A Two-View Approach to Creating Computer Animation

Wenhui Guo
Institut für Informatik
Universitätsgelände Am Flughafen, Geb. 051
D-79110 Freiburg, Germany
E-mail: guo@informatik.uni-freiburg.de

Abstract: In this paper, we report about the design and implementation of a new interactive animation system. The system is based on the idea that all objects occurring in animation sequences should have two views:the structural view and the shape view. Using our system, the animator can create the two views simultaneously without programming and can manipulate objects by sending messages. Thus, the system is based on an interactive, object-oriented data model implemented on the Macintosh.

1 Introduction

The process of generating computer animation can be divided into three phases: object definition, motion specification and rendering. The purpose of object definition is to describe the appearance of an object and its relation to other objects. Motion specification describes how the object moves and changes during animation display. Rendering computes a 2D image of the scene for each frame in the animation sequence from the viewpoint of the camera.

To define geometric objects, current animation systems provide one of two techniques: interactive or procedural. Using the procedural technique, it is possible to define both monolithic and composite objects by describing them as an object hierarchy. However, it is often difficult for the animator to describe composite objects and access their structures using interactive techniques. The main problem is that current interactive animation systems have no efficient or expressive tools to describe and manipulate an object hierarchy.

To specify motions, many current animation systems [5,3,6] offer the animator one or more animation techniques (or specification methods): task-level, guide(or interactive) and programming [8]. Few interactive animation systems, however, enable the animator to use all of these techniques to define motions for a node of an object hierarchy in a combined and interactive manner. The major problem in specifying motions is that current interactive animation systems lack a powerful user interface that can seamlessly integrate various motion specification methods.

A potential solution to these problems is to replace the traditional user interface of interactive animation systems with a structural system environment following the object-oriented design philosophy. Our objective was to develop an interactive animation system called SINAS (**S**tructural **IN**teractive **A**nimation **S**ystem) which can integrate different animation techniques and allow the animator to present objects in an intuitive and structural way and to manipulate objects at a high level of abstraction.

SINAS is an ongoing research effort to develop interactive animation tools for non-programmer authors [4,1]. Our work has concentrated on the development of an object-oriented animator interface (OOAI), with which we wish to replace the traditional animator interface. With this new interface, the animator can present an object as a shape view and a structural view and attach different animation techniques to nodes of structural views graphically.

2 A Two-View Object for the Animator

The theoretical foundation of our two-view approach is what we call the SSM model (**S**hape view, **S**tructural view and **M**essage) that converts an object hierarchy into an interactive object-oriented model consisting of three separate but closely related components: the structural view, the shape view and messages. The structural view represents the semantic aspect of objects at hand, whereas the shape view represents the syntactic aspect, and messages are actions performed on both views. The core of the SSM model contains the two-view objects and three-level messages as discussed below.

2.1 Two-View Objects

A composite object consists of low-level objects, which in turn are composed of lower-level objects and so on. Therefore, a composite object can be presented as an object hierarchy. An object in a hierarchy is composed of geometric primitives and inclusions of lower-level subobjects. Our idea is to separate object hierarchies into structural and primitive levels. The structural level contains structural elements with relationships between objects, and the primitive level consists of primitive elements. Both levels can be displayed separately with two views on the screen: a structural and a shape view. A structural view is actually a hierarchy diagram. A shape view of an object shows what it looks like in the real world. Figure 2 shows how a car can be decomposed into two levels and two views.

In general, the structural view has the following characteristics:

1. Each node is named or labeled with a text string or an icon denoting an object. The node may be related to kinematic, dynamic and physical parameters such as position, orientation, color, velocity, force and torque, etc.

2. Each node is associated with a local coordinate system and a transformation matrix. A coordinate system of a lower level node in the structural view is located in that of a higher level node. This implies that a motion specified on a higher level node can be inherited by lower level nodes.

3. Each node can contain geometric models that may be defined by a CSG representation or by geometric primitives such as polygons, lines or a piece of an image.

4. Each node is represented by a set of lists including the primitive list, the motion list, the subobject list and so on. These lists are used to store different message types sent by the animator.

Since an object has two views, the modelling an object needs to create the two views. Specifying motions means attaching motions to nodes of the structural view. Updating a scene can be considered as a process of traversing structural views or portions of them.

The animator can communicate with an object through its two views. The shape view allows the animator to manipulate shapes at a primitive level, just as one does with a drawing program. The structural view, on the other hand, enables the animator to access a hierarchical structure and to manipulate shapes at different hierarchical levels.

