THE JOURNAL OF VISUALIZATION AND COMPUTER ANIMATION, VOL. 6: 49–65 (1995)

Object-Oriented Motion Abstraction

BİLGE ERKAN AND BÜLENT ÖZGÜÇ
Department of Computer Engineering and Information Science, Bilkent University, Bilkent, 06533 Ankara, Turkey

SUMMARY

An important problem in the production of an animation sequence is the great amount of information necessary to control and specify the motion. Specification of complex animation sequences with a smaller amount of information is possible if they are built over some abstracted sequences. Abstraction supports dealing with complexity by structuring, so that the necessary features are made available while those that are not necessary are hidden. In our work, motion abstraction is used to build complex animation sequences with the help of object-oriented concepts. A parametric key-frame interpolation method is used for producing the in-between frames of an animation sequence. The parameters that define the motion of a model, in our work, are position, orientation, size, shape and colour. Orientation transformations are implemented by unit quaternions.

KEY WORDS Object-oriented animation Parametric key-frame interpolation Motion abstraction

1. INTRODUCTION

Both traditional and computer animation require great amounts of human labour to generate an animation sequence. An important problem in the production of an animation sequence is the great amount of information necessary to control and specify the motion. If the scene gets more complicated and more realistic, the user faces difficulties due to the amount of information that must be provided to the system. A good approach to this kind of problem is the use of an abstraction mechanism.[1] Different levels of abstractions can be used to decrease the amount of information to generate complicated animation sequences. At higher levels, the details decrease because the most useful features are abstracted while the features that are not needed are hidden, hence less information that is built over lower levels in a step-wise manner will suffice for complex ideas. If motion is considered as a building block and abstractions are made on motion, then many different kinds of motion can be obtained, and this makes the animation richer and more sophisticated. A sequence created at the lowest abstraction level with good and detailed control of the animator can be used later, whenever it is needed, as a building block for a new sequence. If thought of recursively, this abstraction mechanism allows the creation of complex and detailed motions with dependencies among them, easily and without much detailed information except for those at the lowest level. This gives the user the opportunity to reuse previously generated productions, in conjunction and relation with newer ones, in a well-defined abstraction.

A very helpful idea for computer animation and motion abstraction is the idea of object-orientation. Many implementations have been done for computer animation using object-oriented paradigm, as in References 1–7. The object-oriented idea facilitates both the im-

CCC 1049–8907/95/010049–17

Received 29 October 1993
Revised 5 June 1994

plementation of an animation program and the generation of animation sequences. It is thus advantageous, useful and natural to use object orientation in computer animation. Our work mainly deals with motion abstraction through the object-oriented paradigm to decrease the amount of information needed to specify complicated animation sequences.[8] The system is implemented in C++[9] under XView,* and it provides an interactive environment for the specification of motion.

2. OBJECT-ORIENTED ANIMATION

Many implementations have been done for computer animation using the object-oriented paradigm, as in Clockworks,[2, 5] SOLAR,[1] Pinocchio,[6] and in others described in References 3, 4 and 7. Some of these have been influenced by earlier systems that carry object-oriented ideas. The important ones among these earlier systems are ASAS (Actor/Scriptor Animation System)[10] and Mira.[11]

ASAS is a high-level animation language, an extension of a Lisp-based actor system and essentially object-oriented. It supports abstractions and a form of adaptive motion control. Actors pass messages to others for synchronization, and this facilitates the adaptive motion. The Mira system is based on structured programming and data types. It consists of some data types such as animated basic types, actor types and camera types. It provides some abstraction levels similar to ASAS but does not support message passing, however, synchronization can be provided through the use of common parameters. Our system provides an abstraction mechanism for motion and some transformable object types as actors, cameras and lights similarly.

These two systems carry the initial ideas of abstraction and object orientation for computer animation. The later systems are obviously more object-oriented. Fiume *et al.* have developed a temporal scripting language for object-oriented animation.[4] They have provided an environment in which animated objects are fashioned into complex animations using a concise, directly executable specification language. Their language facilitates specifying a global temporal behaviour and constraining local temporal behaviour to meet the specification. This is done in a similar way in our system as well; however, we provide an interactive environment instead of a scripting language. Another language, SOLAR (Structured Object-oriented Language for AnimatoRs), provides high-level abstractions and adaptive motion through the class inheritance and message passing mechanisms of the object-oriented paradigm.[1] They make abstraction levels that enable an animation sequence to be defined in steps.

Pinocchio is an animation system that controls human motion with some sequencing facilities.[6] A general movement dictionary has been developed to classify movements. Tokens in the motion dictionary are elementary movements that are not decomposable into submovements. Elementary movements are building blocks and can be combined to describe complex motions. The time constraints in a sequence are relative times, the actual starting time of a scene is set by a director object and hence absolute times are generated. Chmilar *et al.* have built a software architecture integrating the data structures for 3D modelling and animation so that time-based models that can change shape during animation can be described.[3] In their approach, they integrate modelling and animation processes which makes an orthogonal system where anything: position, orientation, size, shape and structure, can be animated. Our system supports shape and geometrical structure change during motion as well. In our system the geometrical structure change is accomplished by the motion

* XView is a trademark of Sun Microsystems, Inc.

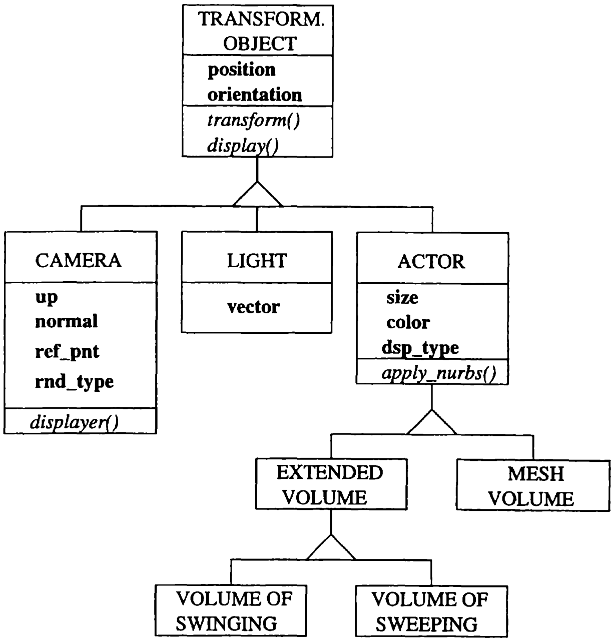


Figure 1. Transformable object class and its subclasses

abstractions on sequences where hierarchically dependent sequences can be constructed. As this dependence is done at the motion level, it can be changed through time.

