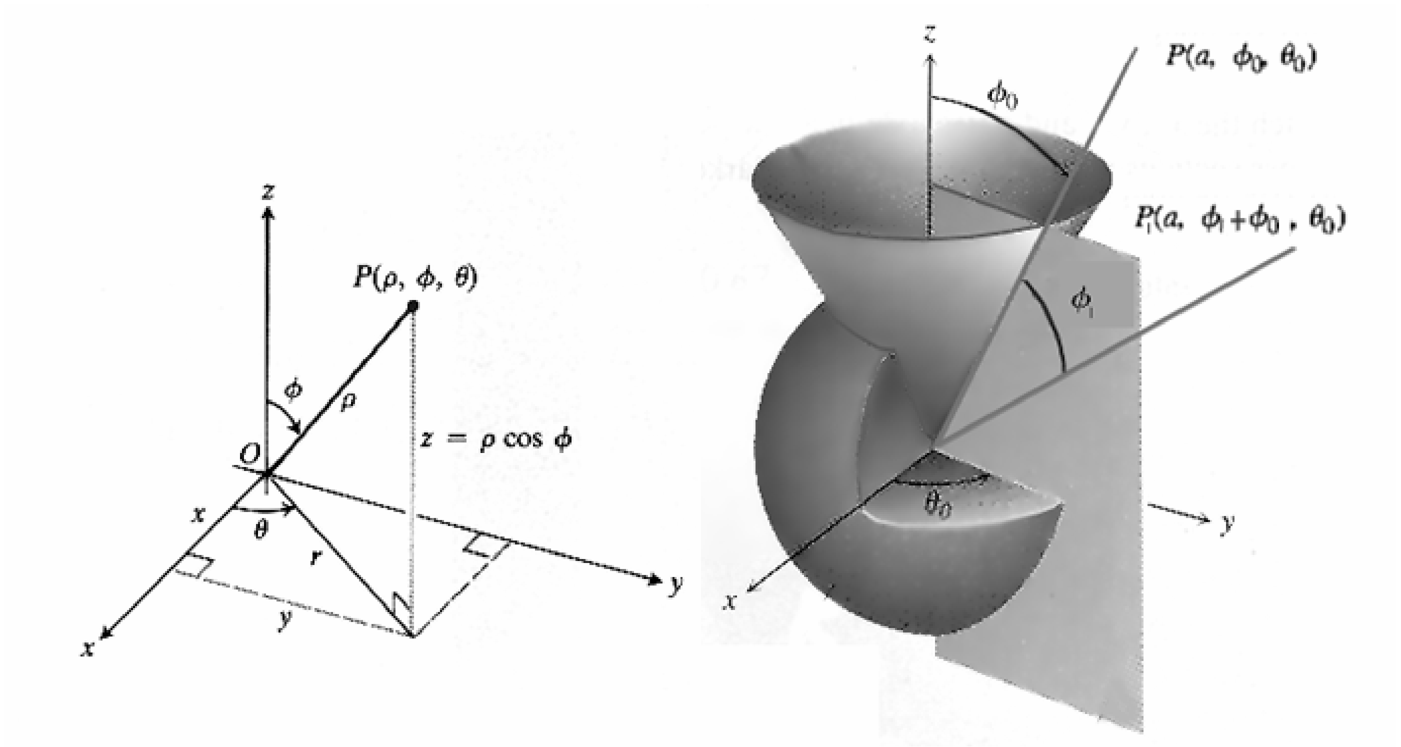


Figure B.1 Spherical Coordinates. Incremental rotation from ϕ_0 to ϕ_1. The total rotation is represented with a single angle ϕ_{0+1}. Thus, the angles of rotation, ϕ and θ, provide a general recipe of orientation.

B.2 Spherical Coordinates

One scheme to implement rotations is similar to that involving lines of longitude and latitude, where rotations about two axes are added up separately and finally put together at the end. In this scheme, several incremental rotations can be made both left to right (latitude) and up and down (longitude) in any order, but only the separate totals are recorded.