2.2 Three-Level Messages

In object-oriented programming, a message is passed between objects or between the user and objects. The SSM model is concerned with only these messages passed by the animator to two-view objects. The role of messages is to define objects' appearances and movements. To define motions, we have designed and implemented three-level animation techniques (task level, guiding level, and animator level) [8] as message types. First, simple task-level commands are used to define a behavior. This level also includes some traditional motion control methods such as easy-in, easy-out. At this level, we have programmed some typical functions and equations and parameterized them and placed the parameters on form fillins.

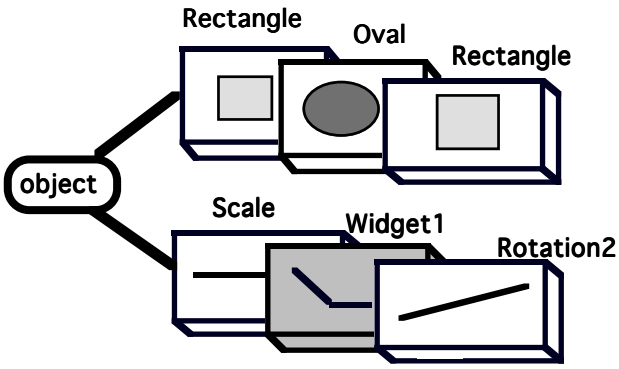

Figure 1: An object retains messages sent to it.

The animator can then choose from the functions, set proper parameter values. Second, different guiding-level techniques such as spline editors are available. Finally, we implement a simple animator-level scripting language. Different message types are implemented as different message classes, all of which contain the start times and durations as private variables.

Before a message is sent to an object, the message may need to be edited. Editing a message may involve using a text editor, spline editor, or a form fillin, etc. For example, creating a scripting-level message involves using a text editor, whereas creating a task-level-command message requires the animator to employ a form fillin. Messages, when passed by the animator to an object, may be retained by the receiving object. Figure 1 illustrates the situation. Interpretation of the messages gives the object appearances and behaviors.

3 Creating Computer Animation

3.1 Defining Objects

The best way to explain how the animator can create two-view objects and define motions for them is to take an example. In Figure 2, we describe how to create the 2D scene shown. Let us define the two views of a car. Our car consists of a body and two wheels. The editing process of the structural view is shown in Figure 3. The other objects in the scene of Figure 2 can be generated in the same way.

This editing process is supported by a syntax-controlled structure editor. The main feature of the editor is its ability to suggest to the animator in advance a potential object position in the structural view.

Once the structural view of an object is constructed, the animator can begin to present the object's shape view by drawing the object' shapes in the shape editor. SINAS provides the drawing tools for designing shapes.

3.2 Specifying Motions

After the scene is populated by two-view objects, we can send messages to the structural view of an object and watch what happens to its shape view. Sending an object a message is done simply by clicking a node in the structural view and editing the message by selecting a proper tool from the Motion menu or the Operation menu. Let us now watch Figure 2 and send messages to **car**. If we want **car** to drive, we can send the Translate message to **car** and the Rotate messages to **wheel1** and **wheel2**. We can see **car** immediately driving across the screen after we click one of the display buttons on the control panel shown on the top right in Figure 2. If the desired motion is not activated, we can change the parameter values of the message or remove the message and choose a new one.

