

OBJECT-ORIENTED MOTION ABSTRACTION

A THESIS SUBMITTED TO
THE DEPARTMENT OF COMPUTER ENGINEERING AND
INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

by


Bilge Erkan

September 1993

TR
897.7
.E75
1993

OBJECT-ORIENTED MOTION ABSTRACTION

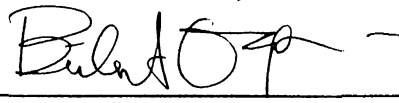
A THESIS SUBMITTED TO
THE DEPARTMENT OF COMPUTER ENGINEERING AND
INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BİLKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE


Bilge Erkan
Candidate in the Department of Computer Engineering and Information Science

by
Bilge Erkan
September 1993

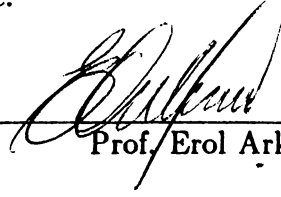
TR
897.7
.E75
1992

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Prof. Bülent Özgüç (Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Prof. Erol Arkun

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Dr. Aral Ege

Approved by the Institute of Engineering and Science:



Prof. Mehmet Baray,
Director of the Institute of Engineering and Science

ABSTRACT

OBJECT-ORIENTED MOTION ABSTRACTION

Bilge Erkan

M.S. in Computer Engineering and Information Science

Supervisor: Prof. Bülent Özgüç

September 1993

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 less 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 by the help of object oriented concepts. Parametric key-frame interpolation method is used for producing the in-between frames of an animation sequence. In this technique, the parameters of the model are interpolated for smooth in-betweens. The parameters that define the motion of a model, in our work, are position, orientation, size, shape and color. Orientation transformations are implemented by unit quaternions. Sufficient and good kinetic control provides a good illusion of dynamics, so timing, slow-in and slow-out controls are being supported.

Keywords: Object-oriented animation, parametric key-frame interpolation, motion abstraction, quaternions

ÖZET

NESNEYE YÖNELİK HAREKET SOYUTLAMASI

Bilge Erkan

Bilgisayar Mühendisliği ve Enformatik Bilimleri Bölümü

Yüksek Lisans

Tez Yöneticisi: Prof. Bülent Özgüç

Eylül 1993

Bir animasyon dizisi üretiminde önemli bir problem hareketi tanımlamak ve denetlemek için verilmesi gereken çok yoğun bilgidir. Karmaşık animasyon dizilerinin üretimi, bu dizilerin bazı soyutlanmış diziler üzerine kurulmasıyla, daha az miktarda bilgi vererek olasıdır. Soyutlama, önemli özelliklerin ön plana çıkartılıp önemli olmayanların gizlenmesi şeklinde bir yapılaşma ile karmaşıklıkla başa çıkılmasını sağlar. Bizim çalışmamızda, nesneye yönelik kavramlar yardımıyla, karmaşık animasyon dizileri hazırlanması için hareket soyutlaması yapılmıştır. Bir animasyon dizisinin ara çerçevelerinin üretiminde parametrik anahtar-çerçeve interpolasyon tekniği kullanılmıştır. Bu teknikte, akıcı ara çerçeveler üretilmesi için modelin parametreleri interpolate edilir. Çalışmamızda, bir modelin hareketini ifade eden parametreler, yer, yön, büyüklük, şekil ve renktir. Yön değişimleri birim *quaternion*lar ile gerçekleştirilmektedir. Yeterli ve iyi bir kinetik denetim sayesinde dinamik hareketin iyi bir hayali görünümü verilebilir. Bu nedenle zamanlama ve yavaş giriş, yavaş çıkış denetimleri sağlanmıştır.

Anahtar kelimeler: Nesneye yönelik animasyon, parametrik anahtar-çerçeve interpolasyonu, hareket soyutlaması, *quaternion*lar

ACKNOWLEDGEMENT

I am grateful to my advisor Prof. Bülent Özgüç for his guidance and support during the development of this thesis. I would like to thank Prof. Erol Arkun and Dr. Aral Ege for their remarks and comments on the thesis and Prof. Yılmaz Tokad for his valuable contributions to the mathematical aspects.

I would also thank Aydın Ramazanoğlu and Müjdat Pakkan for their efforts in taking and printing the photographs and slides, and all my friends for their valuable discussions and morale support.

I am grateful to dear Cem Yüceer for his sentimental and morale support and the Yüceer family for their morale support. I have deep gratitude to my parents and my brothers for everything they did to bring me here.

