STORY: A HIERARCHICAL ANIMATION AND STORYBOARDING SYSTEM FOR ALPHA_1[1]

Michael S. Blum, Robert Mecklenburg
C.S. Department

UUCS-93-009

Department of Computer Science
University of Utah
Salt Lake City, UT 84112, USA

April 7, 1993

Abstract

We introduce an integrated animation and storyboarding system that simplifies the creation and refinement of computer generated animations. The framework models both the process and product of an animated sequence, making animation more accessible for communication and as an art form. The system adopts a novel approach to animation by integrating storyboards and the traditional film hierarchy in a computer animation system. Traditional animation begins with storyboards representing important moments in a film. These storyboards are structured into shots and scenes which form a standard hierarchy. This hierarchy is important to long animations because it reduces the complexity to manageable proportions. We also introduce the animation proof reader, a tool for identifying awkward camera placement and motion sequences using traditional film production rules.

[1]This work was supported in part by the DARPA (DOD) (N00014-91-J-4123 and N00014–91–J–4046), and the NSF and DARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-89-20219).. All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

1 Introduction

The *story* system is an integrated animation and storyboarding system that simplifies the creation and refinement of computer generated animations. This framework departs from standard computer animation systems by modeling both the *process* of creating animation and the *product* of the animators efforts explicitly. The framework allows an animator to produce better organized and more dynamic animations in a shorter amount of time than is currently possible. By structuring the animation process, we have taken a step towards making computer animation both a more accessible art form and a more useful communication tool.

The process of creating an animation is iterative in nature. The individual or team producing the animation must develop a story (often represented as storyboards), model the characters and props, decide the composition of each shot, script all dialogue or other sound effects, generate the motion of all the moving characters, and render each frame of the animation. All of these elements may be modified or refined before the final animation is completed. Enabling animation teams to make simple refinements quickly, minimizing the time consuming and computationally expensive operations required to revise models, generate motion and render images, will result in significant time savings. In addition, such improvements will result in a better organized and more dynamic presentation of the story.

The film and animation community have developed a decomposition technique[1, 16] for describing animated and live action sequences which we draw upon for our framework. The framework consists of a hierarchy[12] which describes an animation as a sequence of *scenes* containing a number of *shots* (see Figure 1). This framework is well suited to a progressive refinement style for constructing animations. In particular, a scene is a piece of animation with a collection of objects in specific locations during a period of time. A scene may be broken up into a number of *shots* for dramatic purposes. Shots record the action as seen from a particular camera position, and by editing these shots together, a complete scene is created.

The film and animation community also have developed production techniques for creating animated and live action sequences. A shot may be represented by one or more *storyboards*. Storyboards, first used by Disney animators in the 1930's, are used to suggest action, relationships, locale, and the appearance of characters. These storyboards are expanded and refined and eventually become the scenes and shots of the film. In a computer generated animation a much more fluid relationship between storyboards and the final animated sequence is possible. In our system, storyboards are an integral part of the animation creation process.

Computer animation systems have not adequately addressed the process of film creation through storyboards and the product of the animation as a hierarchy of scenes and shots. By applying the process and product models of the film community to computer animation, we gain several advantages. First, by using the structure established in the film and animation industries we make computer animation more accessible to industry professionals. Second, we directly address the problem of managing long, animated sequences. Feature length computer animations will be technically possible in the near future. Unfortunately, a practical framework for managing and refining such features has not been adequately explored. As early as 1978 Catmull[2] stressed the importance of using computers to manage storyboards and associated information, but we have not seen storyboards as traditionally used discussed in the literature[8]. Finally, the framework applies even to short animations since it stresses efficient use of computational resources which is a general problem.

Eventually animation will become a "desk top" industry as publishing is today. Novices who produce animations lack the aesthetic judgement which professional film makers have gained through

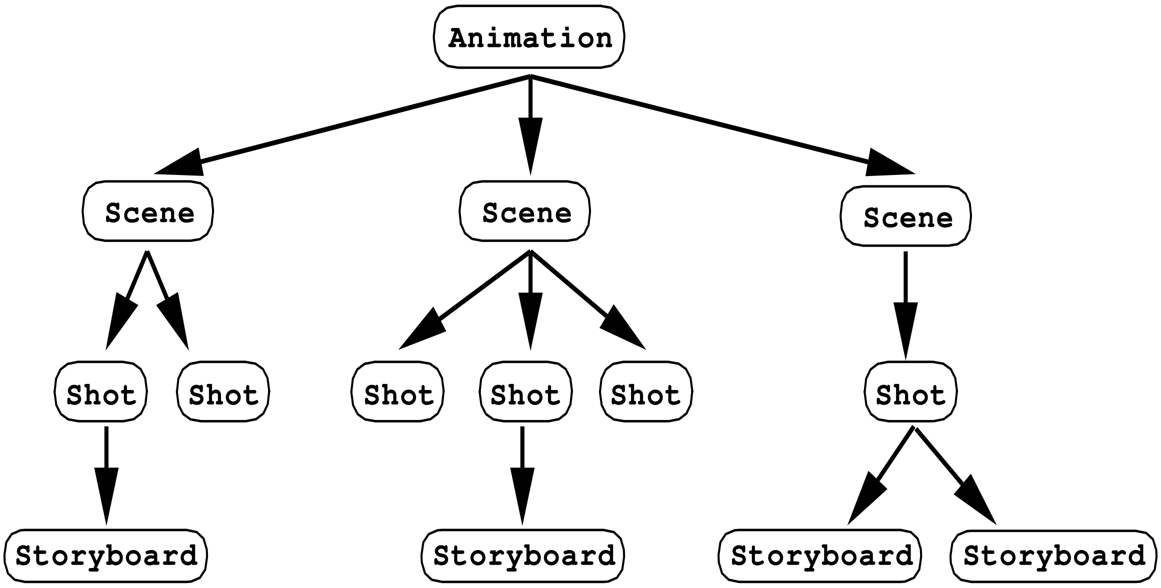

Figure 1: In traditional film a movie consists of one or more scenes. Each scene consists of one or more shots. During film production shots are often represented by storyboards.