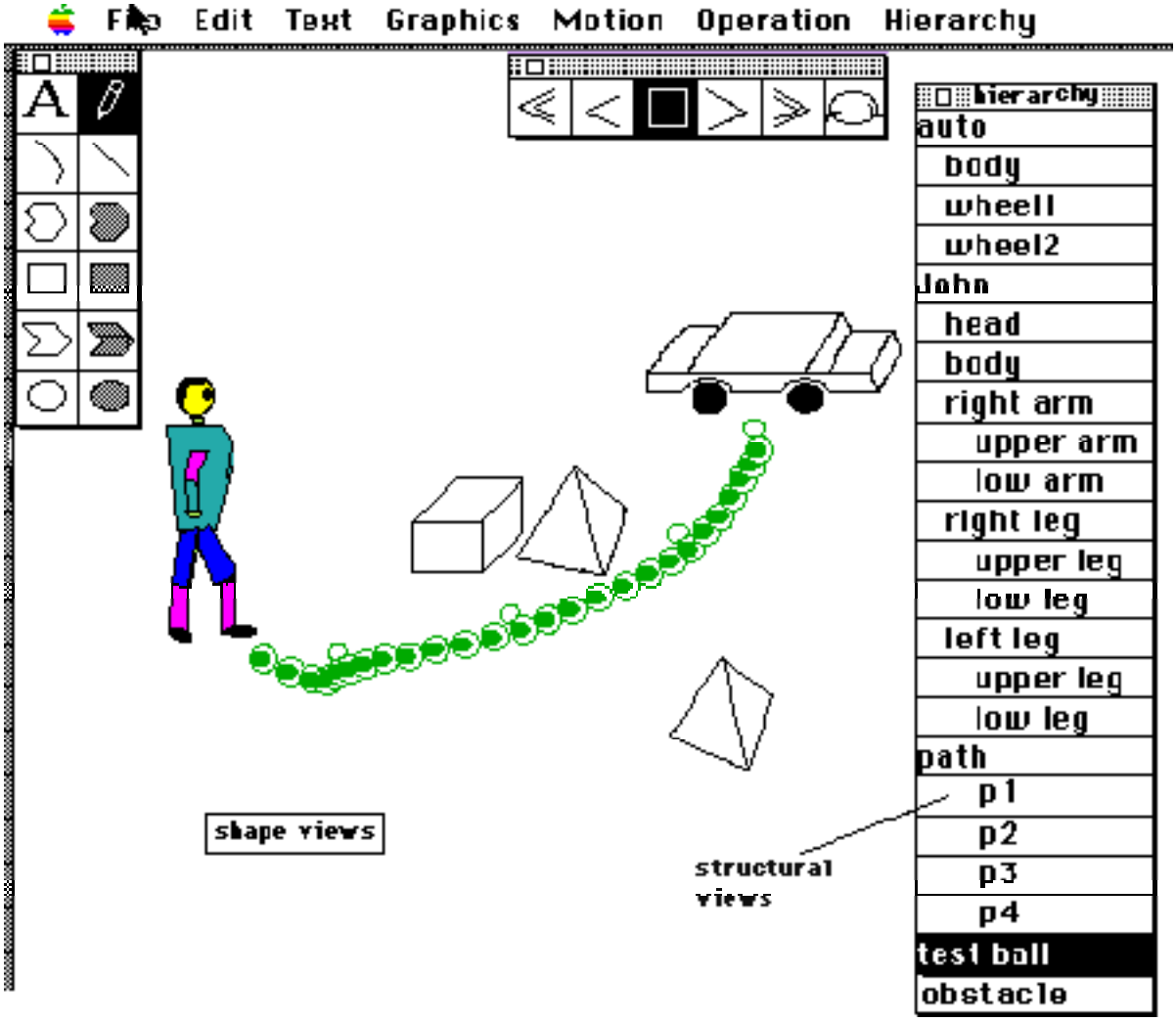

Figure 2: John walks to the car.

Now let's watch **John** and define motions for him. For example, passing the message "swing" to his **left leg**, we can see immediately **left leg** swinging. The same message with different parameter values can be sent separately to **John**'s other joints in order to compound a complex motion like "walk".

It is important to note that we edit and send these messages at a higher level without the concerned details. For example, the Rotate message sent to **wheel2** contains a parameter "center of rotation". Obviously, **wheel2** should rotate around its own axis. Therefore, we set the parameter value to"car/wheel2" that we get from the structural view. The system computes precisely the center point of car/wheel2, i.e., **wheel2** and passes it to the computation model of rotation. Let us consider another example. We send **John** the Goal message that contains a parameter "destination". If the parameter value is set to **car**, we can see **John** moving to **car** along a default path. If **car** is driving at the same time, we can see **John** moving by tracking the moving car.

If the default path is not satisfactory, we can plan a new path by using other animation techniques which SINAS provides. Let us design interactively a special path consisting of several control points. We first register names of the path and points in the structural view and then draw these points along the path in the scene, as shown in Figure 2. **Path** in the structural view is used as a parameter value of the message named Kochanek Curve [2] that interpolates the path. Before passing the message to **John**, we send this message to the object named **testball** in order to see the path clearly. The trace of the **testball** is shown in Figure 2. From this trace, we know how quickly **John** should walk. The above examples show

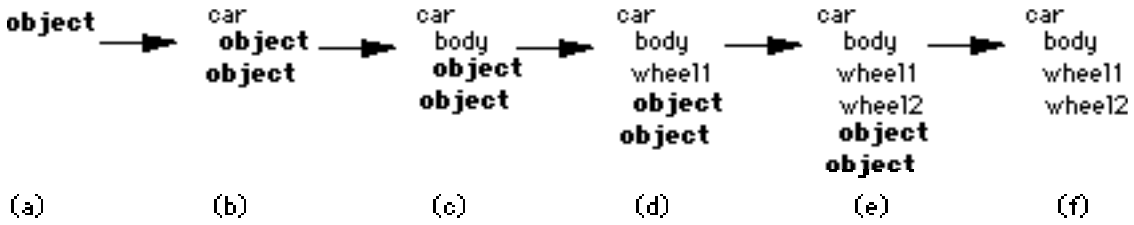

Figure 3: Snapshot sequence of generating the structural view of our car

that it is very easy to create animations using our system.