Contents

1	Introduction	1
2	Animation	4
2.1	An Overview of Traditional and Computer Animation	4
2.1.1	Key-Frame Animation	5
2.1.2	Parametric Key-Frame Interpolation	5
2.1.3	Procedural or Script Based Methods	6
2.1.4	Dynamic Simulation	6
2.2	Motion Abstraction	7
2.3	Object-Oriented Animation	8
2.4	Object-Oriented Design	13

3	Motion Abstraction System: Low Level Controls	15
3.1	The Classes in the Motion Abstraction System	16
3.2	Transformable Objects	23
3.2.1	Actors	23
3.2.2	Cameras	27
3.3	Animation Parameters and Their In-betweening	29
3.3.1	Interpolation Scheme	29
3.3.2	Representation and Interpolation of the Orientations by Quaternions	34
3.3.3	Interpolation of Shape	38
3.3.4	Interpolation of Color	42
3.4	Kinetic Control	44
4	Motion Abstraction System: High Level Controls	47
4.1	Simple Sequence	49
4.1.1	Combined Sequence	50
4.2	Compound Sequences	51
4.2.1	Temporally Dependent Sequences	52

CONTENTS	iii
4.2.2 Spatially Dependent Sequences	57
5 Conclusion	61
A Derivations	64
A.1 Conversion from a rotation matrix to a unit quaternion	64
A.2 Conversion from a unit quaternion to a rotation matrix	65
A.3 Derivation of the spherical linear interpolation formula	66
A.4 Rotation of a vector by a quaternion	67
B Colored Images	68

List of Figures

3.1	Structure of the system	16
3.2	Lowest level classes	17
3.3	Spline class	18
3.4	Curve class	18
3.5	Transformable object class and its subclasses	19
3.6	Frames and animation parameters classes	20
3.7	Sequence class and its subclasses	21
3.8	Relations among some of the classes	22
3.9	A volume of swinging	26
3.10	A volume of sweeping	26
3.11	World and viewing coordinate systems.	27
3.12	The specification of vector \vec{n}	34
3.13	Shape interpolation by changing the weights of the control points	39

3.14	Shape interpolation by changing the control points	39
3.15	Mapping of three curves and their in-betweening	40
3.16	Determination of the angle of a control point	41
3.17	Algorithm for mapping two curves	42
3.18	Varying number of in-betweens	44
3.19	Constant number of in-betweens	45
4.1	Visual representations of sequences	48
4.2	Spatially dependent sequences	59
A.1	Derivation of the <i>slerp</i> formula	66
B.1	Frames from the animation Jellybon	69
B.2	Frames from the animation Spin	70
B.3	Frames from the animation Sunny	71
B.4	Frames from the animation Glass	72
B.5	Frames from the animation Glass rendered by radiosity	73

Chapter 1

Introduction

Animation is generating a series of frames of a scene, where each frame is an incremental alteration of the previous frame. Three-dimensional computer animation is the process of generating and displaying transformed images through time to give the illusion of motion. While the other areas in computer graphics deal with static images, computer animation conveys information in the form of motion or more generally, in the form of transformations through time.

Three dimensional computer animation involves three main phases [26]:

- Modeling; creating a three-dimensional computer model of the scene and the characters in it,
- Motion specification and synchronization; describing how a model will transform through time, and
- Rendering; producing realistic images by removing hidden surfaces and adding effects like shading, shadows, transparency and texture for each frame in the sequence obtained from the model and motion data.

Generally, modeling and the rendering phases have taken the most attention and proportionally, less research has been done on the motion itself. Both traditional and computer animation require great amount of human labour to generate an animated sequence. The main purpose in many of the computer animation systems developed so far is to decrease the amount of human labour used during the generation of the animation while still giving good enough control to the animator so that he can translate his artistic vision to the product.

An important problem in the production of an animation sequence is the great amount of information necessary to control and specify the motion. Many dependable systems need lots of specification even for a short animation sequence. If the scene gets more complicated and more realistic, the user has more trouble because of the great amount of information he has to provide for the system. A good approach to this kind of problem is the use of abstraction mechanism [10]. Different levels of abstractions can be used to decrease the amount of information to generate complicated animation sequences. In 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.

In fact, 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 complicated.

A very helpful idea for computer animation and motion abstraction is the object orientation. Many implementations have been done for computer animation using object oriented paradigm as in [6, 9, 10, 12, 14, 17, 33]. Some of those authors implemented scripting languages for the description of animation sequences [10, 12], while in [6, 9, 14] both interactive and scripting facilities have been supplied. Our approach is not to use any scripting language that is hard to use for naive users, but instead, provide an interactive environment for the

specification of motion. The object oriented idea facilitates both the implementation 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 object oriented paradigm to decrease the amount of information needed to specify a complicated animation sequence. The system is implemented in C++ [25] under Unix and XView.

The organization of the text is as follows. Chapter 2 presents a short overview of traditional animation and the evolution of computer animation. The methods of computer animation, an explanation of motion abstraction and related work done in object-oriented animation are given in respective sections in this chapter.