Clockworks is an object-oriented computer animation system with integrated capabilities such as modelling, image synthesis, animation, programming and simulation.[2, 5] It has been implemented in portable C under Unix with object-oriented features such as objects with methods, instances, class hierarchies, inheritance, instantiation and message passing. Clockworks is made up of simple basic tools that can be combined. Zeleznik et al. have presented an interactive modelling and animation system that facilitates the integration of a variety of simulation and animation paradigms.[7] Their system extends modelling tools to include animation controls and provides the modelling of objects that change in shape, appearance and behaviour over time. Their system provides indications that next generation of graphics systems is headed towards an environment with time and behaviour as first-class notions rather than shape description and rendering as in the earlier systems.

3. CLASSES IN THE MOTION ABSTRACTION SYSTEM

In this section the triangle sign denotes inheritance, the diamond sign denotes aggregation and the dot sign denotes multiplicity in associations. This style is taken from the Object Modeling Technique (OMT) by Rumbaugh *et al.*[12]

The *transformable object* class is one of the biggest classes in the system, and has an important role because it is the highest abstraction of the animating elements. It has the subclasses *actor*, *camera* and *light*, as seen in Figure 1. *Actor* is the parent class of the graphical objects. It has two subclasses, *extended volume* and *mesh volume*, and the subclass *extended volume* has two more subclasses, *volume of swinging* and *volume of*

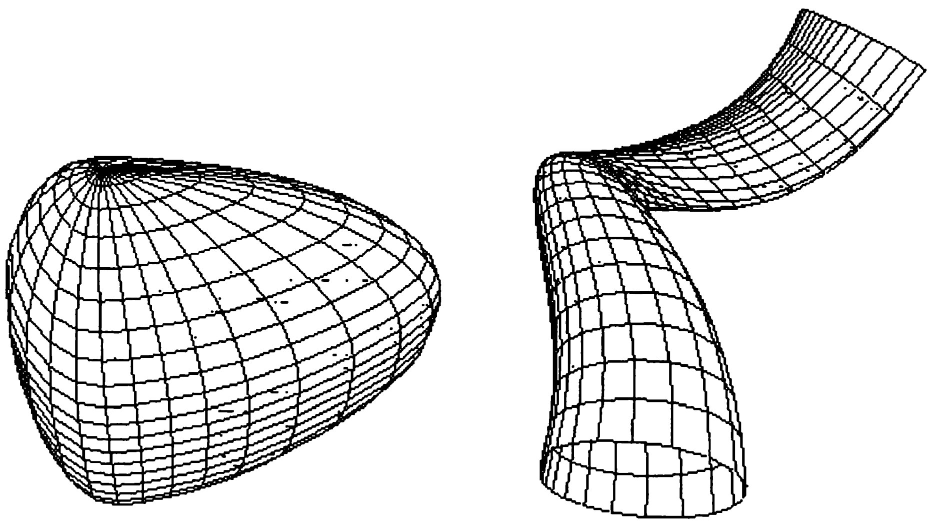


Figure 2. Volumes of swinging (left) and sweeping (right)

sweeping. Transformable objects are actors, cameras or lights. Two kinds of light objects are designed, one is a transformable object which has the same properties as an actor and the other one is a point light source, again a transformable object but having no geometry. The light source which has a geometry is used as a light emitting object in rendering by radiosity methods, whereas the other one is for the other shading techniques such as constant, Gouraud and Phong shading that are provided by our system. By sending the frames to the *radiosity* renderer developed on the IPSC/2 Hypercube in Bilkent University, more realistically rendered images can be obtained.[13] An animation sequence rendered in this way can be seen in Figure 14.

An actor can be defined either as a network of 3D data supplied externally, or it can be defined as a 3D volume extended from 2D, such as a volume of swinging or volume of sweeping[14] as seen in Figure 2. If preferred the 3D geometry of an actor can be smoothed by fitting NURBS (Non-Uniform Rational B-Splines)[15] on it.

The *frames* class is the main component of a motion. It contains the key values of the animation parameters, their key times and their interpolated values according to those key times. There are three levels of hierarchy in the *frames* class, as seen in Figure 3. The highest class, seen at the lower left of the figure, is for the transformable objects, so it has the two animation parameters, *position* and *orientation*, its subclass is for actors and has the extra animation parameters, *size* and *colour*, and finally the lowest subclass, seen at the lower right of the figure, is for extended volumes and it has an extra *shape parameter* indicating that the extended volumes can change shape. As the low-level motion in our system is supplied by the parametric key-frame interpolation method, the key values for the position, size, colour and shape parameters given at the key frames are interpolated through an interpolation scheme providing the respection of the key values. The orientation parameter is interpolated through a special interpolation scheme for quaternions called spherical linear interpolation.[16,17]

The most important class of our system is the *sequence* class because it is the highest abstraction of the animation sequences. As seen in Figure 4, the *sequence* class has two

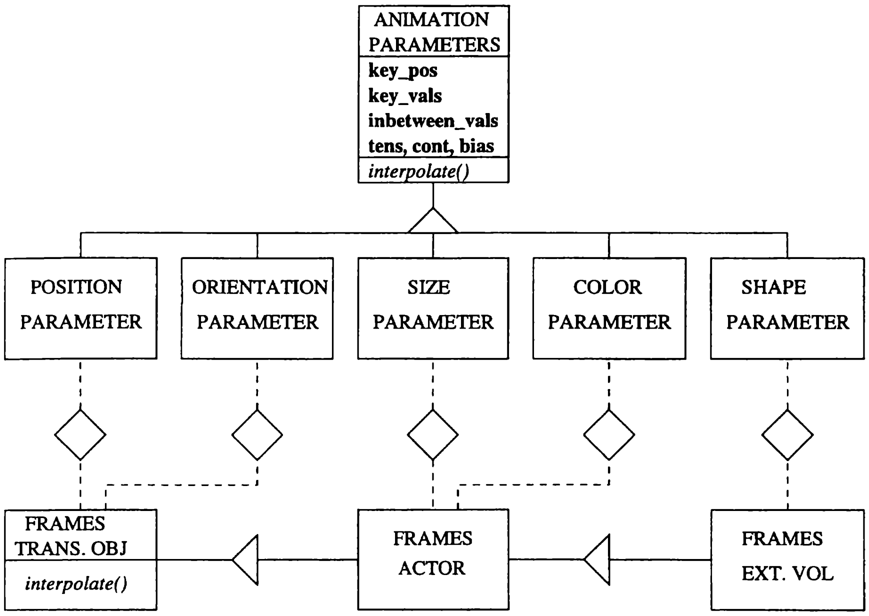


Figure 3. Frames and animation parameters classes

major subclasses, *simple sequence* and *compound sequence. Simple sequence* has one subclass, *combined sequence.* The *compound sequence* class has two subclasses, *temporally* and *spatially dependent sequences.* A compound sequence is made of two sequences as is denoted by the recursive aggregation in Figure 4. There are three subclasses of temporally dependent sequence, *concatenated, simultaneous* and *overlapped* sequences, and two subclasses of spatially dependent sequence, *guided* and *hierarchic* sequences. Sequences will be explained in detail in Section 4.