3.3 Output Format

The animation sequence created as described above can be saved as a bitmap file in the pict format, or as a SGML file. The system can be extended to include more common graphics or animation file formats so that animation sequences created via our system can be used in various hypermedia presentation tools such as Macromind Player. The purpose of using SGML (standing for Standard Generalized Markup Language) in our system is to save the specified animation information for further modification or data interchange. In addition, the file together with the file produced by TRAIN [1] can be interpreted to present a hypermedia document.

4 Message Interpretation

Each time the animator selects an object in its structural view and clicks one display button, the process of interpreting messages retained in the object and its subobjects begins. The message interpreter of SINAS traverses the structure with the selected object as the top node in the preorder. Our interpreter works in two steps. First, it builds an object display list from the structure. It visits every object in the structure and checks if its motion list is empty. If the list is not empty and the object is located at an internal node, all motions in the list are transferred to a lower-level object and inserted into the front positions of a motion list of the lower-level object. A lower-level object therefore accumulates more and more motions in its motion list during the traversal phase. If the object has an empty primitive list, the object is not placed into the display list. As the result of the first step, a display list is obtained that contains only objects with both non-empty motion and primitive lists.

In the second step, our interpreter traverses the object display list repeatedly where each traversal produces a frame of the animation sequence. It checks each motion in the motion list of an object and activates the motions where start times have been reached. Motions are interpreted with respect to each primitive of the object's primitive list. The detail process of interpreting motion messages is much like the one discussed in [7]. When a primitive is transformed by a motion, our interpreter draws it onto an offscreen bitmap. After all objects in the object display list have been visited, the interpreter copies an offscreen bitmap onto the current bitmap. The previous image of primitives is erased and a new image is generated so that the dynamic illusion is obtained.

During this traversal time, the animator may interact with the object through a widget interface (see Figure 1). If there are animator interactions, the interpreter switches to process the animator interaction event and interprets the message which the animator sends. If there are no animator interactions, it continues to carry out the traversal process.

5 Conclusion

In this paper, we have presented a two-view approach to creating animations. The approach consists of the SSM model and the two-view animator interface.

The SSM model is an interactive object-oriented model for the animator. This model states that each object has two views and that the animator can interact with the two views and pass three-level messages to an object through its two views. The animator interface allows the animator to create two-view objects and to edit and pass three-level messages.

In this paper, we have established a framework for the design of animations. Until this point, we have developed only a few algorithms for geometric models and motions. However, the system is fully operating already in its current status. To improve our current system, the next objective is to develop and implement more algorithms. For object definition, we shall develop algorithms for fusion of solids, merging of 2D object shapes, and image compositing. For motion specification, we shall develop algorithms for motion descriptions such as collision avoidance.

Acknowledgement The author would like to thank Th.Ottmann for his support, reviewing and modifying this manuscript.

References

[1] Friedrich Augenstein, Thomas Ottmann, and Juergen Schoening. Logical markup for hypermedia document:the train-system. In Hermann Maurer, editor, *Educational Multimedia and Hypermedia Annual, 1993*, pages 17–25, Spring Verlag, June 23-26 1993.

[2] Doris H U. Kochanek and Richard H. Bartels. Interpolating splines with local tension, continuity, and bias control. In *the ACM SIGGRAPH, Computer Graphics*, pages 33–41, ACM SIGGRAPH, Addison–Wesley Publishing Company, 1984.

[3] Peter C. Litwinowicz. Inkwell: a 2 1/2-d animation system. In *the ACM SIGGRAPH, Computer Graphics*, pages 113–122, ACM SIGGRAPH, Addison–Wesley Publishing Company, 1991.

[4] Thomas Ottmann. Elektronische kurse-kritische bilanz und konsequenzen fuer die werkzeugentwicklung. In P.Gorny, editor, *Informatik und Schule 1991*, pages 1–15, Spring Verlag, Oldenburg, Germany, June 23-26 1991. Informatik-Fachbericht, Nr.292.

[5] William T. Reeves, Eben F. Ostby, and Samuel J. Leffler. The menv modelling and animation environment. In *The Journal of Visualization and Computer Animation*, pages 33–40, January 1990.

[6] Nadia Magnenat Thalmann and Daniel Thalmann. *Computer Animation:Theory and Practice.* Addison-Wesley Publishing Company, 2 edition, 1990.

[7] Robert C. Zeleznik, D.Brookshire Conner, Matthias M Wloka, Daniel G. Aliaga, Nathan T. Huang, Philip M. Hubbard, Brian Knep, Henry Kaufman, John F Hughes, and Andries van Dam. An object-oriented framework for the integration of interactive animation techniques. In *the ACM SIGGRAPH, Computer Graphics*, pages 105–111, ACM SIGGRAPH, Addison–Wesley Publishing Company, 1991.

[8] David Zeltzer. Towards an integrated view of 3-d computer animation. *Visual Computer*, 249–259, 1985.