ARTICULATION CONCERNS FOR SPECIFYING GENERIC CLASS CHARACTER ANIMATION

Martin Hash

BSEE
California State University, Sacramento, CA. 1980

MBA
University of Portland, Portland, OR. 1984

A thesis submitted to the faculty
of the Oregon Graduate Center
in partial fulfillment of the
Master of Science
in
Computer Science

June, 1989

ABSTRACT

ARTICULATION CONCERNS FOR SPECIFYING GENERIC CLASS CHARACTER ANIMATION

Martin Hash
Oregon Graduate Center, 1989

Supervising Professor: James R. Goecks

Articulation tools are required by animators for creating the motion assumed by computer generated characters. The global concept of specifying hierarchial articulated motion is termed a "scene editor" in this paper, and allows a traditional animator to specify reusable character actions across a "Class" of characters. Different character dimensions are handled invisibly to the user. The hierarchial nature of connected body parts is treated as an integral part of the process and poses special problems and advantages.

The thesis, "Articulation Concerns For Specifying Generic Class Character Animation", by Martin Hash has been examined and approved by the following Examination Committee:

James R. Goecks, Thesis Research Advisor
Adjunct Assistant Professor,
Dept. of Computer Science and Engineering

Richard B. Kieburtz
Professor,
Dept. of Computer Science and Engineering

Robert G. Babb II
Associate Professor
Dept. of Computer Science and Engineering

TABLE OF CONTENTS

1. INTRODUCTION

The seed idea for this project germinated in 1983 as a result of my introduction to the crude computer graphics of the Commodore VIC 20 home computer. At that time, computer animation was a rough, two-dimensional affair, run with too little memory in too much time. Commercial use of computer graphics was starting to mature, but the major thrust of such efforts was mathematically based. The narrow range of things these first "computer animation systems" could do were exploited almost beyond their ability to recover from the stereotype they engendered. Character animation, what some people call "Disney-style" animation [21], was almost totally ignored and the few major projects doing work in this area were far beyond the knowledge or means of the traditional artist. The thrust of my interest lay in character animation because with it someone could actually tell a story. Computer animation, in my view, is not an end in itself, but instead a means for the creativity of people to be loosed [35]. The computer should take over the mechanical portion of producing an animated story; the artist/writer should provide the creativity.

The kind of animation we are concerned with fits the traditional definition of "character animation", a very powerful story telling medium. The treatise covered here is the application of the computer to character animation [6 7 8 9 10 13]. There are three general categories of computer graphics for the artist: stills, two-dimensional animation, and three-dimensional animation.

"Saturday-morning" character animation is perceived as being two-dimensional, but in reality is three-dimensional in scope: as a character turns towards the camera, new topographical information is presented [2]. So-called "2-1/2D" animation exists where 2D characters can overlap each other but no topographical change occurs. I think the true distinction of three-dimensional character animation is topographical change, whether or not shading exists. Three-dimensional character animation therefore requires articulation of movement in three dimensions.

A computer animation program can be divided into three distinct parts: a "scene editor", an "object editor", and a "renderer". Much work has been devoted to the final phase of this triad because it is the "eye-candy" that motivates computer art [12]. Object editors have received more attention in the academic environment where it is recognized that being able to produce any object in a standardized manner is desirable. However, relatively little effort has been expended on a scene editor, which provides movement articulation and is the phase used by the animator to specify the scene's activity [18]. Other scene editors have been developed [11], but this is the first to be implemented fully on a microcomputer. It is also the first to utilize "figure substitution" and "libraried actions".

This paper explains the concept of a "scene editor", and explicitly describes one which I have fully implemented in a commercial product.

Contrary to existing computer animation systems where every object's motion is explicitly defined, I think object activity should be implicit and independent of the object representation so that the defining of a "Choreograph" is an abstraction. The Choreograph should be able to combine different kinds of objects and activities with most differences invisible to the animator.

1.1 TERMINOLOGY

A Bone is the fundamental component of a single non-articulated object, independent of representation technique.

A Figure is the hierarchial binary tree construction of Bones that represents articulated objects.

A Class is a group of Figures with the same hierarchial connectivity.