In Figure 5, some important associations and aggregations between classes are presented. *Animation parameters* are parts of a *frames* object, a frames object and a transformable object are parts of a simple sequence and finally, the *controller* controls the sequences and has references to the current camera, light and background objects through the sequence.

4. SEQUENCES

In our system high-level controls are provided with the help of motion abstraction and object orientation. These high-level controls provide the user with an easy-to-use environment for generating highly complex animation sequences in a recursive manner based on lower level abstractions that play the role of building blocks.

The sequence class supports the motion abstraction concept. In Figure 6, some visual representations for each sequence class are given. These visual representations help in a way for visualizing what we mean by these different sequence types. In the left column the temporally dependent sequences and in the right column spatially dependent sequences are seen. Temporally dependent sequences are displayed in a time co-ordinate because they are generated by combining the time properties of their subsequences, whereas the spatially

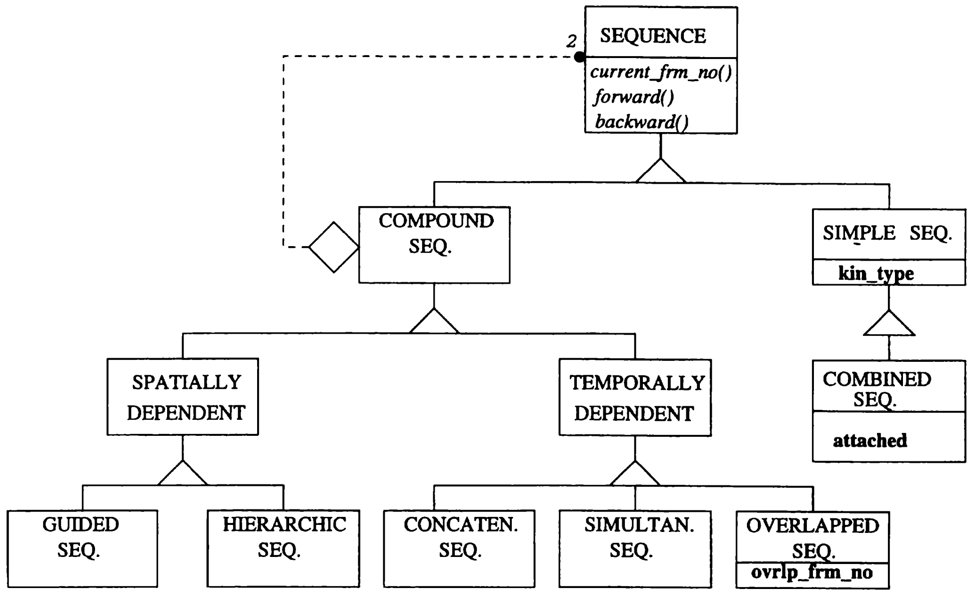


Figure 4. Sequence class and its subclasses

dependent sequences are generated by combining their space properties, such as position, orientation, etc. As seen in Figure 4, a sequence is either a simple sequence or a compound sequence. A compound sequence is made of two sequences which can again be simple or compound sequences. Compound sequences can be spatially or temporally dependent where these are kinetic dependencies. Building dynamic dependencies among sequences in this way is not suitable yet. A sequence is displayed in a recursive manner according to the properties of its subsequences. These properties are explained in detail in the following subsections.

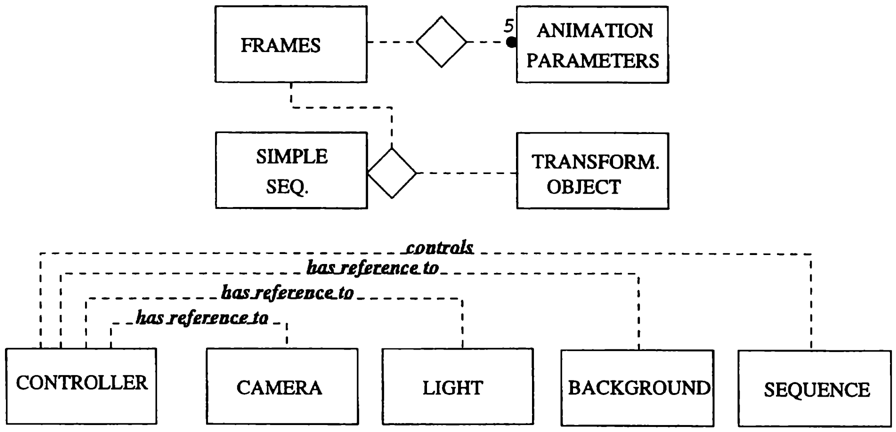


Figure 5. Relations among some of the classes

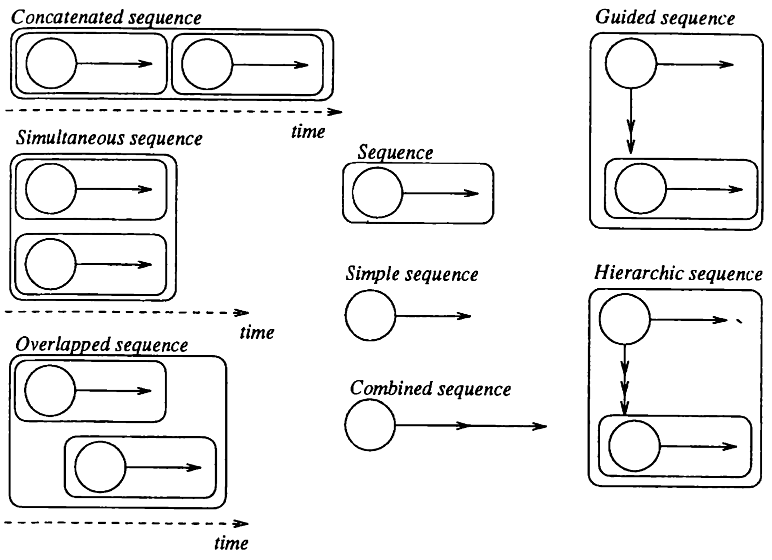


Figure 6. Visual representations of sequences

4.1. Simple sequence

Simple sequence is the basic kind of motion in our system. It consists of a transformable object and frame data as seen in Figure 5. In the frame data the key values and the key times of the animation parameters used in this sequence are specified. When a simple sequence is specified with these attributes, the in-between values of its related animation parameters are generated by interpolation.

A simple sequence generated for a transformable object can be used for any other transformable object. This is achieved by applying the values of the animation parameters specified for a transformable object to another transformable object which is structurally similar, i.e. both are extended volumes or both are actors, in our terms.