In Chapter 3, the low level details and controls on sequences are introduced. The classes designed for the problem, the creation of the geometric objects and their characteristics, the animation parameters and their interpolation schemes are explained.

In Chapter 4, the high level control mechanism and the abstractions on motion are presented. Different sequence types and their characteristics are explained in detail.

Chapter 5 gives a conclusion of the work. Appendix A presents some conversions and derivations about quaternions and Appendix B presents some colored images of the animations generated by the motion abstraction system implemented.

Chapter 2

Animation

2.1 An Overview of Traditional and Computer Animation

Early computer animation techniques were based on traditional animation. Two-dimensional animation techniques were used such as story-boarding, key-framing, in-betweening, painting and multi-plane backgrounds which are some steps of line and cel animation [3]. Line and cel animation is the traditional cartoon or character animation. Traditional animation production is a complex and elaborate process. It passes through a complicated sequence of production steps. A film usually starts with a script which is followed by its graphical form, the *storyboard*. It identifies key moments, specifies the sound, movement, composition and color aspects. Animators draw *key-frames* for each foreground character on paper, and specify general timing and movement on the exposure sheet. *Exposure sheet* is the essential book-keeping device of the process. It records all timing and sequencing information in tabular form. Assistant animators prepare the *in-between frames* based on the key-frames created by the animators. This step is the most time

and labour intensive part of the production process. Before the post-production, *line test* can be made which is the quick shooting of uncolored sketched frames, to verify the animation.

Computer animation has been developed for years to accomplish and speed up the process of the in-between frames generation. There are basically four methods for producing computer animation. These are key-frame animation, parametric key-frame interpolation, procedural or script based methods and dynamic simulation [31].

2.1.1 Key-Frame Animation

Based on the traditional animation, this method uses the shape and form of the scene at a number of fixed times, and generates the rest of the frames at other times by means of point-wise interpolation. This technique is the same as that is used in traditional character animation where the in-betweening is done by assistant animators. This can be done in a mechanical way by the assistants since it is a less artistically demanding task than the creation of the key frames. Besides the lack of smoothness and loss of depth and joint information in the in-between frames produced by this method, one more essential drawback is that every detail of the motion can only be defined to depend on one single global time parameter and there is no way to assign autonomous behavior to an object or define other dependencies between motions of the objects.

2.1.2 Parametric Key-Frame Interpolation

A scene can be parameterized in terms of a set of parameters, and the variations of these parameters generate the motion. Like key-frame animation, again some key-frames are given at key times, but this time in terms of some parameters that

are called key values. The intermediate values of the parameters are calculated to generate each frame. Typically the parameter values are interpolated by spline techniques to generate smooth animation. The advantages of this method over key-frame animation are, the compact representation of motion by the key values of the parameters, the implicit interactivity that comes during the specification of those key values, and the smoothness in the motion generated. The smoothness is achieved by the interpolation being based on spatial or physical parameters of the model [24]. However, this method also lacks the definition of dependencies between motions as key-frame animation.

2.1.3 Procedural or Script Based Methods

Script based methods offer a better way to specify kinematic behavior, enables to describe autonomous motion and if the script language is powerful enough then dependencies between motions can be introduced as well. However, script based systems are single-shot systems where the animator writes down a script in a language interpreted by the system and does not see the result and get any feedback until he compiles and runs the script. If the script needs modification, he duplicates the same steps once more. Though the script based methods are more expressive than the previous two methods, the advantage of direct control that comes by the interactive methods is lacking. A second disadvantage is the obstacle of the scripting language while nontechnically trained animators are using it. Using a scripting language generally requires the knowledge and experience of programming.

2.1.4 Dynamic Simulation

Sometimes motions rely on constraint based dynamics, like the motions of falling and tumbling articulated rigid bodies which cannot be generated exactly by script

based methods. Therefore there has been research on dynamic simulation, however this method is again a single-shot method and needs many trials for the adaptation of the initial conditions until the desired motion is obtained.

2.2 Motion Abstraction

Computer animation is an art and like any art form, it requires artistic ability and creativity to reach an acceptable final product. An animator using a computer animation system must be able to translate his artistic vision to the animation he is creating. For this purpose the system must be easy to use, interactive, flexible and fast. It can be thought that giving the animator the ability to control every single polygon or pixel in every single frame gives a great flexibility. But this does not provide any power to the animator other than pushing him into a bunch of cumbersome utilities during the production phase of the animation. However, main purpose of a computer animation system should be to give users high-level controls, but also provide them ability to command on the system in the lower levels. The great amount of specification information necessary to produce an animation sequence is an important problem in computer animation. A technique widely used in computer community to manage such problems is the use of abstraction [22].

Abstraction is a fundamental human capability that permits to deal with complexity. It is the selective examination of certain aspects of a problem. The goal of abstraction is to isolate those aspects that are important and useful for some purpose and suppress those that are unimportant and conceptually unnecessary [21].