An Action is a generic movement of any Figure in a Class, and can be libraried for future use by any member of that Class.

A Script is a series of Actions associated with a Figure.

A Character is the hierarchial binary tree construction of Figures. A Character can be as simple as one Figure, or composed of many Figures.

A Choreography is the interrelated activities of all the Characters in a play, plus technical control such as camera motion, light source positioning, and focal length.

A Vignette is a rendered Choreography approximately one half to fifteen seconds in length. A full length animation is composed of hundreds of vignettes.

Objects can be represented in the computer in a variety of ways: [20]

polygonal/triangular meshes [14]

mathematic primitives (CSG) [1 24]

mathematic surfaces ("patches") [15 19]

mathematic phenomenon (fractals) [22]

particle systems (stochastic processes) [4 16]

octrees [23]

explicitly ("hashnique")

others

As with most things, each method has its advantages and disadvantages. The mathematical-based representation techniques predominate and produce most of the computer animation we are familiar with.

2. THE SCENE EDITOR

Three-dimensional computer articulation animation systems, "scene editors", are divided into three categories by Zeltzer: guiding, animator level, and task-level [25]. Guided systems consist of a list of objects with associated lists of transformations. Animator level systems specify movement using algorithms. In task-level systems the behaviors of objects are defined as events and relationships; it is also called "actor-based" animation. My scene editor is a guided system by this definition.

To explain a scene editor we first need to break traditional animation into its component parts. A good artist does not necessarily make a good animator. An animator must understand how things move; how to make twelve slightly different pictures look like natural motion for one second. Once an animator has discovered the leg and arm motions of a particular person running, he can extrapolate that knowledge to any person running. But in traditional animation, only the knowledge can be transferred: there is no mechanism for applying walking motion to any character mechanically. A library of sorts can be formed by an animator so that all an artist need do is design the character, then pick and choose

perform from the library.

The artist would need to know about the library motions in advance so that the character could be specifically designed such that the motions would look natural. This implies that the component parts that a character is made up of must somehow fit into the framework of the motion. For example, if the motion was designed for a man walking and the man has pivots at the elbows and knees, then the new character needs also to pivot in the same places, (this is the concept of a "Class"). You could consider the motion to be designed for a skeleton composed of jointed bones over which any flesh could be assigned.

Once there is a substantial library of relative motions, or "Actions", built up, a story-teller could pick and choose a skit for a given set of Characters. We now introduce absolute motion. Consider, for example, if asked, "Do you know how to walk?" You would say "yes" even though I did not specify which direction you were to walk, how fast or how far. Of course, there are many different kinds of "walks": i.e. bouncing, sad, rotund, etc. For a library of actions to be general, it cannot be dependent upon variables such as speed, distance or direction. These specifications compose what I call absolute motion. The user must calculate the proper absolute motion to prevent a Character that

is walking from appearing to be ice skating; meaning if a Character's "walk" Action makes the feet separate 20 units each step, the Absolute movement of the Character should also be 20 units per step.