a life time of education and practice. Advice on animation aesthetics, like advice on word usage, grammar or spelling in publishing, might be provided by a set of automated tools. This advice can be formulated as a set of rules applied by an animation proof reader which is able to identify and advise animators on problems in animated sequences.

Our system is conceptually divided into five components. *Animation Production* facilities manage storyboards and maintain the hierarchical framework used by the rest of the system. *Camera Placement* tools provide an animator with a variety of mechanisms to create and position cameras and examine their views. The *Property Manager* is a flexible model manager that provides version and complexity control of models. The *Rendering Manager* provides various rendering and previewing features for producing the frames of the animation. Finally, the *Animation Proof Reader* identifies awkward camera placement and motion sequences according to a set of traditional film production rules. We begin by briefly discussing the context in which an animation system must perform, followed by sections discussing each of the major components. Finally, we discuss the status of the system, future work, and conclusions.

2 Overview

Computer animation is not a new field, and many of the problems associated with it have been solved adequately for current needs. Among these "solved" problems are producing photo-realistic images, physical and non-physical motion, simulations of natural processes (e.g., procedural modeling of fire and plants), and geometric modeling. Clearly, no monolithic animation program can provide all of these features and provide fast, efficient, and current algorithms. Therefore a complete animation system providing all of these facilities must operate in a diverse environment and integrate a variety of modeling tools.

Our system is written in the context of the Alpha_1 modeling system[4]. This system is centered around a NURB-based geometric modeler that can be used both interactively and through a network

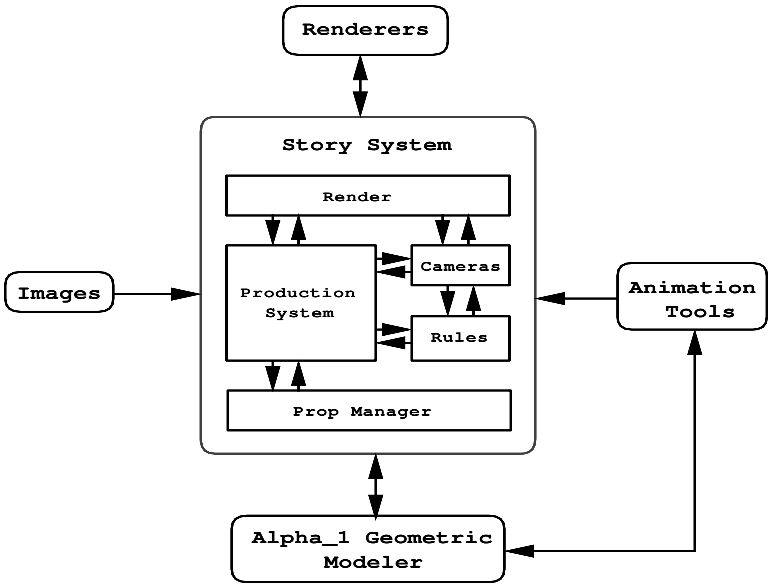

Figure 2: An animation system must communicate with and rely upon external tools to provide models, motion curves, and rendering.

interface as a server. In server mode, the modeler acts partly as a gateway for transferring models and control information between various utilities. Although oriented towards geometric modeling and manufacturing, Alpha_1 provides a tool-rich environment conducive to complex animation.

Some of the services provided by the Alpha_1 system are: scanline and ray traced rendering, physically and non-physically based motion generation, interactive sculptured surface creation, and a common communications format for tool integration. Our animation system a forms "hub" from which these tools are accessed. Figure 2 provides an overview of how the system interacts with the rest of Alpha_1.

The story program communicates directly with the geometric modeler, shape_edit, which mediates further communication with clients for physically-based motion generation and a key-frame system for non-physically-based motion. Story's rendering module controls the type and action of rendering. The system can also import scanned images provided by the animator. Providing loose integration for this complex environment produces a flexible and powerful animation system which would not be possible with a more monolithic design.

3 Animation Production

The heart of our system is support for the animation hierarchy. This support is provided by four objects: movie, scenes, shots, and storyboards. These objects model the traditional film hierarchy and structure the film, providing a focus for various production information (see Figure 1). This structure allows longer and more intricate animation to be produced by reducing the production complexity.

In conventional film and animation, production begins with a set of storyboards depicting each shot in the film. These storyboards may diagram the view through the camera and also traditionally include dialog, sound effects, special effects, and suggest movement (both of the camera and of the characters). Our system supports two types of storyboards: images generated by the animation system and images imported from other sources. Initial storyboards are scanned in from photographs or sketches. These storyboards will form the basis for model creation and further refinements of shot composition. A storyboard can have associated text including all the traditional information as well as lists of objects important to a computer generated animation such as models, texture maps, and motion curves. When the geometry associated with an imported storyboard is complete, a new storyboard is defined by composing the objects in the scene. An image of the storyboard is then generated by the system using one of several renderers. Once a synthetic storyboard is defined the system can automatically update the storyboard when changes to the component objects occur (see Section 5.3).