As an example, suppose the total rotation about the y-axis is 40° and the total x-axis rotations is 83°. Then, the overall rotation is taken to be a single rotation about the y-axis of 40° followed by a single rotations about the x-axis of 83°. Note that this isn't the same as rotating about the x-axis first and then the y-axis second which gives a different result.

Doing a rotation about the first axis, followed by a rotation about the second axis, does provide a recipe for always getting to the same orientation each time. This is just like finding a position on the globe uniquely with circles of longitude and latitude. The rotation abut the first axis gives the angle of latitude, and the rotation about the second axis gives the angle of longitude. This results in a simple scheme to uniquely orient an object, but is restrictive for a link structure as a whole because it ignores rotations about a third axis.

B.3 Euler Angles

Euler angles are a way of specifying the orientation of one reference frame to another using three angles. An important aspect of rotations is they must be specified about different axes in a fixed order. There are many combinations possible. The sequence commonly used by aeronautical engineers is called the 321 sequence because it describes rotations about the x, y and z axes. These rotations are called yaw, pitch and roll. See Figures B.2 and B.3.

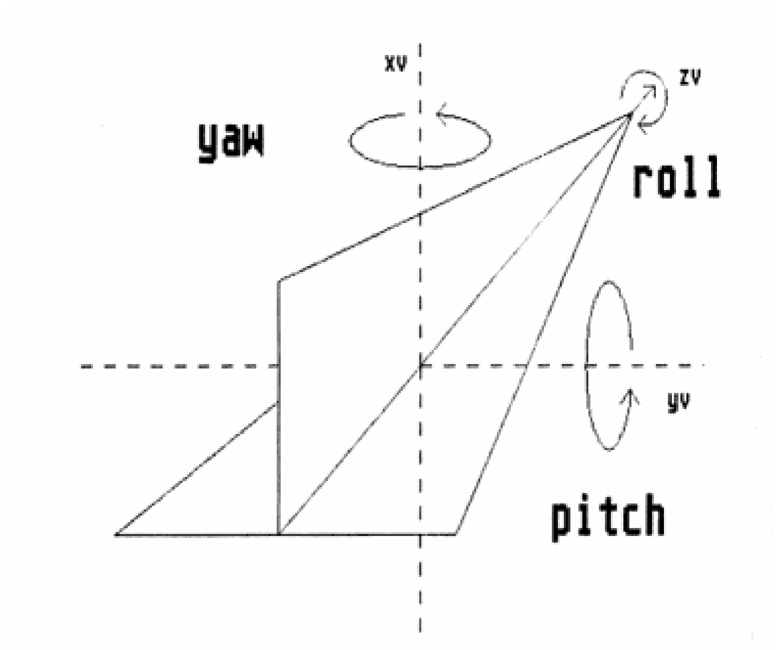


Figure B.2 Aeronautical Terms

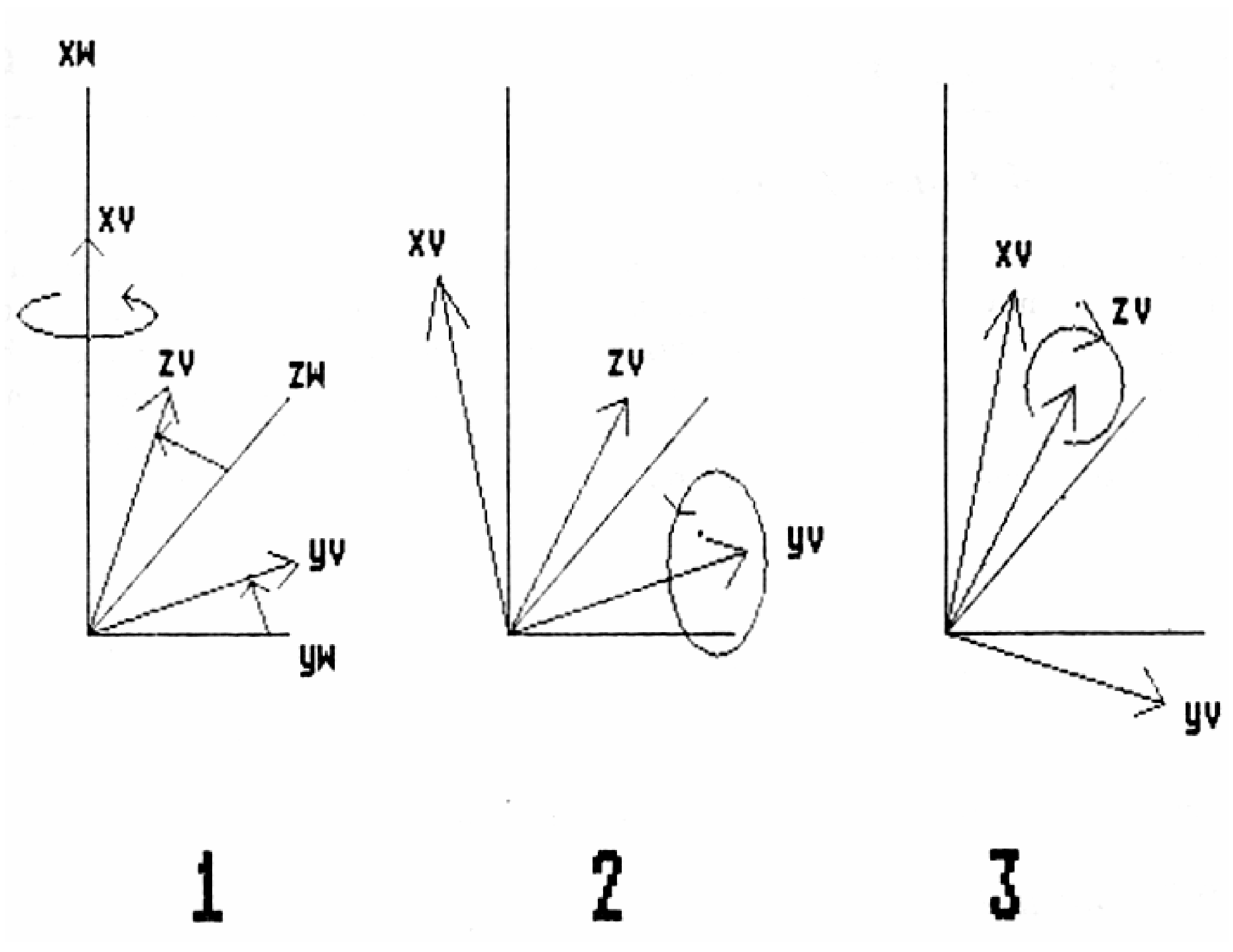
A typical sequence of rotations which carry one reference frame into another is given below.

1. rotate by θ about the x-axis (yaw) .
2. rotate by ϕ about the y-axis (pitch).
3. rotate by γ about the z-axis (roll).

The sequence actually used by this system is 312. Because the user enters line segments to describe orientation, it is hard to derive pitch for some body parts. For example, the animator is likely to enter a vertical line segment to represent the chest in both views, thus giving no clue to the pitch orientation. For this reason, the pitch orientation is often specified in the window in the three-dimensional rotoscoping software. So make pitch the first rotation in any control matrix .

Euler angles are different from the other scheme in that one orientation is not uniquely described by three angles. There are many possible combinations. When describing link orientation it is more flexible, but when solving for rotations the other scheme must be used .