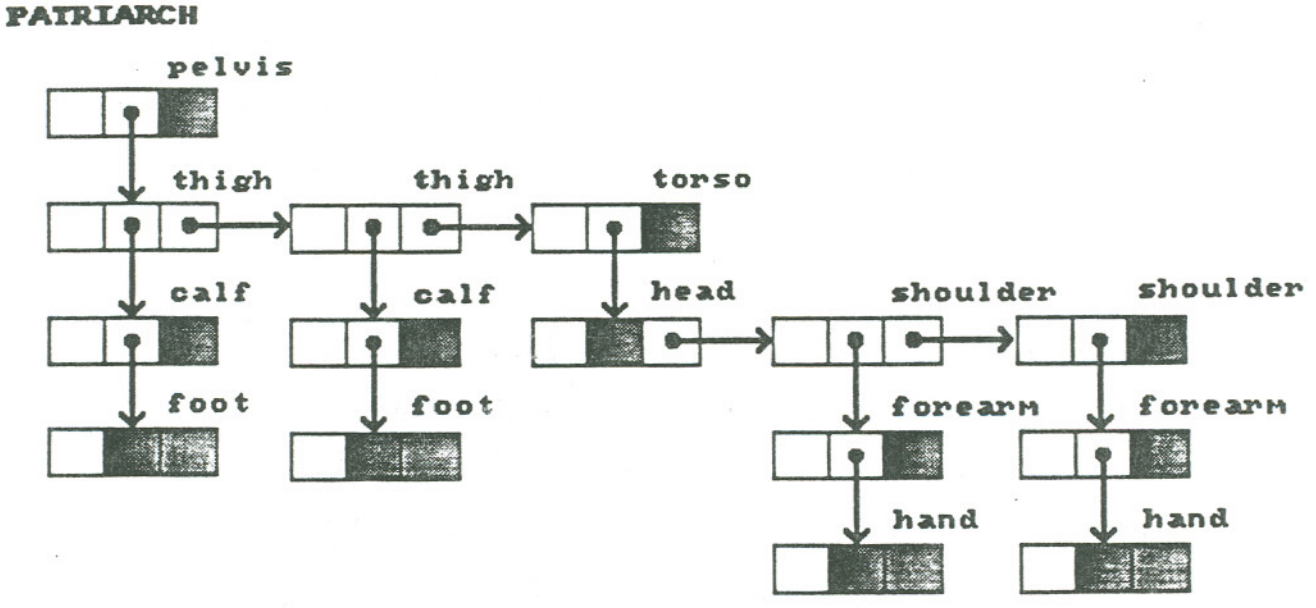
3. SKELETAL REPRESENTATION

Because a scene editor has no knowledge of the object representation technique used by the rendering process, it must have its own internal method of representing objects and their relationship to each other. The fundamental building block I call a "Bone", which is simply represented as a vector defining the distance from the pivot to the major axis of the Bone.

Bones have the following components: reflection constants for lighting, the Bone name, the Bone's position in relation to its parent, the end points of the Bone vector, rotational constants defining the Bones's initial orientation, pointers to the adjoining Bones in the Figure, and pointers to adjoining Bones if Figure substitution occurs, (discussed later).

```
struct Bone {
 float      specular,    /* reflection */
            diffuse;
 char       bonename[STRINGSIZE];
 Coordinate position,    /* relative to parent */
               extent1,  /* vector */
               extent2;
 TSR        tsr;         /* transformation */
 struct Bone *progeny,   /* binary tree */
            *sibling,
            *hierprogeny, /* substitution links */
            *hiersibling;
};
```

Characters are composed of Figures, which in turn are composed of Bones. A Bone would be a fundamental moving part of an object. Consider modeling a person's head: the Bones would be the skull, lips, jaw, ears, eyelids, eyes, and eyebrows. This combination would be a Figure.