A shot is one component of a scene and describes the view of a collection of objects from a particular camera. The objects in a shot are called `props` and can be geometric models, light sources, motion curves, or any other object supported by the underlying modeler. Each shot contains a list of props used and a set of storyboards which originally inspired the shot. In addition, shots include a start and end time and the duration of the shot in the final scene. Shots also have a name and associated textual information. A scene is a sequence of shots all viewing a common collection of props. Finally, a movie object consists of a sequence of scenes. Each of these objects can be annotated with a name, subject summary, responsible individuals and other text. In traditional film and animation, production information includes the *route* and *exposure sheets*. The *route sheet* lists the length, vital statistics and the name of the person in charge of each scene. The *exposure sheet* lists dialogue, background, models and model attributes (e.g., texture, surface quality), and camera position for each frame in the animation[2]. In our system this information is located in scene and shot objects, and in the geometric models themselves.

Although the movie itself is described by a strict object hierarchy, the user is free to create an animated sequence in any order he chooses. For instance, if the user chooses to define a shot before a storyboard, the system will automatically create a movie and scene to contain the shot. Later the user can rename and reorder these components to fit the evolving sequence. Adding, removing, and reordering storyboard, shot, and scene objects can all by performed by picking the object and selecting the desired action from a menu. The ordering of elements at each level of the hierarchy dictates the order of the final animation. These features allow an animator to quickly sketch out the structure of an animation in just a few mouse clicks.

Once a shot has been created, the user adds props and a camera to the shot to define its contents and view. Initially, props are displayed in a separate workspace window. The user adds props either by picking the graphical object from the workspace window or by indicating the prop name. Cameras can be created and positioned by the user.

4 Cameras

Cameras in our system are defined with the standard from-to-twist specification[6]. A view matrix for each camera is calculated from this specification. Each camera has a separate viewport which can display the view volume that the camera actually sees. Much as television directors choose from a series of monitors with various camera angles, so too an animator has the ability to examine a shot from different camera locations.

Cameras have a variety of different display representations. They may be displayed as a trihedron of vectors that indicate position and orientation or as a more familiar icon resembling an actual movie camera. The *center of attention* or "to" point may be optionally displayed. Additionally, a skeleton representing the view frustum of a camera may be shown (see Figure 3). To help the animator discern one camera from another, we attach a color and a name to each camera. For

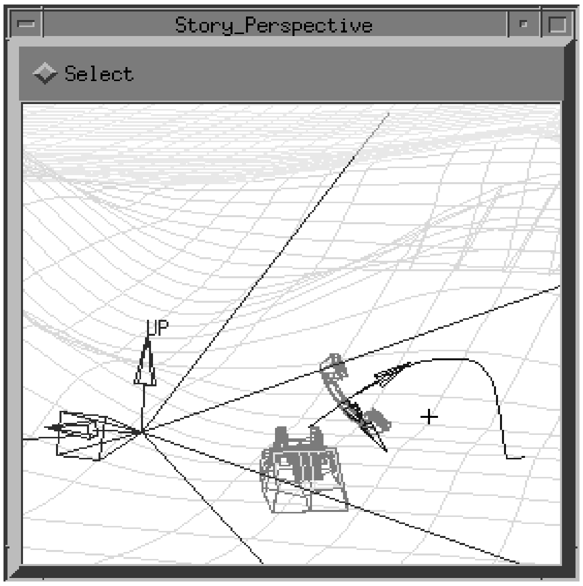

Figure 3: The story workspace window. The camera is rendered as the trihedron on the left with the viewing frustum expanding to the right. The small cross on the right is the camera's center of attention. Motion curves can also be displayed as polylines and an orientation trihedron.

instance, a green camera might be named "green1-camera". The viewport for a camera is easily recognizable by the color of the displayed camera.

We have provided a variety of tools to aid in camera positioning. The most primitive of these allows the user to enter numerical values for the "to" and "from" coordinates and for the twist angle. Somewhat better is the ability to use sliders to control each coordinate. In practice, neither of these methods is very convenient. A more intuitive technique is to allow either the "from" or "to" points to be snapped to objects in the scene. This is implemented by casting a ray from the eye and selecting the intersection point on the closest object. This technique allows the general position and orientation of a camera to be quickly established.

We also support the "flying vehicle control" metaphor for cameras that Ware[17] chose as the metaphor best suited for movie making. We allow the user to "fly" the camera from its own perspective (coordinate system) through the scene. The flying mode is selected from the same set of buttons we use to change the view of the scene. However, instead of mouse movements altering the view, these movements change the position and orientation of the camera. The user may translate or rotate the camera along or about any of its three axes. Some of these modes can be used in parallel. These tools make camera positioning straightforward.

5 The Property Manager

An animated film uses many geometric models, light sources, texture maps, and motion curves. In addition, there may be several versions of any one prop and possibly several different props designed to be interchangeable with one another. Add to this the desire to have multiple levels of complexity for a single model (to efficiently use computational resources) and we have a significant model management problem. We address this problem with a simple, but flexible model database called the *property manager*. The property manager has three main functions: associate a name with an object, link an object with a motion curve, and generate a simple approximation for a complex model.