Figure B.3 Rotation Sequence for Euler Angles



Appendix C

```
/* Pseudo-code algorithm for returning angle from line segment  */
/* camera system is u, v, and n, figure orientation is given by */
/* angular displacements of γ, β, and α.                        */

/* project point (dir vector) into orthographic plane */
Get_Angle( u, v, n, γ, β, α)
Begin

  if α=0 and β=0 then                /* trivial case */

        Rot2D( -α, <u,v> )
        u = u * ( 1 / cos( β ))      /* screen x */
        v = v * ( 1 / cos( γ ))      /* screen y */

  else                               /* have to solve for unknown n */

        vector P=<0,v,u>                   /* vector space is x,y,z       */
        matrix M                           /* inverse transform matrix    */
        mag=sqrt( u*u + v*v )              /* length of line segment      */

        while ( stack not empty )
                Rot(γ, β, α)=Pop(user_orientation_stack)
                M = M * Rot(γ, β, α)
        end while

        /* in (x,y,z) x is the lost coordinate */
        for x = 0.0 to 1.0 step tolerance  /* tolerance≅.001 */
                P′ = < x, P.y, P.z>
                P′ = P′ · M
                if P′.x = ± tolerance      /* aligned with zy plane? */
                        P = P′ * mag
                        u = P.z
                        v = P.y
                        break;             /* break from for..loop         */
                end if
        end for

        return Line_To_Angle( u, v )       /* use atan2(y,x) in C    */
End
```

References.

[1] Thomas & Johnston, “Disney Animation: The Illusion of Life”, Abbeville Press, New York, ’84.

[2] Lasseter, John, “Principles of Traditional Animation Applied to 3D Computer Animation”, Computer Graphics, Vol. 21, No. 4, ’87.

[3] Gomez, Julian, “Twixt: a 3-D Animation System”, Proceedings of Eurographics 84, ’84.

[4] Sturman, David, “Interactive Keyframe Animation of 3-D Articulated Models”, Proc. Graph. Interface, ’84.

[5] Bruderlin, A. and Calvert, T. “Goal-Directed, Dynamic Animation of Human Walking,” Computer Graphics, Vol. 23, No. 3, pp. 233-242, 1989.

[6] Cohen, M., “Interactive Spacetime Control for Animation,” ACM SIGGRAPH ’92, pp. 293-303, 1992.

[7] Phillips, C., Zhao, J. and Badler, N. “Interactive Real-time Articulated Figure Manipulation Using Multiple Kinematic Constaints,”,ACM SIGGRAPH ’90, pp. 245-250, 1990.

[8] Witkin, A. and Kass, M., “Spacetime Constraints,” ACM SIGGRAPH ’88, pp. 159-168, 1988.

[9] Zeltzer, D. “Motor Control Techniques of Figure Animation,” IEEE Computer Graphics and Applications, Vol. 2, No. 9, pp. 53-59, 1982.

[10] Bruderlin, A. and Williams, L., “Motion Signal Processing,” ACM SIGGRAPH ’95, pp. 97-104, 1995.

[11] Guo, S. An Approach to Computer Keyframe Animation Incorporating Motion Control: Parametric Keyframe Space Interpolation, Ph. D. Thesis, Illinois Institute of technology, 1992.

[12] Guo, S., Roberge, J., and Grace, T. “Controlling Movement Using Parametric Frame Space Interpolation,” Models and Techniques in Computer Animation, Magnenat-Thalmann N., Thalmann D.(eds.), Springer-Verlag, pp. 216-227, 1993.

[13] Witkin, A. and Popovic, Z., “Motion Warping,” ACM SIGGRAPH ‘95, pp. 105-108, 1995.

[14] Steketee, S.N. and Badler, N.I., “Parametric Keyframe Interpolation Incorporating Kinematic Adjustment and Phrasing Control, SIGGRAPH ’85.

[15] Barsky, B. and Beatty, J.C., “Local Control of Bias and
Tension in beta-Splines,” ACM Trans. on Graphics, 109-134.

[16] Guenter, B. and Parent, R., “Computing the Arclength of Parametric Curves,” IEEE Computer Graphics and Applications, May 1990.

[17] Watt, A. and Watt, M., “Advanced Animation and Rendering Techniques Theory and Practice,” ACM Press, New York, ’92

[18] Girard, M. and Maciejewski, A.A., “Computational Modelling for the Computer Animation of Legged Figures, “ SIGGRAPH ‘85, 263-270, 1985.