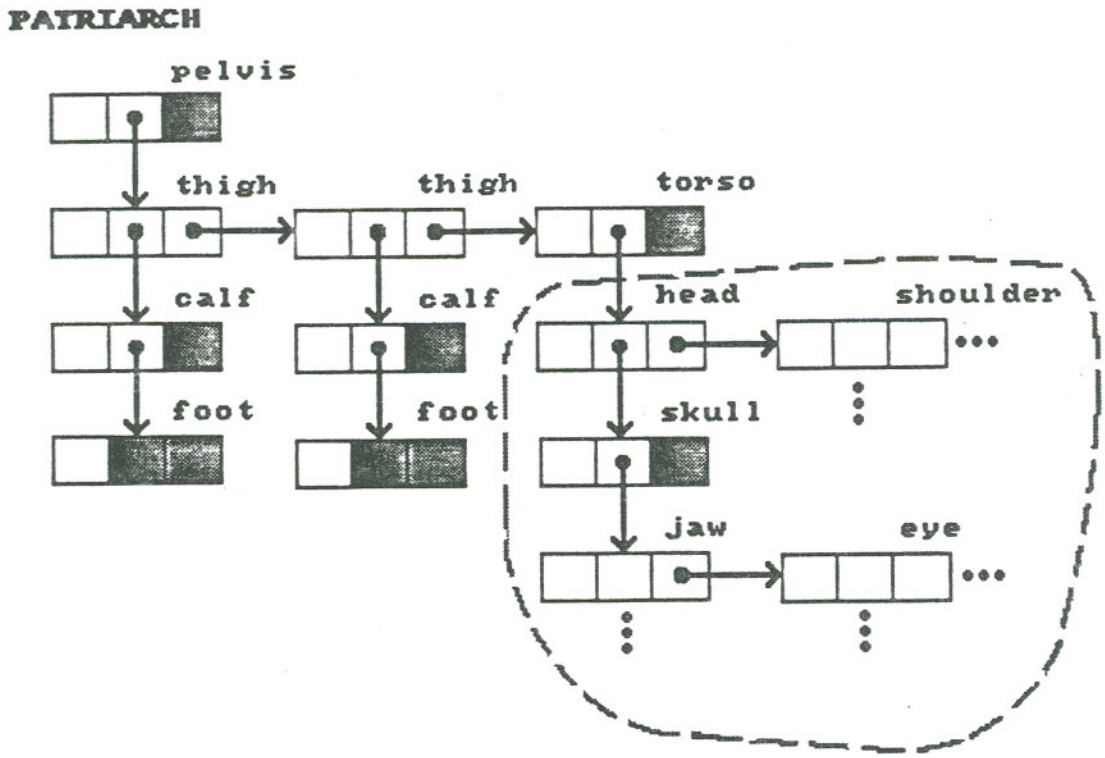


Bones combines to form "Figures"

If the head Figure were a single part of a larger construction, (a whole body), and the body was similarly divided into Segments such as thigh, calf, forearm, torso, pelvis, foot , etc., the combination of Figures is called a "Character". The differences are subtle since a single Figure is also a Character. Consider the following example.

A biped Figure is designed that has an articulated head as an integral part of the Figure. In this case, the biped is both a Figure and a Character. A "walk" Action is designed for the biped and saved in a library, (an Action is the computerized definition of motion, see next chapter). Later, the biped needs to be able to walk and say the word "hello" simultaneously. Another Action is designed and saved in the library. Later yet, the biped needs to be able to walk and say the word "bye" simultaneously. As can be deduced, this could go on endlessly, even though the "walk" Action is the same in all three cases.

I propose a better way to construct a biped is to design the body without an articulated head as one Figure, and another Figure which is an articulated head that could be substituted onto the body as needed. The head's Actions are independent of the body's Actions, hence the body could use a universal "walk" and the head could have Actions for the words "hello" and "bye". In this case, the combination of the two Figures is one Character. The power of this concept allows a relatively small set of libraried Actions to generate a great amount of combinatorial motions, and many different Characters can be constructed from a small set of Figures.



Figures combine to form "Characters"

The software structure used to represent this concept is constructed as connected binary trees. It contains a list of absolute movement for the Figure, the name of the Figure, a pointer to the root Bone, a list of relative Actions for the Figure, and pointers to adjoining Figures in the binary tree.

```
struct Figure {
   Action         *action;     /* defined later */
   char           figurename[STRINGSIZE];
   Bone           *patriarch;  /* root of B-tree */
   Script         *script;     /* defined later */
   struct Figure *progeny,     /* binary tree */
                  *sibling;
};
```

I used hierarchial object connectivity utilizing Bones in a successfull software program called, "ANIMATION:Apprentice", initially released in May 1987.

4. ACTIONS

After a Figure is created, the intended Actions of the Figure can be specified and saved in generic "Class" libraries for use with all Figures in the Class. A Class is defined as all Figures with the same connectivity.

Actions are saved as key frames with the intermediate or "between" frames interpolated between them. This is important because a walking Action must accommodate a range of speeds. For example, a "walk" may have been defined for a person traveling at 6 feet/second, but the animator wants his Figure to "walk" at 10 feet/second. By using key-frame-timing, the same walk can be used at any speed. Animation does not rigorously follow physical laws, so great leeway is given for this concept. Also, remember that a light, springy "walk" is different from a slow, dejected "walk" and should be defined differently.