As an example, assume we need a gun prop. The property manager might contain three different gun models, a Colt, a Luger, and a BB gun. We indicate to the property manager that each one is a possible *instance* of a gun prop and that the Luger is the currently active instance. These three models are referred to as *prop instances*. Furthermore, we may have several different *versions* of each of these guns. For instance, one version of the Colt may have an engraved stock and one a plain stock. Also, each instance might have versions of varying *complexity*. We might have a bounding box version of the Colt, and a complete trimmed B-spline version. Although the "gun" prop name has been bound to the Lugar object, the version number and complexity measure are still free to change. Since all references to props are routed through the prop manager, the version number and complexity can be varied on-the-fly when the geometric models are requested for display.

As stated earlier, a prop is just another name for an object in our system. Therefore the property manager handles not only geometric models, but also lights, textures, *motion curves*, etc. A motion curve is a function, represented as a B-spline in the Alpha_1 modeling system. This spline specifies the values of some transformation over time. These motion curves can specify rotation, translation, scaling, and other transformations (e.g., quaternions). The y-axis specifies the value of some transformation at each point in time, represented by the parameter, t. For example, $rx(t)$ is a curve representing rotation about the x axis.

The x-axis of a motion curve corresponds to time and the y-axis specifies the value of some transformation at each point in time.

Currently, Alpha_1 includes several tools which are capable of generating motion curves. These tools are representative of both common and state-of-the-art animation techniques. Alpha_1's physically-based modeling tools can simulate the dynamics of rigid link/joint systems[5], create goal directed motion of linked creatures using spacetime constraints[3], and animate flexible surfaces (such as cloth)[15]. The non-physically based tools include a simple key-framing system and the Alpha_1 geometric modeler, shape_edit, which can be used to create models and to procedurally animate them. Since shape_edit is also a server we can perform modeling and animation operations in shape_edit and then incorporate these models into the property manager interactively.

In general, the property manager does not create models, it manages associations between models and names and provides flexible access through the name, version number, and complexity measures. However, there are two instances under which the property manager will create models. The first is when associating a model with a motion curve; the other is when generating simple versions of models. We discuss these two cases next.

5.1 Linking Props and Motion Curves

The Alpha_1 modeler supports *instances* which consist of a geometric model and a transformation matrix. This transformation can be either a collection of simple matrices or a collection of motion curves. In order to derive the greatest flexibility from models and motion curves, the animator can register models and motion curves with the property manager separately and link them dynamically while composing an animation. This is done by creating an instance whose geometry and transformation are both property names. Because the references are symbolic, by changing the value of a property name we change the representation of the linked object. For instance, suppose we have a "projectile" prop and a "trajectory" prop. Initially the "projectile" prop might be a "jet" object. We can create a new prop, called "flying" prop, that links "projectile" and "trajectory". When we added this new prop to a shot, and preview the animation, we see a flying jet. Now we can switch

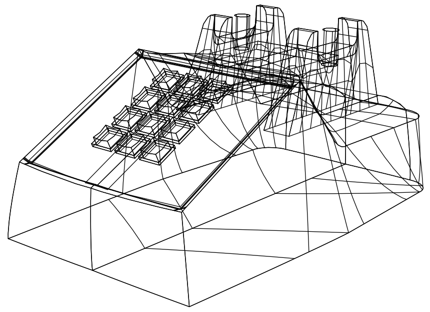

Figure 4: A wire frame rendering of a complex B-spline model.

to another instance of the "projectile" prop, perhaps a "train" instance (we can make anything fly in our world!), the corresponding "flying" prop now has the train attached to the motion curve. This facility allows an animator freedom to mix and match geometry and motion quickly and easily.

5.2 Generation of Simple Models

The property manager can handle multiple copies of the same object modeled at varying levels of complexity. Switching between these models on-the-fly allows fast display on low performance graphical devices as well as better time/quality trade-offs during rendering. For this feature to be most useful, there should exist many models of the same object at varying levels of complexity, but the modeled components should correspond as closely as possible to the actual object. We solve this problem by automatically generating lower complexity approximations from full geometry.

One approach to approximating simple models from complex geometry calculates bounding boxes for each surface in the full model. Figure 4 shows a wire frame rendering of a B-spline telephone base. Figure 5 shows a version with a bounding box for every surface. This approach is insufficient for some models because of the large number of bounding boxes generated The resulting "simple" models are often more confusing than the original and the data reduction is relatively poor. A second approach is to extract boundary curves for every surface in a model. This also results in relatively poor data reduction, although the "simple" model approximates the original more closely.

The third approach is to selectively extract bounding boxes or boundary curves for surfaces dependent upon the contribution of the surface to the total model. Begin by approximating the total surface area of the model. Then, for each surface approximate its area and compare this value to the total surface area. If a surface's area is greater than the user defined total percentage, then a bounding box or boundary curve is extracted from the surface. Figure 6 shows a bounding box version of the telephone with a percentage of 0.020. Figure 7 shows derived models with percentages of 0.034 and 0.020, respectively. This algorithm can generate models which closely approximate the key features of their full complexity counterparts. The ratio of surface area to total area also serves as a complexity measure for the property manager. Unfortunately, spline

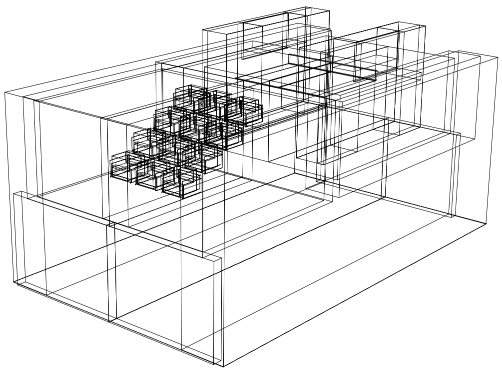

Figure 5: This model is generated by constructing a bounding box for every surface.

surfaces can vary arbitrarily within their boundary curves. Animators are encouraged to model with property manager capabilities in mind.