Our system supports motion abstraction to decrease the effort the animator spends for the creation of a complex animation sequence which may consist of several different models and motion classes. The abstraction mechanism allows the user, both to command on many low level details of the animation and to

control it with increasingly higher levels of control mechanisms. The main idea in this kind of abstraction is that once an animation sequence is defined, its specification can be viewed as a unit sequence and it can be used as a building block for the definition of more complex animation sequences [7]. A sequence created in 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, hence helps the creation of the new sequence by letting it be defined at a higher level of abstraction. If thought recursively and repeatedly, this abstraction mechanism lets the creation of complex and detailed motions with dependencies among them, easily and without much detailed information except those in the lowest level.

As a result, with motion abstraction, the animator has high level controls but he can also command on the animation he is producing whenever he wants to, by controlling the lowest level sequences in detail. In this way, the system is easy to use, powerful, interactive and flexible. It gives the user the opportunity to re-use previously generated productions, in conjunction and relation with newer ones, in a well defined abstraction.

2.3 Object-Oriented Animation

Many implementations have been done for computer animation using object-oriented paradigm as in Clockworks [6, 14], SOLAR [10], Pinocchio [17], and in others like [9, 12, 33]. 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) [20] and Mira [27].

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.

Mira system is based on structured programming and data types. It consists of some data types like 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. A preprocessor called Miranim which is a command-driven and director-oriented interface was developed for creating CINEMIRA code. CINEMIRA is an extension in MIRA system.

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 [12]. In their work, an object is an encapsulation of activities and data, and the basic properties of an object are inherited from its prototype. With their own words, “the object-oriented approach favors viewing an application as a set of communicating capsules of activity, perhaps executing concurrently, rather than a large single piece of code”. They have provided an environment in which animated objects are fashioned into complex animations using a concise, directly executable specification language. They support the idea that computer animation is inherently object-oriented. Their language facilitates to specify a global temporal behavior and constrain local temporal behavior to meet the specification. It carries the idea from BGRAF2 [4] that time is a quantity that can be directly sampled and can be used to drive animation so that it is possible to separate the static representation of a graphical object from how it is animated. The motion specification is encapsulated also, so that the complex motion generated by the language is concise, re-usable and open-ended. The similarities of our system to this one is in motion abstraction, specification of complex animation with global temporal behavior based on local temporal behaved animations, and the discrimination of static representation of graphical objects from how they are animated so that an animation sequence can be assigned to any graphical object.

The language SOLAR (Structured Object-oriented Language for AnimatoRs) provides high level abstractions and adaptive motion through class inheritance

and message passing mechanisms of object-oriented paradigm [10]. They make five levels of abstractions which are structural, motion, functional, character and world modeling. These abstractions enable an animation sequence to be defined in steps as, starting from the simple object structure and motion descriptions to a more complex level of functional definition, object character building with adaptive skills, and world modeling of interacting objects. As apparent, structural abstraction supports the definition of the graphical objects, motional abstraction defines motion independent from the graphical objects, functional abstraction supports the grouping of a set of motions with a structural element, called skill, character abstraction associates skills with a class of object structures and supports adaptive motion, and world modeling abstraction defines the environment with all the objects in it.

Clockworks is an object-oriented computer animation system with integrated capabilities like modeling, image synthesis, animation, programming and simulation [6, 14]. It has been implemented in portable C under Unix with object-oriented features like objects with methods, instances, class hierarchies, inheritance, instantiation and message passing. Clockworks is made up of simple basic tools that can be combined. They provide both a scripting language and an interactive environment where the user can get direct feedback by sending messages to the objects. With their own words about object orientation in animation “We have found the object-oriented paradigm an extremely useful one for computer animation. It offers advantages both in software engineering and animation itself. . . . Overall the object-oriented paradigm is an advantageous, useful and natural concept for computer animation”. They say that object-oriented programming results in maintainable and extensible code and hierarchies of the animation system directly map into the hierarchies in geometric modeling. In Clockworks an actor-director model with autonomous objects communicating through messages is implemented by the scene and cue objects. Scene and cue objects constitute the basic building blocks of a script. A cue object represents a set of actions performed by a single object, over a specified span of time. Scene

object coordinates cues. Scene is a subclass of cue and as cue has a clock, it inherits this. At every clock tick, the scene sends a message to all of its cues to update their clocks and a message to the cameras to render the scene. Scenes and cues offer a low level time-based control. Because of writing scripts directly using scenes and cues is difficult, the system provides keyframe, keypoint and chronos objects that allow the user to interactively generate a script. Chronos object supports the time relations between a scene and its cues and actions. Keyframe provides the user to script an object interactively and generate cues automatically. Keypoints represent values for a particular object attribute pair over a period of time in a cue.