Some important attributes of the simple sequence class are total frame number, denoting the total number of frames that this sequence will be displayed, and current frame number, denoting the current frame that the sequence is at locally.

When a simple sequence is forwarded, its current frame number attribute is incremented. Here the current frame number can be treated as a time point during the sequence. Then, the in-between value for each parameter contributing to the sequence, for this current frame, is retrieved from the previously interpolated in-between values in the frame data of the sequence, and these parameter values are applied to the transformable object that this sequence acts on and the transformable object is displayed. This process continues until the local current frame number of the sequence reaches its total frame number, and hence the sequence ends.

A combined sequence is a simple sequence generated by the combination of two simple sequences. They can be combined in two ways. The first way is attached combination, where the second simple sequence is spatially combined to the end of the first simple sequence. In this case, the second sequence starts from the position, with the orientation and size that the first sequence has ended. The second way is non-attached combination, where the two simple sequences are combined by substituting some smoothing in-betweens between the last key of the first sequence and the first key of the second sequence. In both cases, as a result a single simple sequence will be generated that is a combination of the two original sequences, where the combination is done by providing a smooth transition between them. This concept of smooth transition between motions has been dealt with in Reference 18, in a manner called *phrased transition*.

4.2. Compound sequences

Compound sequences are high-level abstractions that provide a base for many other abstractions in the sequence hierarchy. A compound sequence is composed of any two independent sequences, either simple or compound. Every simple sequence in a compound sequence has its own transformable object and frame data that contains the values of the animation parameters. As seen in Figure 4, there is a recursive aggregation on the compound sequence. This recursive aggregation shows that a compound sequence consists of any other two sequences where these sequences can again be any compound sequence or a simple sequence. Hence there is a recursive definition on this class that leads to a recursive definition for keeping these sequences in a sequence library. Therefore, if a compound sequence is defined by two other sequences, only the names of these sequences are kept, and in order to generate it, these sequences are instantiated according to their own definitions. Recursively instantiating all the sequences in this manner, results in the lowest level sequences, simple sequences, being reached. This idea of recursion fits very well to motion abstraction. Generating higher level sequences using the lower level ones in a recursive manner supports motion abstraction easily, concisely and precisely.

4.2.1. Temporally dependent sequences

These sequences are compound sequences created according to the temporal behaviours of their building block sequences. The building blocks of such a sequence have their own spatiotemporal behaviour and behave according to their local information. However, this local behaviour of the building sequences is converted into a global behaviour with temporal dependence. This is achieved in terms of time. The temporal behaviour of building sequences is according to relative times. Absolute times for these sequences are generated when they are joined as temporally dependent sequences. Hence a global temporal behaviour is generated at this level without touching the spatial behaviours of the individual sequences. In Figure 7, some examples of temporally dependent sequences can be seen. In temporally dependent sequences the object is displayed during the time interval the sequence is defined, otherwise it disappears.

Concatenated sequence. This is a temporally dependent compound sequence, made up of two independent sequences, inheriting the features of its parent class, the compound sequence. The two independent sequences are made temporally dependent under the abstraction of the concatenated sequence where the second sequence starts its execution just after the first one ends. Local key times of the key animation parameter values of the building sequences are converted into global times according to concatenation. The local times of the first sequence are not affected, whereas those of the second sequence are shifted to the end of the first one. The current frame number of a concatenated sequence is the

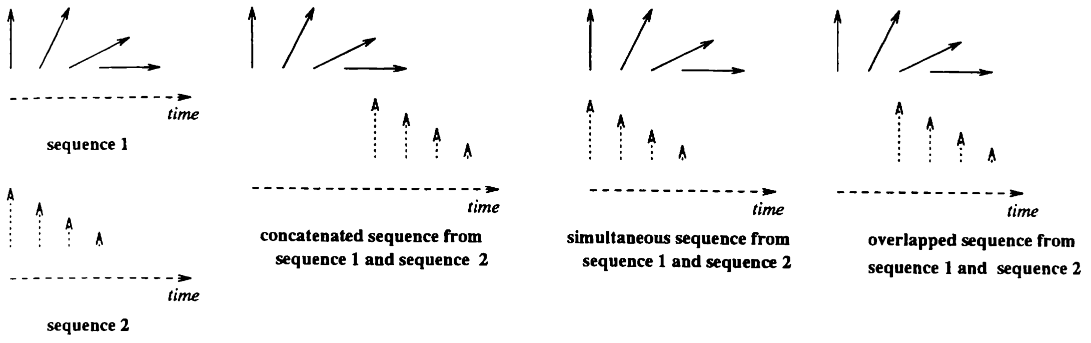

time
sequence 1
time
sequence 2
time
concatenated sequence from sequence 1 and sequence 2
time
simultaneous sequence from sequence 1 and sequence 2
time
overlapped sequence from sequence 1 and sequence 2

Figure 7. Temporally dependent sequences

current frame number of its first sequence if it has not yet reached its end. Otherwise it is the sum of the total frame number of its first sequence and the current frame number of its second sequence, meaning that the first sequence is finished and the concatenated sequence is currently in its second sequence. The total frame number of a concatenated sequence is the sum of the total frame numbers of its building sequences. The concatenated sequence is displayed by displaying the frames of its first sequence until it ends and then of its second sequence, as the local times of its second sequence are shifted to the end of the first sequence generating a global temporal behaviour for the whole sequence.

Simultaneous sequence. This is a temporally dependent compound sequence where the two sequences start execution at the same time. The local key times of both the building sequences are converted to global times without any change. A simultaneous sequence is displayed by displaying the frames of both of its sequences at the same time. Such a sequence ends when the longer of its sequences ends. The current frame number of a simultaneous sequence is the current frame number of its longer sequence. And the total frame number of a simultaneous sequence is the total frame number of its longer sequence.

Overlapped sequence. This is a temporally dependent compound sequence where the second sequence overlaps the first sequence starting execution at the specified overlapping frame number. Such a sequence is displayed by displaying the frames of its first sequence until the overlapping frame is reached. After that point, displaying the frames of both of the sequences continues. It ends when the longer of its sequences ends. If it is currently executing its first sequence, then the current frame number of an overlapped sequence is the current frame number of its first sequence. Otherwise, it is the sum of its overlapping frame number and the current frame number of its second sequence. The local key times of the second sequence is shifted to the overlapping frame to generate its global temporal behaviour. An example of an overlapped sequence can be seen in Plate 2.