Surface area is approximated by subdividing each surface into polygons to within some tolerance, and summing the area of each triangle generated in the subdivision process. The algorithm can be summarized as:

```
complexity = user defined value [0-1].
totalArea = compute_surface_area( <model> );
simpleModel = NULL;
for each <srf> in <model>
  area = compute_suface_area( <srf> );
  if ( area / totalArea > complexity )
    curves = extract_boundary_curves( <srf> );
    simpleModel = append( curves, simpleModel );
```

These simple boundary curves work well when displayed in wire frame mode, but are not as useful when rendered with hidden surface algorithms. This is because the models are composed of curves, not surfaces. For “pencil test” renderings where an animator checks for occluding surfaces these simple models cannot be used. Fortunately it is straightforward to enhance this algorithm to extract surfaces instead of boundary curves.

The computation time needed to generate a simplified model from even a fairly complex B-spline model is fast enough to make these algorithms acceptable for interactive work. The telephone base contains 172 surfaces and can be approximated in less than one second on an HP 9000/730 workstation. Any number of simplified models can be generated for a given model from within our system. The new model may be added to the property manager by selecting the object from the workspace.

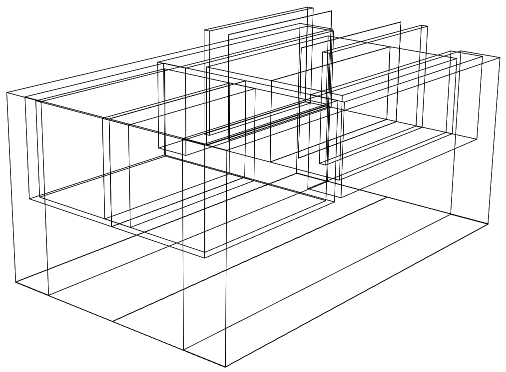

Figure 6: This model is generated with a complexity measure of 0.020.

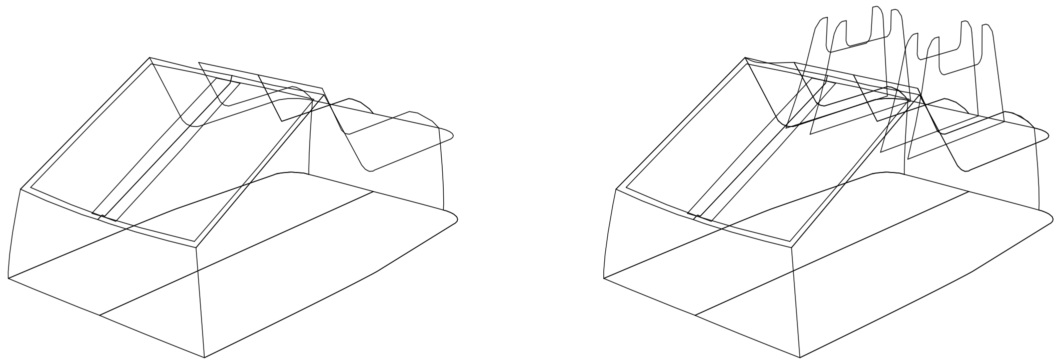

Figure 7: Two boundary curve versions of the telephone at complexity 0.034 (left) and 0.020 (right).

5.3 Additional features

A single model may be referenced through a variety of different prop names. For instance, in one animation, we may use a model of a pair of scissors as a "murder weapon" prop while another animation uses the scissors as a "barber shop" prop. By allowing quick substitution of one prop instance for another, an animator can easily experiment with different ideas. In an extreme case, we might have a scene which consists of the three props: landscape, hero, and action. Through the use of the property manager this might be a cowboy walking down the street of a western town or a ball falling down a set of stairs. In addition, the ability to rapidly substitute one level of model complexity with another, encourages progressive refinement of an animation.

Complex geometric models are large and manipulating them quickly requires careful use of physical and virtual memory. To improve memory use, the property manager lazily loads models associated with props and retains only the active instance of each prop. This optimization means that only one model per prop resides in memory at a time. When the user changes prop instances, the old model is freed and the new model is loaded. Reducing memory requirements improves overall performance of the system at the cost of small delays when a model is first referenced.

6 Rendering

In this section, rendering refers to viewing all or part of the animation in either wire frame or shaded image mode. We first discuss the "fast render" preview and editing options available to the animator. Previewing and editing typically use wire frame renderings of models but may use shaded images if hardware shaded image rendering is available. Next we discuss the high quality rendering facilities.

6.1 Preview and Edit

In film and animation, the quality of motion and the timing of scenes are critical[10]. Therefore, it is important for the animator to get a feel for the final motion as early in the production as possible. To facilitate this, we have implemented a comprehensive previewing capability. Among the features of the previewing module is the ability to view abstractions such as cameras, viewing frustrums, and motion curves (see Figure 3). We can also preview storyboards, shots, scenes, or the entire animation from the director's vantage or through the lens.

Producing animation by computer has advantages over either traditional animation or live action. Both cameras and motion curves can be previewed. As described earlier a camera can be represented as either a trihedron or a camera model. The animator can preview a shot with either a through the lens view or a director's view. Also, the view through several different cameras can be displayed simultaneously. The viewing frustum of a camera can also be displayed giving a better idea of what is visible when using the director's view. Motion curves are displayed by sampling the curve over its valid range and forming a polyline from the resulting positions. A moving trihedron is attached to the translation path to represent the rotational component of the curve.