Pinocchio is an animation system that controls human motion with some sequencing facilities [17]. Movements performed by real actors are digitized by a 3D vision system called Elite. A general movement dictionary has been developed to classify movements and sequences of movements are described by an animation script where the characters themselves take care of their coordinations in the scene before display. 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. An object-oriented mechanism is associated to the movements retrieved from the motion database for synchronizing the characters in the scene. This mechanism provides the animator with high level control. The transitions between sequences of movements are controlled by this object-oriented environment. Every entity in this system is an object and all the objects operate independently and concurrently. 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. In our system this is done in a similar way where sequences have their own local times and the absolute times are generated according to their combinations controlled by a controller object.

Chmilar et al. have built a software architecture integrating the data structures for 3D modeling and animation so that time-based models that can change

shape during animation can be described [9]. They have implemented the system in C++. They designate the modular animation systems that isolate the modeling, animation and rendering phases as reductionist approaches, and signify that in these systems, as the geometry and structure of the models are given before motion control is specified, animation is limited to the basic transformations like scaling, rotation and translation, and it is difficult to animate changes in geometry. However in their approach which they call holistic approach, they use the powerful mechanism, object-oriented programming, that provides data and function encapsulation and multidirectional communication scheme between modules. In their approach, they integrate modeling and animation processes which makes an orthogonal system where anything: position, orientation, size, shape and structure, can be animated. They supply both an interactive environment and a scripting facility. Our system supports shape and geometric structure change during motion as well. The geometric structure change is accomplished by the motion abstractions on sequences where hierarchically dependent sequences can be constructed and, as this dependence is done on the motion level, it can be changed through time.

Zelevnik et al. have presented an interactive modeling and animation system that facilitates the integration of a variety of simulation and animation paradigms [33]. Their system extends modeling tools to include animation controls and provides the modeling of objects that change in shape, appearance and behavior over time. The system has an object-oriented architecture with objects like displayable objects, controllers, cameras, lights, renderers and user interfaces. Objects send and receive messages, and inquire information from each other. Through messages, objects can be transformed, deformed, colored, shaded, texture mapped, dynamically moved and volumetrically carved. Messages are functions of time and an object's list of messages describes the object's time varying structure and behavior and this list can be edited to alter the structure and behavior over time. Their system is a delegation system rather than a class-instance system, where in a class-instance system an object has two sorts of

associations which are association of an instance with its class, and association of a class with its super-class and these are static relations. However, a delegation system has only one relation which is between an object and its prototype. An object in a delegation system interprets a message by using one of its prototype's techniques. In this style an object called extension is created from another object called its prototype. Their system provides indications that next generation of graphics systems is headed towards an environment with time and behavior as first class notions rather than shape description and rendering as in the earlier systems.

2.4 Object-Oriented Design

Object-oriented modeling and design is a way of thinking, using the real world concepts and model of the problem [21]. In this style of programming the software is constructed as a collection of discrete objects that incorporate both data structure and behavior, which is in contrast with conventional programming style where data structure and behavior are loosely connected. Generally four basic characteristics are required in an object-oriented approach. These are: *identity*, *classification*, *polymorphism* and *inheritance*. *Identity* is quantization of data into discrete, distinguishable entities called objects. Each object has its own inherent identity and every object is distinct and unique. *Classification* is grouping objects that have the same data structure (attributes) and behavior (operations), into a class. Each class describes a possibly infinite set of individual objects and each object is an instance of its class. *Polymorphism* means that the same operation may behave differently on different classes. A specific implementation of an operation by a certain class is called a *method*. An object-oriented language automatically selects the correct method to implement an operation, based on the name of the operation and the class being operated on. Therefore, the user of the operator need not be aware of how many methods exist to implement a

polymorphic operation. Hence new classes can easily be added without changing the existing code, providing the methods for each applicable operation on the new class. Thus polymorphism facilitates the extendibility in designs. *Inheritance* is the sharing of attributes and operations among classes based on a hierarchical relationship. A subclass inherits all of the properties of its superclass and adds its own unique properties. The factorization out of common properties of some classes into a common superclass and inheritance of them reduces the repetition within designs. As a result, the characteristics in object oriented approach facilitates extensible, re-usable and well designed softwares.

Chapter 3

Motion Abstraction System: Low Level Controls

There are two major environments in our system. These are graphical object creation environment and sequence generation environment as seen in Figure 3.1. Graphical object creation environment has two subparts which are curve design and extended volume creation, and sequence generation environment also has two subparts which are key-framing and sequencing environments. There is a close relation between the key-framing and extended volume creation environments. During the key-framing phase the extended volume creation environment supports the creation of other transformable objects as well, like camera and light. Hence a transformable object created in this environment can be easily used in the key-framing environment as these two environments are closely related.