4.2.2. *Spatially dependent sequences*

These sequences are compound sequences created according to the spatial behaviours of their building block sequences. The building blocks have their own spatiotemporal behaviour, but under the control of such a sequence their spatial behaviours get modified. The spatially dependent sequences have one of their building blocks as a simple sequence. This

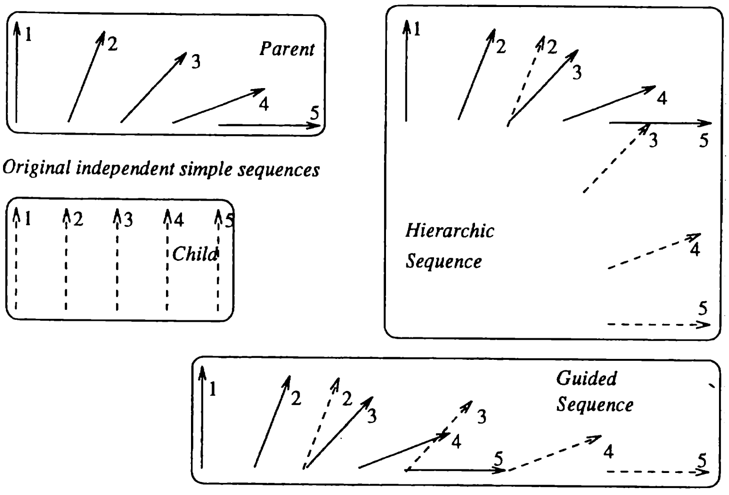


Figure 8. Spatially dependent sequences

simple sequence plays the parent role, whereas the other building block can be any kind of sequence playing the child role. The spatial behaviour of the child sequence is modified according to the spatial behaviour of the parent sequence. The temporal behaviour of spatially dependent sequences is not modified, so the global temporal behaviour is exactly same as the local temporal behaviour of the subsequences. The frames of these sequences are displayed simultaneously until the parent sequence ends. The current and total frame numbers depend only on the parent sequence. In Figure 8, some examples of spatially dependent sequences can be seen. In this Figure the sequence drawn by dotted lines is the child sequence. The sequences are represented in space co-ordinates and the numbers over the arrows indicate frame numbers to give the notion of time in the sequence. The child and parent start at the same position, so in the first frames the child arrow cannot be seen since it coincides with the parent arrow.

Guided sequence. A guided sequence is a spatially dependent compound sequence where the spatial behaviour of the child sequence is modified so that it is being guided by its parent. Figure 8 shows examples of spatially dependent sequences. As seen in this Figure, in a guided sequence the child sequence is affected by the size, orientation and position parameters of its parent. These parameters of its parent are contributed to its corresponding parameters, so that the total values of those parameters at a frame i is as follows:

$$\begin{aligned} \text{result\_size}_i &= \text{size}_i \times \text{parent\_size}_i \\ \text{result\_orientation}_i &= \text{orientation}_i \times \text{parent\_orientation}_i \\ \text{result\_position}_i &= \text{position}_i + \text{parent\_position}_i \end{aligned} \tag{1}$$

These parameters are applied to a point in $3D$ in the order of size, orientation and position as follows:

$$((\text{point} \times \text{result\_size}_i)\%\text{result\_orientation}_i) + \text{result\_position}_i \tag{2}$$

where the % operator is for rotating a point by a quaternion.

Hierarchic sequence. This is the other spatially dependent compound sequence. This behaviour is exactly the hierarchic behaviour seen in a linked body. However, in our system the definition for linking bodies to generate hierarchic action is done at the sequencing level. An advantage of this definition mechanism is that the linkage of the bodies in animation is not defined statically, such that any different linkage may be defined with the help of motion abstractions at a sequence. Thus, geometrical structure can change during a sequence. The definition of the child sequence is very easy with the help of the motion abstraction mechanism. However, this approach is not powerful enough for defining a precise linked body and its motion, as human animation systems support. As seen in Figure 8, in a hierarchic sequence, the child sequence is affected by the size, orientation and position parameters of its parent in a hierarchic manner. These parameters of its parent are contributed to the corresponding parameters of the child and a complete 4×4 transformation matrix is obtained as follows:

$$T = [\text{orientation}_i][\text{position}_i][\text{parent\_orientation}_i][\text{parent\_position}_i] \tag{3}$$

The quaternion that represents the resultant orientation is obtained from this matrix by converting the upper left 3×3 part, the rotational part, into a quaternion. The resultant position and size are the same as in guided sequence and all these values are applied in the same manner. An example of a hierarchic sequence can be seen in Plate 1.

With the help of the hierarchic sequence definition and an on/off control that can be set at any time interval of a simple sequence, an initial setting can be given to the child sequence. The parent sequence can be set with some parameters and switched off at its time interval, and when the child sequence is joined to this sequence hierarchically, the parent will not be displayed but will provide an initial setting to the child sequence. In this way a sequence can be executed starting at a desired position, with desired orientation and size.

5. USER INTERFACE

There are two major environments in our system. These are the graphical object creation environment and the sequence generation environment as seen in Figure 9. The graphical object creation environment has two subparts which are the curve design and extended volume creation environments. The sequence generation environment also has two subparts which are the key-framing and sequencing environments.

An interactive editing facility is provided in the key-framing environment, seen in Figure 10, that supports editing the key values of the parameters easily. The parameters that can be edited are position, orientation, size, colour and shape. Any key value for these parameters can be adjusted for any key frame. The adjusted key value is applied to the transformable object and displayed to give feedback as seen in the display window in the Figure. In this manner, the system provides a flexible, extensible and modular environment in which the user can work comfortably.

The key frames generated in the key editing environment are used to generate a simple sequence in the sequence generation environment. Any simple sequence generated in this manner or any compound sequence can be used to generate any other compound sequence, as was explained in Section 4.2. In Figure 11, a simultaneous sequence is being created using a previously created hierarchic sequence of the object jellybon and a simple sequence of the floor. A few steps of this resultant sequence can be seen in Figure 12. The objects are displayed as control points and in smoothed versions respectively in the frames displayed