All TSR, (translate, scale, rotate), variables and the keyframe timing associated with these variables are defined interactively by the animator using the Apprentice "Action" module. Actions are really a series of SubActions connected as a list for each Bone in the Figure. For example, the left forearm Bone would have a SubAction list specifying:

* move up 15 degrees for 3 frames
* hold for 10 frames
* move down 30 degrees for 6 frames
* hold for 2 frames

Actions have the following components: the transformation variables, the number of "between" frames for the keyframe, the number of pause or "hold" frames after the final position is achieved, the acceration/deceleration for the motion (called "ease"), and a link to the next Action.

```
struct action {
   TSR           tsr;    /* transformation */
   float         factor, /* key frame timing */
                 hold;
   UBYTE         ease;   /* acceleration */
   struct Action *next;  /* next keyframe */
};
```

Actions should be as generic as possible in their scope of Figure sizes. A short man should be able to use the same walk as a tall man. Rotations and scales are not size dependent, but translations are. Apprentice handles translation by comparing the Character's "patriarch" Bone, (the root of the binary tree), to a default value embedded in the Action, then scaling all translations in the Action by the ratio between the two.

4.1 Connecting Actions

Most motion is very complex. The simple act of walking involves almost every moveable Bone of a Character, and all of the Bones interact. For example, as the "upperarm" Bone swings up, the "forearm" Bone swings at a faster pace.

Each movement of every individual Bone is called a Subaction.

Because Actions are so difficult to create due to their complexity, we have established certain procedures and tricks. With Animation:Apprentice, you can choose to look at the stick-figure representation of a Character only from the ordinal views: top, front, right, bottom, back, and left. Ordinal views reduce the complexity of complex angles.

Luckily, many Actions are cyclic, like walking. Walking is also symmetrical, since the left hand side of the body mirrors the right.

Actions should always start from a known initial position. For example, a Transition Action gets the Character from a known standing position to the start of a "step".

A "walk" is compound of "steps", and "steps" are cyclic. The initial position of a "step" should also be the final position so that the "steps" can be smoothly

repeated to perform a "walk".

The hardest thing to understand about Subactions is Holding. Holding is simply maintaining the last established position for a given number of frames, however, in practice, Holds may be confusing. Consider the list of Subactions for your Figure's "right forearm" Bone during a "step". Let's consider the Holds involved with the "forearm" Bone as related to its Parent, the "upperarm" Bone. Initially, the "upperarm" Bone angles back from the "torso" Bone at a 30 degree angle. The "forearm" Bone has no angle initially because it extends straight down from the "upperarm" at the starting position. The "step" occurs in 24 frames and it is cyclic, so we have 12 frames to move upperarm forward and 12 frames to move it back to the starting position. If you watch someone walk, you will notice that halfway through the upperarm's forward swing, the forearm begins its own forward swing, relative of the upperarm. So, as we have identified it, the forearm Holds its starting angle for 6 frames, swings to an angle of 35 degrees from the upperarm in 6 frames, returns to the starting position for 6 frames, and Holds for the last 6 frames.

Since the skit of a Figure or SubFigure is really a series of serially connected Actions, accessing a Character's Pose at any particular frame must be carefully handled. For example, at frame 37 of a skit,

a Character's SubFigure body may be in its third Action of its list while its head is in the twelfth Action, and the right arm SubFigure in its first Action. A "Script" is kept for each Figure in the Character defining the "frames" and "hold" values for each Action in serial order. To get to frame 37, each Script's frames and hold combination is subtracted from the total until the activated Script is found. The Action name of this activated Script is then loaded and interpolated to compute the desired frame. For example, a script contains something like this:

		FRAMES	HOLD
1	walk	12	0
2	walkend	2	6
3	walkstart	2	0
4	walk	12	0
5	walk	12	0

37 - (12 + 2 + 6 + 2 + 12) = 3

So the fifth Script is activated and is interpolated to three frames.