There is no need to switch from one environment to another because these four environments can all be active simultaneously. Hence, there is no restriction for separating modeling and animation steps. As all environments can be active at the same time, the animator can easily go back to the object creation environment while he is dealing with sequencing, so that no work is lost because

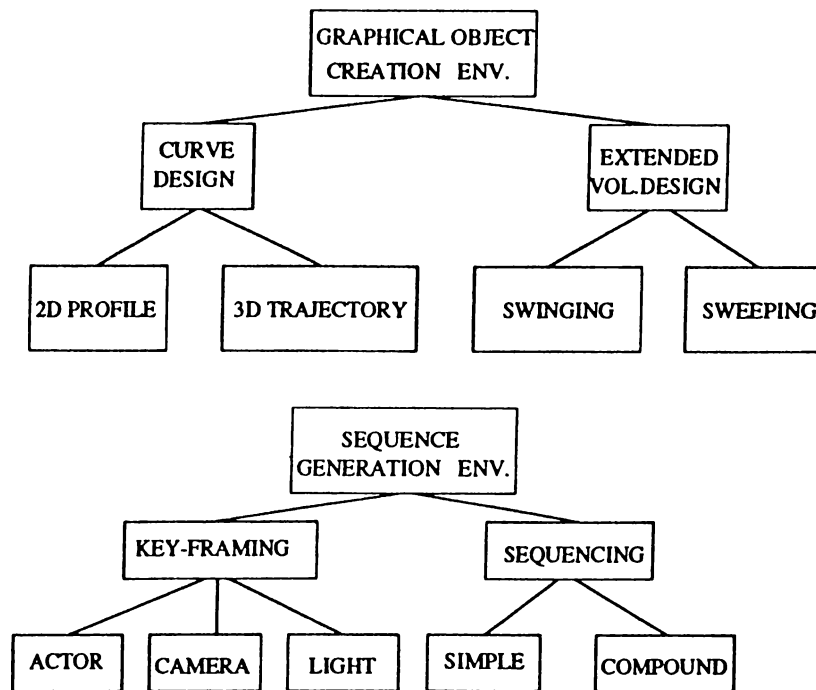


Figure 3.1: Structure of the system

of things forgotten during modeling or key-framing, etc. The interactive editing facility provided in the key-framing environment supports editing key values of the parameters easily. In this manner, the system provides a flexible, extensible and modular environment in which the user can work comfortably.

3.1 The 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 [21].

The lowest level classes used in the system as seen in Figure 3.2 are the basic

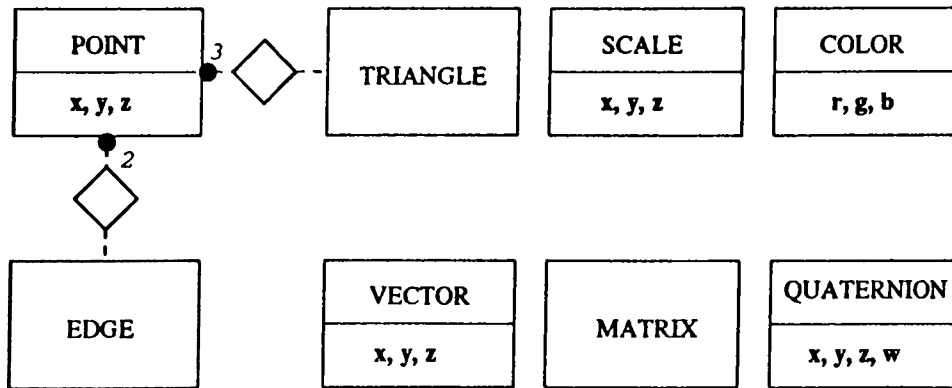


Figure 3.2: Lowest level classes

concepts used in any graphics subject like points, edges, vectors, etc. An *edge* is made up of two and a *triangle* is made up of three *point* objects. Conversion methods from a *quaternion* object to a *matrix* object and from a *point* object to a *vector* object and vice versa exist. All these classes have their own methods of their behavior on their own data. Although *matrix* class is implemented, it is not used for applying any transformations to any graphical object, instead *quaternion* is used for rotations, *scale* is used for scalings and *point* is used for translations. *Matrix* objects are used in some cases for representing and concatenating transformations but their *quaternion* and *point* conversions are used for applying those transformations.

Generic linked lists and arrays are implemented as classes which can keep any kind of object in them and generic mesh class is implemented which keeps points, doubles or integers. The advantage of genericity is that it enables parameterized modules so that it provides re-usability in code.

There is a *spline* class that has three subclasses namely, *nurbs*, *hermite* and *b-spline* as seen in Figure 3.3. This class provides the Hermite spline interpolation on the key values of the animation parameters and NURBS (Non-Uniform Rational B-Splines) approximation on the curves and meshes that the graphical objects are made of.