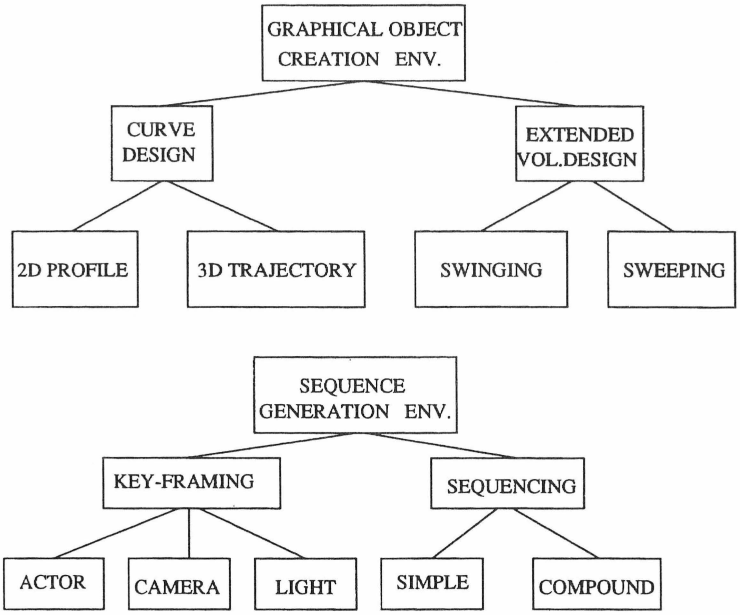
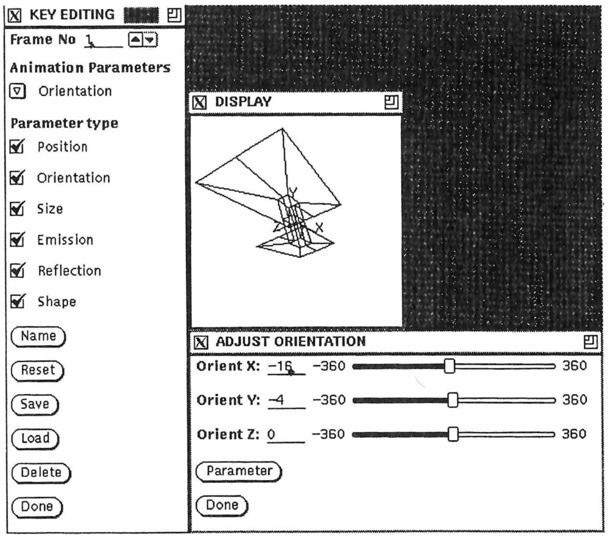


Figure 9. Structure of the system



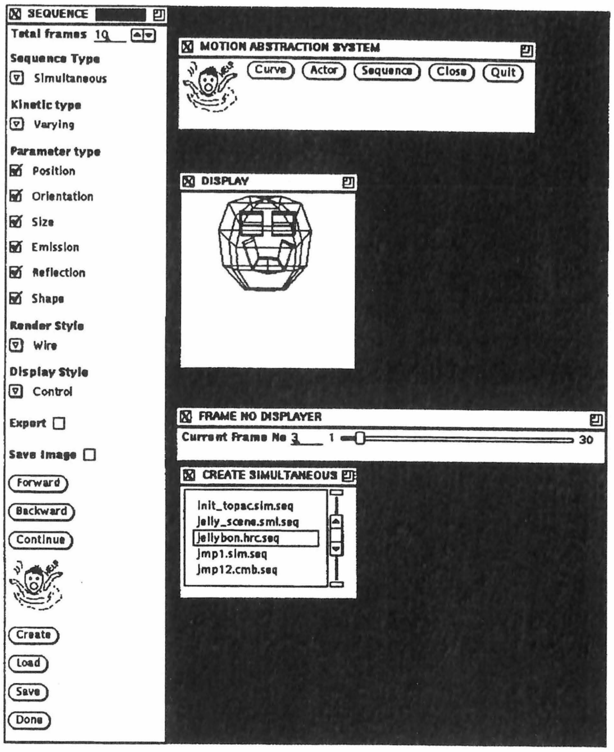
Frame No 1

Animation Parameters
Orientation

Parameter type
☑ Position
☑ Orientation
☑ Size
☑ Emission
☑ Reflection
☑ Shape

Orient X: -16 -360 360
Orient Y: -4 -360 360
Orient Z: 0 -360 360

Figure 10. Key editing environment


SEQUENCE
Total frames 10
Sequence Type: Simultaneous
Kinetic type: Varying
Parameter type: ☑ Position ☑ Orientation ☑ Size ☑ Emission ☑ Reflection ☑ Shape
Render Style: Wire
Display Style: Control
Expert ☐
Save Image ☐

FRAME NO DISPLAYER
Current Frame No 3 1 30

CREATE SIMULTANEOUS
init_topacsim.seq
jelly_scene.sml.seq
jellybon.hrc.seq
jmp1.sim.seq
jmp12.cmb.seq

Figure 11. *Sequence generation environment*

in the left and right parts of Figure 12, respectively.

In Figure 13, some key frames defined during the key editing phase of the animation Jellybon are displayed. As seen here, the simple sequences of head, mouth, eyes and floor are generated separately and independently by defining the required key frames. After these simple sequences are generated, they are made compound, spatially or temporally, in the sequence generation environment. The simple sequences of the mouth and eyes are made hierarchically dependent on the simple sequence of the head, and then this hierarchic sequence is made simultaneous with the simple sequence of the floor. This results in the sequence seen in Plate 1.

6. CONCLUSION

Our system is a motion abstraction system designed and implemented using object-oriented techniques. Object-orientation is advantageous for animation both in the implementation of the system and the generation of the animation sequences. New sequence or transformable object classes can be inserted in the system in their hierarchy.

For the specification of the animation sequences, scripting languages are not used. Such languages might indeed be very useful for giving concise and accurate definitions. However they are not easy to use for naive users since they require experience of programming and introduce parsing overhead. Instead, an interactive environment is provided at all levels that gives ease, flexibility and speed during specification, at the cost of providing concise and

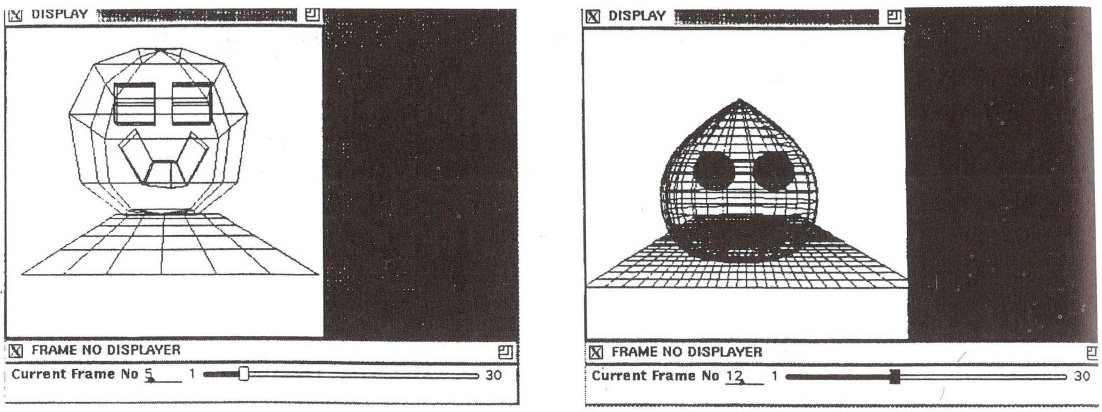


Figure 12. Some frames from a simultaneous sequence

accurate definitions. Both scripting and highly interactive systems have their advantages and drawbacks. Neither approach is superior over the other, as mentioned in Reference 19. It is often useful to have a collection of different approaches.