4.2 How Actions Deal With Figure Substitution

Remember, an Action is eligible for use with a Figure if its hierarchial layout is consistent with the "Class". This causes major concern when dealing with substituted Characters because the hierarchal layout is substantially changed. Since this layout can never be

predicted, a method of general Action loading must be developed. ANIMATION:Apprentice solves this problem in the following manner:

When a Figure is loaded, all the Bones are loaded, whether or not they are displaced by substitution. Since Actions are actually composed of a series of SubActions attached to each Bone, and the Bones are hierarchial in nature, there is no way to simply terminate loading of a Action when a substitution is discovered. Instead, two pointers are maintained, labeled "hiersibling" and "hierprogeny". If a substitution is to occur, these pointers are updated with the patriarch Bone of the substituting Figure, otherwise they remain NULL. So what we end up with in a Character is a linked list of Bones, a very few of which have the hiersibling or hierprogeny fields pointing to other entire Figures.

When a new Action is to be loaded, it ignores the hiersibling and hierprogeny fields and simply loads in the SubActions to each Bone of the respective Figures. In short, loading an Action into a substituted Figure is no different than loading it into a "vanilla" Figure.

The single contrary operation is the "transversal" code. Transversal is named after the term "transversing the tree", since a figure is a hierarchial B-tree and

this operation "transverses" it. A Character is rendered a Bone at a time, and if it sees any value other then NULL in the hiersibling or hierprogeny fields, it takes that path rather than the parochial one. SubActions are still loaded and cleared generically, but heir substituted Bones are bypassed.

5. CHOREOGRAPHY

Compiling classes of building-block Figures and generic Class Actions into libraries is the major strength of a complete scene editor, however, a scene editor is not complete without a method of designing choreographs for your Characters to perform in. Many other explicit directions must also be made to define the complete rendering process, including camera motion and light source placement. Animation:Apprentice elegantly handles all of these settings via a user-interactive interface for the animator.

5.1 Adding A Character To A Play

The definition between Character and a Figure is exemplified in this module. Remember, a Character can be a combination of Figures, where for example, an articulated "head" Figure is substituted for the simple "head" Bone of a "biped" Figure. This substitution takes the following form.

Suppose you want to add a new cast member to the play. You select the ADD POSE menuitem and a list of Figures on the current "library" disk is offered for your inspection. You select "Bobby" as your base biped Character. As Bobby is being loaded into the

Choreography, each Bone name in Bobby's Figure description is compared with other Figures in the library. Now, you have previously composed the library to include a Figure named "Bobby_head" which matches a Bone of the same name in the Bobby Figure. If you did not wish the substitution of the new Figure to occur, you would not have included the "Bobby_head" Figure in the library, or you would have renamed the head Bone in the Bobby Figure. You might do this during check renderings for speed, since unavailable Figures are ignored and cause no computation penalty.

Assuming you want the Bobby_head substitution to occur, a list of eligible Actions for Bobby_head will then be listed. Similarly, lists of eligible Actions for other of Bobby's SubFigures will be presented. When this recursive process is completed, the choreography module will have defined a cast member whose initial Pose is defined for every Bone.

Selecting the PERSPECTIVE menuitem will show the stickman representation of the total Character with full substitution and the initial Actions attached.

Now, each SubFigure can be individually selected graphically and lists of Actions attached to them. In this manner, Bobby's body can be performing a "stock" "walking" Action, while Bobby_head is doing something entirely independent. Editing can also be performed on

this list of Actions, including the substitution of a new initial Action. One caveat remains: the new "Bobby" character now consists of two SubFigures and their associated Actions. If any portion of this Character cannot be fully loaded at render time, the entire Character is aborted.

5.2 The Script Structure

Each Figure in the choreography has "Scripts" which contain the series of Actions and key-frame-timing to be used to modify the Actions's speed. A Script consists of the Action's name in the library, the keyframe length (the relative Action can be speeded up or slowed down through interpolation using the keyframe length), and the number of hold frames the Figure is to remain still after the Action is completed. There is also a pointer to the next Script.

```
struct Script = {
   char          actionname[STRINGSIZE];
   int           frames, /* activation values */
                 hold;
   struct Script *next;
}
```

6. CONCLUSION

Scene editors are mandatory for "Disney-style" character animation. To make scene editors available to non-computer-literate artists, they must be intuitive and graphical. The new generation of interactive, "windowing" environments on inexpensive graphic computers, such as the Commodore Amiga, Atari ST, Mac II, Sun, and the like, are ideal for implementation of these requirements.