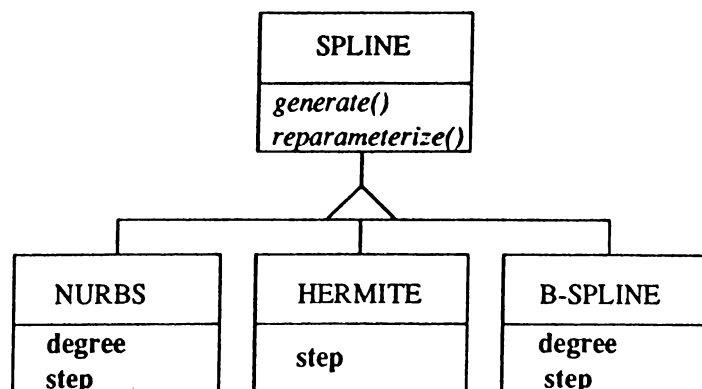


Figure 3.3: Spline class

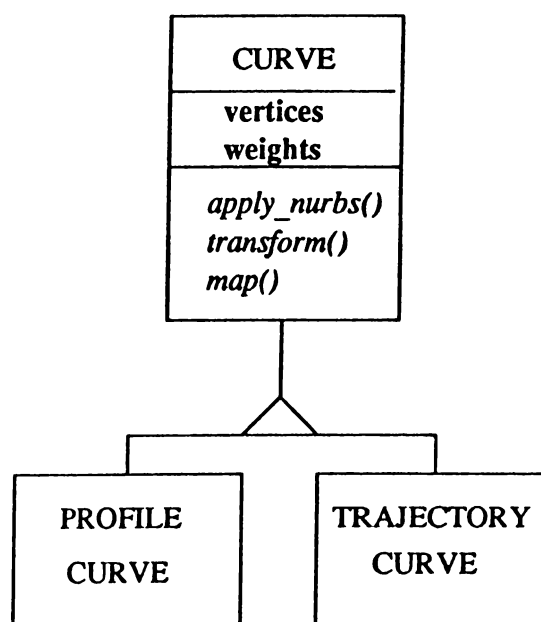


Figure 3.4: Curve class

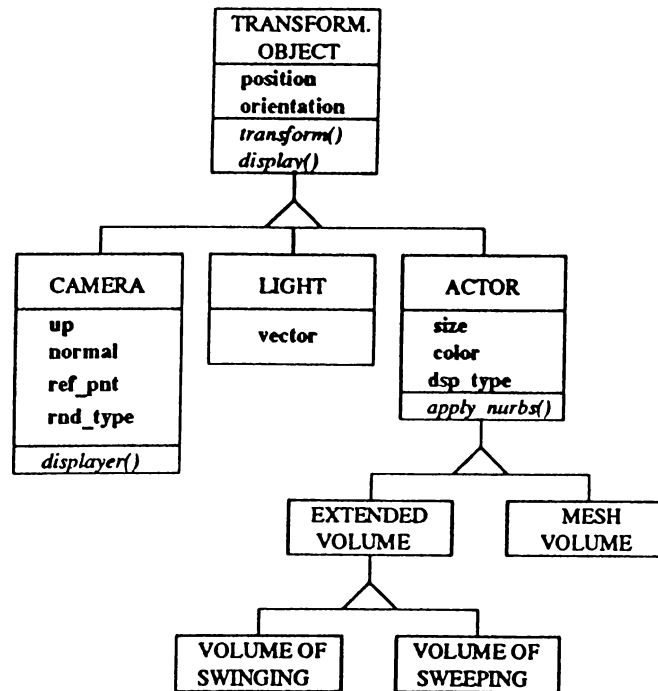


Figure 3.5: Transformable object class and its subclasses

There are two subclasses of *curve* class, Figure 3.4, which are *profile* and *trajectory*. *Curve* objects are used for the creation of the extended volumes which are volumes of swinging and volumes of sweeping. Swinging is a generalization of revolution. One profile and one trajectory curve is needed for an extended volume. *Curve* class has attributes like, **vertices** which are arrays of points that make up the curve and **weights** which exist for each point in the vertices. These weights are used when NURBS approximation is applied onto the curve. The most important methods in the *curve* class are `apply_nurbs()`, `transform()`, and `map()`. The last one is for comparing two curves and generating a map for shape interpolation on curves. This shape interpolation is a base for the shape interpolation on the extended volumes explained in Section 3.3.3.

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 3.5.