To preview a sequence, the animator indicates the scene and shot within the preview window and selects the play button (see Figure 8). Initial start and end time for the shot are determined by the motion curves of objects in the scene. To change the length of the shot the animator either indicates different start and end times or alters the time increment used to sample the motion curves. If the start and end times are altered, this also affects where the motion curves are sampled, thus allowing some control over which portions of the motion curves are used. This also allows shots to overlap in time, so that the same action can be viewed multiple times from

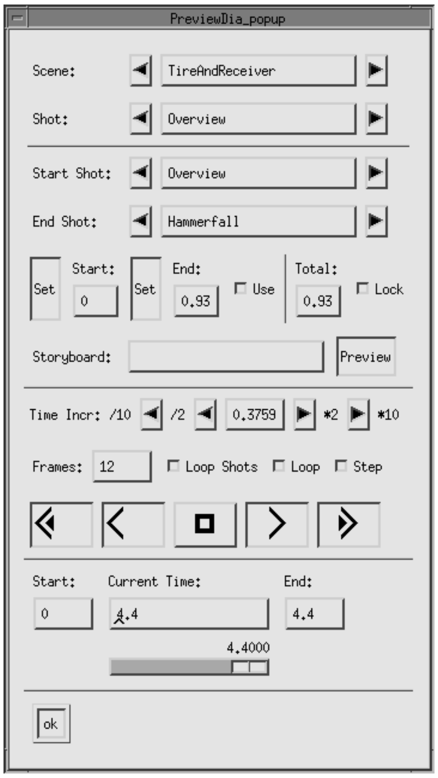

Figure 8: The preview control window.

the same or multiple cameras. For convenience the preview window contains the familiar set of "tape player" control buttons indicating stop, play forward, play backward, fast forward, and fast backward. There is also a slider bar for directly changing frames with the mouse. If the animator changes the time increment used to sample the motion curves, this changes the apparent length of the shot in preview mode but does not effect the final rendered frames.

As the animator modifies the start and end time for shots in the preview window these values are permanently stored in the shot object and are used when generating high quality frames for the final product. In this way, previewing and editing are merged into a tight interaction loop. When the animator is finished editing several shots, he can preview the entire sequence by selecting the starting scene and shot and the ending scene and shot. Shots without cameras are previewed from the viewing location last used to view that shot.

6.2 High Quality Rendering

The system supports high quality rendering by accessing the rendering tools provided by the Alpha_1 modeling system. There are two renderers provided by Alpha_1: a fast scanline algorithm and a more sophisticated ray tracer. Both of these utilities provide options for changing the image size and quality (i.e., anti-aliasing). The ray tracer also supports shadows, texture maps, material properties, and motion blur. The animator controls the renderers used, image size, and image quality through a rendering control panel.

The animator can render a storyboard (i.e., a set of single frames), shot, sequence of shots or the entire movie. The timing of an animated sequence can be controlled in several ways. By default, the length of a shot is determined by examining the motion curves of its objects. The start time is the time of the earliest motion and the end is the time of the last motion. As described above, the animator can alter these start and end times to shorten or lengthen the shot. Using the rendering control window, the animator can also indicate that a shot should last a particular length of time or contain a certain number of frames. Thus, shots may be scaled by allowing the animator to specify the actual length of the shot. For instance, the motion curves in a shot may specify that motion begins at time 0 and ends at time 5. The user may decide to actually start the shot at time 2 and end at time 4. When the shot is rendered, the number of frames generated is based on the 2 second time differential. If the animator requests that the shot last for 3 seconds the action slows down, if the shot lasts 1 second the action speeds up. When conflicts occur, the most recently specified value is used. In addition to the duration of a shot, the animator can specify the number of frames per second to generate.

When the animator finishes setting all rendering parameters, the system generates a Unix shell script to invoke the proper tools. This mechanism has several advantages. Foremost is that the actual rendering can be run at a later time in batch mode. It is important that the system not require X windows to be active during rendering. Clearly, it is unacceptable to require the animator be logged in during the time consuming process of rendering. Also, since the rendering process is a script, rendering can occur on a different host and at a different time. The script is structured to allow the frames of the film to be rendered in parallel with each frame being computed on different workstation.

7 An Animation Proof Reader

As animation becomes a "desk top" activity, there will be a growing need for tools which aid the novice animator in producing quality animations. One such tool is a proof reader which can identify awkward camera placement, motion sequences or scenes. Although there have been some attempts to incorporate cinema, stage, and dance knowledge with traditional computer animation[10, 13, 14, 18], the idea of an animation proof reader appears novel. For such a tool to be useful, we must first define a set of rules that are generally agreed to identify "bad" animation practice. From this set we must then devise algorithms or heuristics to uncover instances of these in an animated sequence.

Much of a film maker or animator's job is creative in nature. However, this creativity is filtered through a set of established rules in order to effectively communicate to an audience. These rules have been developed over the past 80 years by film makers and classical animators to ensure the clear presentation of a story[1, 16, 9]. A film maker who fails to obey them will irritate or confuse an audience. There are rules to help directors choose camera locations, shoot multi-person dialogue, and edit action sequences, among others. If these rules are not adhered to, even the most realistically rendered images of the most complex dynamic motion will likely result in an undramatic if not completely confusing animation. Consider a scene in which two actors are having a conversation, and as each character speaks, the audience sees a close up of the speaker's face. If these close ups are filmed so that the speakers appear gazing in the same direction, the audience will believe that the characters are talking to a third person and not each other! We have abstracted several of these rules as the basis for an animation proof reader. We describe here the set of rules derived from [1] which are amenable to computer analysis.