The concept of hierarchial Figures and libraried, independent Actions is presented for what I think is the first time. As far as I know, this is the first working or conceived example of "librarying" in computer animation, even through it is widely used in other aspects of computer science.

6.1 IMPLEMENTATION

The graphical interface was written in the programming language "C" and implemented on the Amiga computer running Intuition 1.2. The Scene Editor consists of over 10,000 lines of code. The Scene Editor is usually the most important part of the animation process.

The major part of the time developing the tool was conceptualizing the needs and requirements of reusable, computer animation, and rest of the time was spent in "trial-and-error" experimentation. The project has continued for over three years and continues to expand.

6.2 DRAWBACKS IN THE INITIAL IMPLEMENTATION

When the concept of the hierarchial matrix was developed, a specific ordering of the matrix multiplications required was placed on the system [17]. Through observation, it was decided the rotational order would be:

ROLL
TILT
SWIVEL

This ordering would allow an arm to turn about its longitude axis even if a tilt or swivel rotation was also applied. Similarly, tilt was chosen before swivel. Though in many instances this ordering is sufficient, and it is easily implemented, a significant portion of motion can not be specified. Complex human shoulder motion is an example.

After investigation, the ideal implementation would actually be two or three matrices per joint, each with a roll, tilt, swivel ordering, which will solve all problems.

6.3 EXPERIMENTAL RESULTS

A videotape of customer created vignettes is available for inspection. All animations were created with the Animation:Apprentice program as implemented for commercial sales. Several small vignettes have been shown on cable access TV and I have been interviewed on a television computer talk show about my programs.

REFERENCES

1. John K. Krouse, "Shopping for a Solid Modeler", Machine Design, January 20, 1983, pp 56-63.

2. Edwin Catmull, "The Problems of Computer-Assisted Animation", SIGGRAPH , Volume 17, Number 3, 1983, pp 348-353.

3. Nadia Magnenat-Thalmann, Daniel Thalmann, "Three-Dimensional Computer Animation: More an Evolution than a Motion Problem", IEEE CG&A, October 1985, pp 47-57.

4. William T. Reeves, Ricki Blau, "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems", SIGGRAPH, Volume 19, Number 3, 1985, pp 313-322.

5. Roy A. Hall, Donald P. Greenberg, "A Testbed for Realistic Image Synthesis", IEEE CG&A, 1983, pp 10-20.

6. T.W. Calvert, J. Chapman, A. Patla, "Aspects of the Kinematic Simulation of Human Movement", IEEE CG&A, November 1982, pp 41-59.

7. K.D. Willmert, "Visualizing Human Body Motion Simulations", IEEE CG&A, November 1982, pp 35-38.

8. Don Herbison-Evans, "Real-Time Animation of Human Figure Drawings with Hidden Lines Omitted", IEEE CG&A, November 1982, pp 27-33.

9. William A. Fetter, "A Progression of Human Figures Simulated by Computer Graphics", IEEE CG&A, November 1982, pp 9-13.

10. James U. Korein, Norman I. Badler, "Techniques for Generating the Goal-Directed Motion of Articulated Structures", IEEE CG&A, November 1982, pp 71-81.

11. Nadia Magnenat-Thalmann, Daniel Thalmann, Mario Fortin, "Miranim: An Extensible Director-Oriented System for the Animation of Realistic Images", IEEE CG&A, March 1985, pp 61-73.

12. Richard Chuang, Glen Entis, "3-D Shaded Computer Animation - Step by Step", IEEE CG&A, December 1983, pp 18-24.

13. Frederic I. Parke, "Parameterized Models for Facial Animation", IEEE CG&A, November 1982, pp 61-68.

14. Marianne Dooley, "Anthropometric Modeling Programs- A Survey", IEEE CG&A, November 1982, pp 17-25.

15. Fumihiko Kimura, "Geomap-III: Designing Solids with Free-Form Surfaces", IEEE CG&A, June 1984, pp 58-72.

16. William T. Reeves, "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects", ACM Transactions of Graphics, Vol. 2. No. 2, April 1983, pp 91-108.