The advantages of motion abstraction in an animation system come in the specification phase of the animation sequences. Creating realistic and complex animation sequences generally requires a great amount of information. By motion abstraction, the amount of information given at an abstraction level decreases considerably, being built on lower levels in a stepwise manner. Hence, in our system motion abstraction is provided to decrease the effort for creating a complex animation sequence by supporting control with less information at increasingly higher levels while still giving enough control at the lowest levels as well. This brings reusability of existing sequences and flexibility in the creation of new ones with concise and precise definitions. Hence, the conciseness of scripting is managed in some sense by abstraction. The created sequences, key-frames or graphical objects, briefly everything created in the system, can be saved in their specific libraries to support reusability. As every high-level specification is based on low-level specifications, only the names of the lower level building blocks and their dependencies are kept for a high-level specification in the libraries.

Quaternions are used instead of Euler's angles or matrix representation of rotations that cause some overheads. Quaternions are compact and require fewer calculations for concatenating and applying rotations than the matrix representation, and conversions to and from quaternions are well defined.

Two important kinds of dependencies are introduced between sequences. These are temporal dependency and spatial dependency. In the first one, local temporal behavioured sequences are joined and their global temporal behaviour is generated according to their temporal dependency. In the second one, not the temporal behaviour but the spatial behaviour of the sequences is modified according to their dependency. These two kinds of dependencies can generate many complex cases. However, no dynamic dependency is defined in our system yet, therefore, such motions cannot be generated.

The hierarchic sequence in our system is not as effective as the human animation systems in the definition of precise linked bodies and their complex motions including dynamics. However, this idea brings flexible definition of objects, parts and their motions and non-

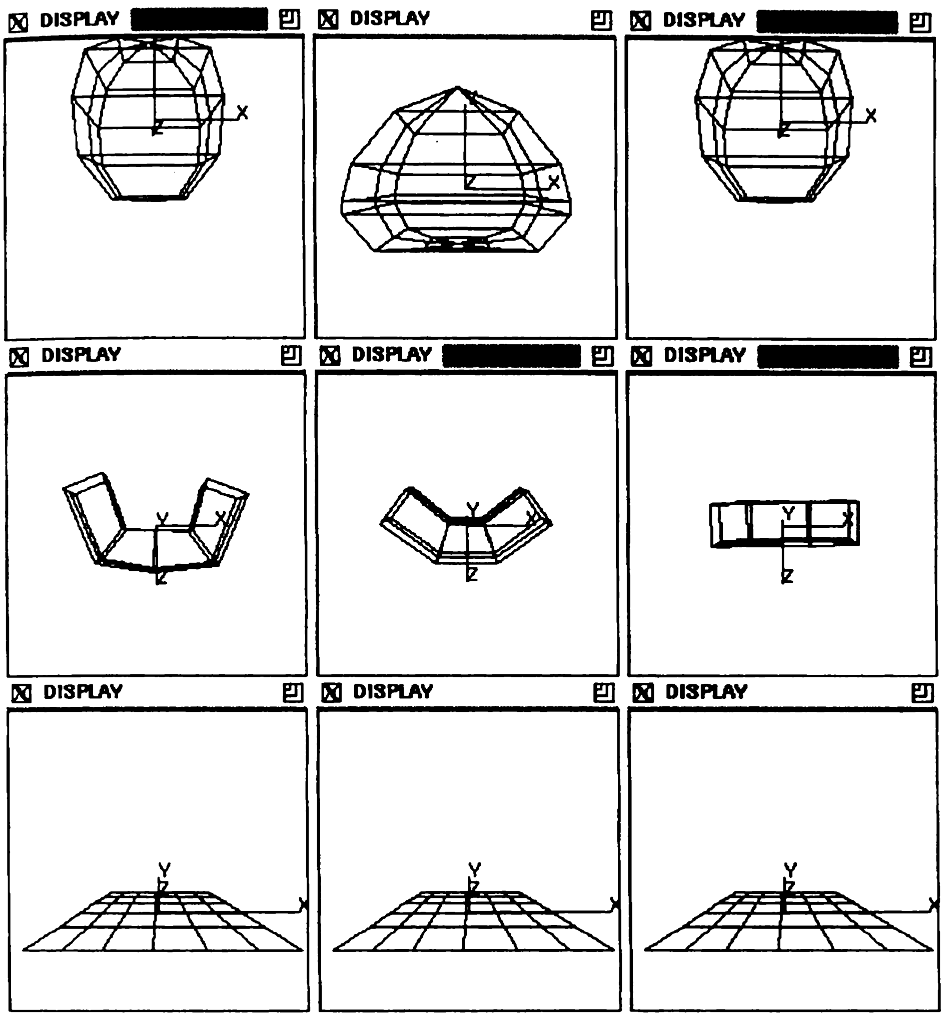


Figure 13. Some keys frames from the animation Jellybon

static linkages among parts by means of defining hierarchy at the sequencing level. The definition of such a sequence can be seen in Figure 13.

If compared with the projects discussed in Section 2, our system resembles that of Reference 4 in handling motion abstraction, generation of complex animation with global behaviour based on local temporal behaviour (a similar approach is also used in Reference 6), and discrimination of static representation of graphical objects from how they are animated. It differs in the part that no scripting language is employed. Furthermore, it resembles the method of Reference 1 in using abstraction levels that enable animation sequences to be defined in steps; however, our abstraction levels do not reach to functional and character abstractions. As in Reference 6, elementary movements are building blocks to describe complex motions, and as in Reference 7 shape and structure can change, as described earlier.