One film convention dictates that the center of attention (COA) should remain unoccluded in the foreground of a shot. The center of attention is the player that is the focal point of a shot. This focal point must be identified by the animator. To check this rule we relate what constitutes foreground to camera position and verify that the COA falls within an acceptable range from the camera. Next, determine surface occlusion using standard hidden surface algorithms.

Not only should the the COA be unobstructed, but the area behind the COA should remain relatively uncluttered. Similarly, the COA should be colored in a way that distinguishes it clearly from the background. This rule can be verified by checking that the color of the background and the COA are "significantly" different. These techniques ensure that an audience can easily identify the COA.

The line of interest (LOI) is defined to be a straight line in the direction of the COA's gaze or movement. As a general rule, a scene should only be filmed on one side of the LOI so as not to confuse the audience. Where the LOI can be identified (i.e., motion curves) this rule is easily checked by computing the dot product of the LOI and a vector from the COA to the camera. Where the LOI cannot be determined the animator may specify it.

Unfortunately, there are exceptions to this rule, which complicate matters. Directors use several tricks to gracefully switch sides of the LOI. Each of these methods relies on the fact that humans are easily fooled through visual tricks. One technique is to use a *cut-away* shot before making the switch. A cut-away is a diversionary shot that does not contain the COA and makes the audience forget the current viewing direction. For instance, if we wanted to switch sides during a scene in which a man is typing on a computer, we could film him from the right, use a cut-away shot of the screen he is typing on, and then film him from the left. If a scene has a player moving in a neutral direction (towards or away from the camera), we can also safely switch sides of the LOI. A player moving obliquely towards or away from the camera can also be used for the switch, however, the allowable range for this oblique angle can only be determined through heuristics.

Motion is a necessary component to any animation, but with indiscriminate use, the audience may not focus their attention on each shot's COA. The eye is attracted to a moving object in a still scene and to a still object in a busy scene[10]. So, the proof reader checks that if a shot's COA is moving, the rest of the shot's objects are generally still, and vice versa.

To ensure a smooth transition from one shot to another when animating motion, (see Figure 9) the animator has several options. These options introduce the concept of a *screen sector*. Screen sectors are defined by vertically dividing the screen into two or more parts. Motion may be repeated in the same screen sector either in the same or opposite direction. Or a moving player may use both half screens if one of the following guidelines are followed. Motion can begin and end at the center of the screen, or motion can start on one side and finish on the opposite side of the screen. Finally, movement may either converge towards or diverge from the screen center.

If the the motion from a single player is split into two shots, the player's speed should be similar in both shots. If this is not practical, then the speed in the second shot should increase. For those objects with known velocity, this rule consists of a few simple comparisons.

Our final rule concerns camera motion. In general, a lot of camera movement is distracting to an audience, so the animator is warned once some pre-determined threshold is reached. This threshold might be heuristically determined.

8 Current Status and Future Work

The animation production (with storyboards), camera placement, and property manager features of the story program are largely complete. The software includes support for X window and Silicon Graphics GL display devices and runs on a variety of hardware platforms. One previewing enhancement would allow the animator to preview the entire animation in a single window with each shot being displayed as a storyboard, wire frame, or final shaded images according to the best representation available. The camera positioning system, although adequate for many applications could be improved. We would eventually like to implement a more robust camera positioning

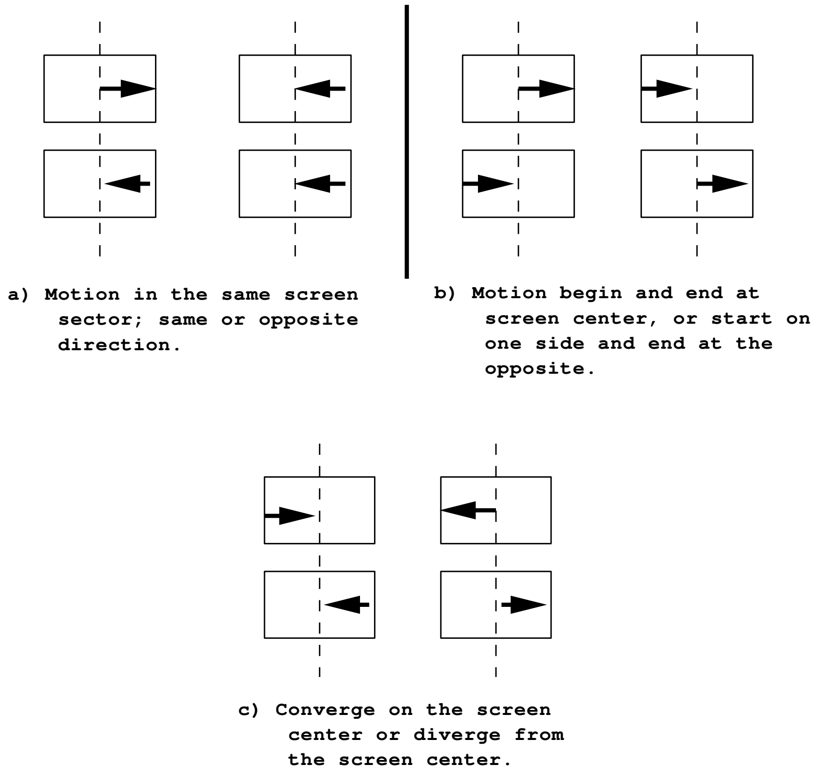

Figure 9: Options to ensure smooth shot-to-shot transitions.

system such as Mackinlay[11] or Gleicher[7] have developed.