17. Jay P. Fillmore, "A Note on Rotation Matrices", IEEE CG&A, February 1984, pp 30-33.

18. Steve Feiner, David Salesin, Thomas Banchoff, "Dial: A Diagrammatic Animation Language", IEEE CG&A, September 1982, pp 43-54.

19. Herve Huitric, Monique Nahas, "B-Spline Surfaces: A Tool for Computer Painting", IEEE CG&A, March 1985, pp

39-47.

20. Brian A. Barsky, "A Description and Evaluation of Various 3-D Models", IEEE CG&A, January 1984, pp 38-51.

21. Frank Thomas, Ollie Jonston, the book, Disney Animation: The Illusion of Life, a Spectrum Book.

22. Shinichiro Haruyama, Brian A Barsky, "Using Stochastic Modeling for Texture Generation", IEEE CG&A, March 1984, pp 7-19.

23. K. Yamaguchi, T.L. Kunni, K. Fujimura, "Octree-Related Data Structures and Algorithms", IEEE CG&A, January 1984, pp 53-59.

24. Wm. Randolph Franklin, Alan H. Barr, "Faster Calculation of Superquadric Shapes", IEEE CG&A, July 1981, pp 41-47.

25. D. Zeltzer, "Computer Articulated Animation", Byte Magazine, April 1982, pp 167-186.

APPENDIX A

TRADITIONAL ANIMATION AND THE COMPUTER (as of February 1987)

To begin a computer animation project, we must first understand how animation is traditionally done by hand. The pencil drawings are created in bulk in the Asian countries of Singapore, Korea, and Japan. Hanna-Barbera Studios, a giant in the Saturday morning cartoons, does virtually no in-country animation. They do, however, use computers to do the coloring. The pencils are scanned into the computer and image processed to delineated the lines and to enclose solid areas in boundaries. A series of people then call up the digitized cell and fill in the areas by selecting the appropriate color and "flood-filling" each closed area. the computer is then used to matte backgrounds onto the cells and record to 1-inch videotape directly. Bookkeeping is also performed to see where each cell is in the process.

At the high end of traditional animation is the "Disney" style. Movement is fluid and subtle. It is now so expensive that it cannot be produced profitably. Hanna-Barbera style animation is so inexpensive to produce that the computer can only offer speed and quality at the same price, a much more difficult task.

APPENDIX B

COMMERCIAL REQUIREMENTS FOR A SUCCESSFUL ANIMATION SYSTEM

1) Frames must be generated in less than 20 minutes. I suggest that if the animation cannot be generated in real-time, then the next major time-critical concern is how many seconds can be completed per night. The range here is ideally the length of a single vignette, 3 to 10 seconds. Normally, animation requires at least 12 frames per second, (24 fps for backgrounds), so each night the renderer must complete 36 to 120 frames. At a maximum of 20 minutes per frame, a 3 second vignette would take 12 hours, which is barely acceptable. An average frame generation time of 5 to 10 minutes must be the norm, or less than 8 hours per 6 second vignette.

2) The object editor must be usable by an artist. It must be easy to use and understand and be totally graphical in its operation. Esoteric numbers must be minimized and mathematically concepts need not be known except as part of a common sense mentality.

3) The object editor must be flexible. People,

cars, buildings and logos should be equally represented.

4) The scene editor must allow the librarying of motions from one animation piece to another. This requirement has the effect of multiplying the animator's ability to quickly and efficiently produce animation vignettes.

BIOGRAPHICAL NOTE

The author was born on June 14, 1958 in Moab, Utah. He attended Elk Grove High School, Elk Grove, California and graduated from high school in June 1976. He entered California State University, Sacramento and received his Bachelor of Science degree in Electrical and Electronic Engineering in May 1980. The author moved to Vancouver, Washington in July 1980 to work at the Hewlett Packard, Vancouver Division plant as a Marketing Engineer. He received his Master of Business Administration from the University of Portland, Oregon in May 1984. He continued his career at Hewlett Packard as a Software Design Engineer until December 1987 when he started a software company that produced computer animation programs.

In September 1984 he began his studies at the Oregon Graduate Center where he completed the requirements for the Master of Science degree in Computer Science in December 1989.