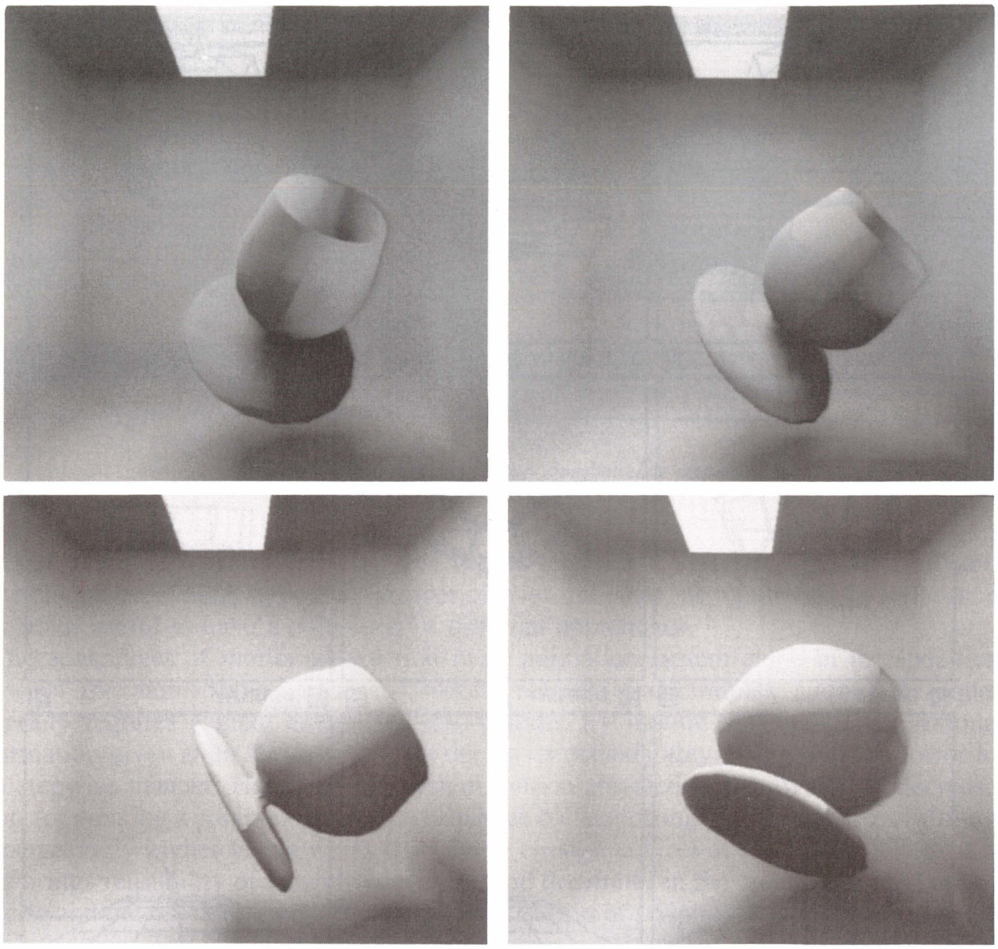


Figure 14. Frames from the animation Glass rendered by radiosity

This motion abstraction system provides an environment to generate flexible, reusa concise and precise animation sequences with the help and advantage of motion absı tion in an interactive environment, avoiding the use of complex (but powerful) scrip languages with only small sacrifices.

7. EXAMPLES

The sequence seen in Plate 1 presents a hierarchic sequence generated by making the sir sequences of mouth and eyes hierarchically dependent on the simple sequence of the h Then this hierarchic sequence is made simultaneous with the simple sequence of the fl Hence the resultant sequence is a simultaneous sequence generated by making a sir sequence and a hierarchic sequence temporally dependent as a simultaneous sequence. head and mouth have shape changes in their simple sequences.

In Plate 2, the sequence of a spin is used as building block and an overlapped sequ is generated by two of these sequences. Then this sequence is overlapped with the sequence once again. The initial positions and orientations of the spins are given by hierarchic dependence and on/off control as explained in Section 4.2.2. The resultant overlapped sequence is made simultaneous with the sequence of a light source that changes its orientation from pointing to the lower left to pointing to the lower right. The motion of spin is generated by providing appropriate key frames at key times, hence this motion is controlled only by kinetic control.

Figure 14 presents the sequence of a glass rendered by the radiosity renderer developed on the IPSC/2 Hypercube in Bilkent University.[13] On the ceiling, there is a light source that has a geometry.

REFERENCES

1. T. S. Chua, W. H. Wong and K. C. Chu, 'Design and implementation of the animation language SOLAR', in N. Magnenat-Thalmann and D. Thalmann (eds.), *New Trends in Computer Graphics, Proceedings of CG International*, Springer Verlag, 1988, pp. 15–26.
2. D. E. Breen, P. H. Getto, A. A. Apodaca, D. G. Schmidt and B. D. Sarachan, 'The clockworks: an object-oriented computer animation system', in G. Marechal (ed.), *Eurographics '87*, North Holland, 1987, pp. 275–282.
3. M. Chmilar, B. Wyvill and C. Herr, 'A software architecture for integrating modeling with kinematic and dynamic animation', *The Visual Computer*, **7**, 122–137 (1991).
4. E. Fiume, D. Tsichritzis and L. Dami, 'A temporal scripting language for object-oriented animation', *Eurographics '87*, ed., G. Marechal. North Holland, 1987, pp. 283–294.
5. P. Getto and D. Breen, 'An object-oriented architecture for a computer animation system', *The Visual Computer*, **6**, 79–92, (1990).
6. R. Maiocchi and B. Pernici, 'Directing an animated scene with autonomous actors', *The Visual Computer*, **6**, 359–371, (1990).
7. R. C. Zeleznik, D. B. Conner, M. M. Wloka, D. G. Aliaga, N. T. Huang, P. M. Hubbard, B. Knep, H. Kaufmann, J. F. Hughes, and A. van Dam, 'An object-oriented framework for the integration of interactive animation techniques', *SIGGRAPH '91, Computer Graphics*, **25**, (4), 105–112, (July 1991).
8. B. Erkan, *Object-Oriented Motion Abstraction*, Master's thesis, Department of Computer Engineering and Information Science, Bilkent University, Ankara, Turkey, September 1993.
9. B. Stroustrup, *The C++ Programming Language, 2^{nd} Edition*, Addison-Wesley, 1991.
10. C. Reynolds, 'Computer animation with scripts and actors', *SIGGRAPH '82, Computer Graphics*, **16**, (3), 289–296, (July 1982).
11. N. M. Thalmann, D. Thalmann and M. Fortin, 'Miranim: an extensible director-oriented system for the animation of realistic images', *IEEE Computer Graphics and Applications*, **5**, (3), 62–73, (March 1985).
12. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, 1991.
13. T. K. Çapın, C. Aykanat and B. Özgüç, 'Progressive refinement radiosity on ring-connected multicomputers', *Proceedings of IEEE Visualization'93 Parallel Rendering Symposium (San Jose, California)*, 71–76, (1993).
14. V. Akman and A. Arslan, 'Sweeping with all graphical ingredients in a topological picturebook', *Computers & Graphics*, **16**, (3), 273–281, (1992).
15. L. Piegl, 'On NURBS: a survey', *IEEE Computer Graphics & Applications*, **11**, (1), 55–71, (January 1991).
16. D. Pletincks, 'The use of quaternions for animation, modeling and rendering', *New Trends in Computer Graphics, Proceedings of CG International*, eds., N. Magnenat-Thalmann and D. Thalmann. Springer Verlag, 1988, pp. 44–53.
17. K. Shoemake, 'Animating rotation with quaternion curves', *SIGGRAPH '85, Computer Graphics*, **19**, (3), 245–254, (July 1985).
18. S. N. Steketee and N. I. Badler, 'Parametric keyframe interpolation incorporating kinetic adjustment and phrasing control', *SIGGRAPH '85, Computer Graphics*, **19**, (3), 255–262 (1985).
19. W. T. Reeves, E. F. Ostby and S. J. Leffler, 'The Menv modelling and animation environment', *The Journal of Visualization and Computer Animation*, **1**, 33–40 (1990).