An important enhancement to the previewing module would allow accurate timing control. Previewing is implemented by sampling the motion curves for each object in a frame, calculating the position of the object along its motion curve, and rendering the object with either the X or Silicon Graphics GL graphics libraries. When rendering is complete, iterate on the next time step. Although this algorithm produces smooth and pleasing motion, the apparent speed and duration in real time is dependent upon cpu speed and graphics performance. To properly handle timing while previewing, a system must monitor the update rate of the graphics display device and calculate the next frame according to this delay.

An important aspect of production animation is managing and integrating the sound track with the animation. Lack of audio support in computer animation is a significant departure from traditional animation techniques. In real production animation, the sound track is recorded before animation is actually rendered. The sound track provides critical timing constraints which drive the rest of production. We chose to omit this feature from our system due to the complexity of sound manipulation (i.e., recognizing word breaks, etc.) and the relatively poor support for digital sound on current workstations. Nevertheless, we feel an animation system sufficient for professional use must include such a facility.

The initial design of the animation proof reader is complete. The suggested rules are plausible in both their utility to the novice animator and in their tractability of implementation.

9 Conclusions

We present an integrated animation and storyboarding system which simplifies the creation of multi-scene animations. Our system models both the process of animating a story and the product of the animation by introducing objects for movies, scenes, shots, and storyboards. These objects help animators quickly sketch out the structure of a story and provide a framework for step-wise refinement of that structure into a complete film. Camera positioning is simplified by several intuitive placement techniques and "through the lens" or "director's" views of the scene. The models used in the animation are managed by a flexible property manager which allows easily substituting one model for another or for automatically generating simple approximations for complex B-spline models. The rendering module manages previewing, editing, and the rendering of high quality final frames. Finally, the animation proof reader identifies awkward views or motion sequences and provides advice to novice animators. The system integrates a diverse collection of tools into a consistent animator's workbench. We believe our system makes computer animation more accessible to the traditional animator and film maker as well as making animation a more useful communication tool.

10 Acknowledgments

We gratefully acknowledge the Alpha_1 group for providing an environment conducive to this work. We thank Michael Cohen, Jamie Painter, Beth Cobb, Robert McDermott, Elaine Cohen and Gershon Elber who provided valuable comments on early drafts of this paper. The storyboards used in the accompanying video were drawn by Elena Driskill. Jim Rose produced the accompanying video. Finally, Mike Blum would like to thank Richard Riesenfeld, Robert Kessler, and Gary Lindstom for their continued support.

References

[1] Arijon, D. *Grammar of the Film Language.* Hastings House, 1976.

[2] Catmull, E. The problems of computer-assisted animation. In *SIGGRAPH '78 Conference Proceedings* (1978), pp. 348–353.

[3] Cohen, M. F. Interactive spacetime control for animation. In *SIGGRAPH '92 Conference Proceedings* (1992), pp. 293–302.

[4] Computer Science Department, University of Utah. *Alpha_1 User's Manual*, June 1991.

[5] Dovey, D. Integrating geometric modeling and motion simulation for mechanical systems. Master's thesis, University of Utah, December 1990.

[6] Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. *Computer Graphics: Principles and Practice, Second Edition.* Addison-Wesley Publishing Company, 1990.

[7] Gleicher, M., and Witkin, A. Through-the lens camera control. In *SIGGRAPH '92 Conference Proceedings* (1992), pp. 331–340.

[8] Gracer, F., , and Blasgen, M. W. Karma: A system for storyboard animation. In *Proceeding Ninth Annual UAIDE Meeting* (1970), pp. 210–255.

[9] Karp, P., and Feiner, S. Issues in the automated generation of animated presentations. In *Graphics Interface '90* (1990), pp. 39–48.

[10] Lasseter, J. Principles of traditional animation applied to 3d computer animation. In *SIGGRAPH '87 Conference Proceedings* (1987), pp. 35–44.

[11] Mackinlay, J. D., Card, S. K., and Robertson, G. G. Rapid controlled movement through a virtual 3d workspace. In *SIGGRAPH '90 Conference Proceedings* (1990), pp. 171–176.

[12] Magnenat-Thalmann, N., and Thalmann, D. *Computer Animation: Theory and Practice*. Springer-Verlag, 1985.

[13] Ridsdale, G., and Calvert, T. The interactive specification of human animation. In *Graphics Interface '86* (Vancouver, 1986), N. Marceli Wein, Evelyn M. Kidd, Ed., pp. 121–129.

[14] Ridsdale, G., and Calvert, T. Animating microworlds from scripts and relational constraints. In *Proceedings Computer Animation* (Geneva, 1990), pp. 107–116.

[15] Thingvold, J. A. Elastic and plastic surfaces for modeling and animation. Master's thesis, University of Utah, March 1990.

[16] Thomas, F., and Johnson, O. *Disney Animation: The Illusion of Life*. Abbeville Press, 1984.

[17] Ware, C., and Osborne, S. Exploration and virtual camera control in virtual three dimensional environments. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics* (Snowbird, Utah, 1990), pp. 175–184.

[18] Withrow, C. A dynamic model for computer-aided choreography. Tech. rep., University of Utah, June 1970.