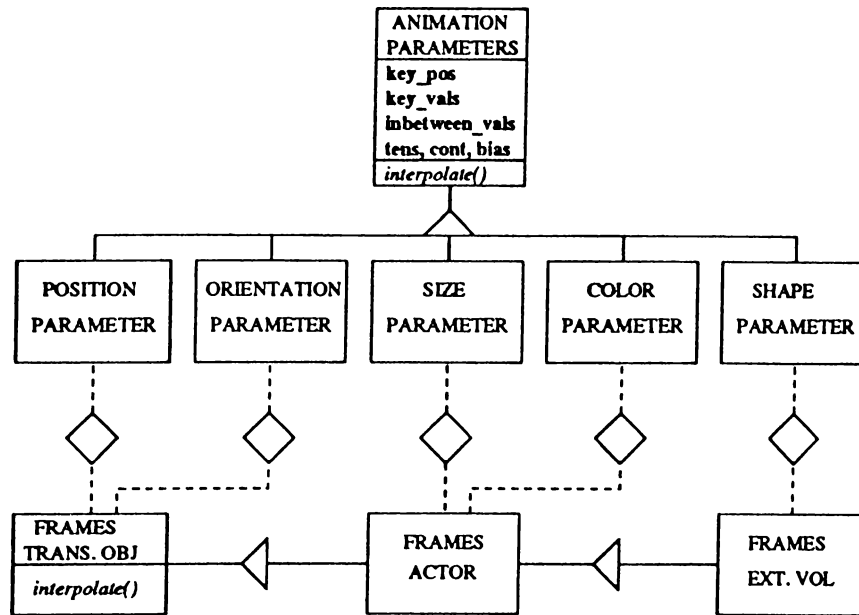


Figure 3.6: Frames and animation parameters classes

Some important methods in this class are *transform()* and *display()*, and like almost all of the other methods of this class, these methods are virtual. Virtual methods support overloading by run-time resolution of methods. This class has the attributes **position** and **orientation** because size has no meaning for its subclasses *camera* and *light*. The subclass *camera* has attributes like **up** and **normal** which are vectors. Obviously, **normal** shows the viewing direction and **up** shows the inclination of the camera. The other attributes are **ref_pnt**, specifying the position and **rnd_type**, specifying the rendering type that the camera will use. The other important subclass of *transformable object*, *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 sweeping*. *Actor* class has the attributes, **size**, **color** and **dsp_type**. The last one denotes the display type of the actor, which can be smoothed (NURBS applied), unsmoothed (as its control points), box (as its bounding box) or triangulated (for exporting information of a scene to external renderers like radiosity).

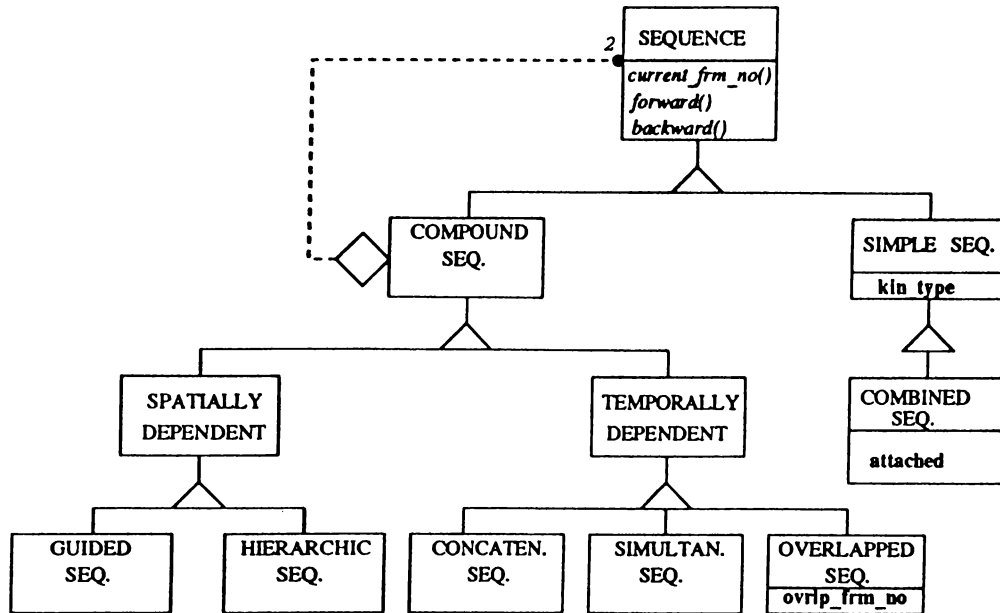


Figure 3.7: Sequence class and its subclasses

There are three levels of hierarchy in *frames* class as seen in Figure 3.6. The upper most class 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 *color*, and finally the lowest subclass is for extended volumes, so as the extended volumes can change shape, it has an extra *shape parameter*.

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 3.7, the *sequence* class has two major subclasses, *simple sequence* and *compound sequence*. *Simple sequence* has an attribute *kin.type* that denotes the kinetic information type of the sequence. *Simple sequence* has a subclass *combined sequence* which has an attribute, *attached*, denoting that the sequences combined will be attached into each other providing a smooth transition. *Compound sequence* class has more varieties. It has two subclasses, temporally and spatially dependent sequences. A compound sequence is made of two sequences that is denoted by

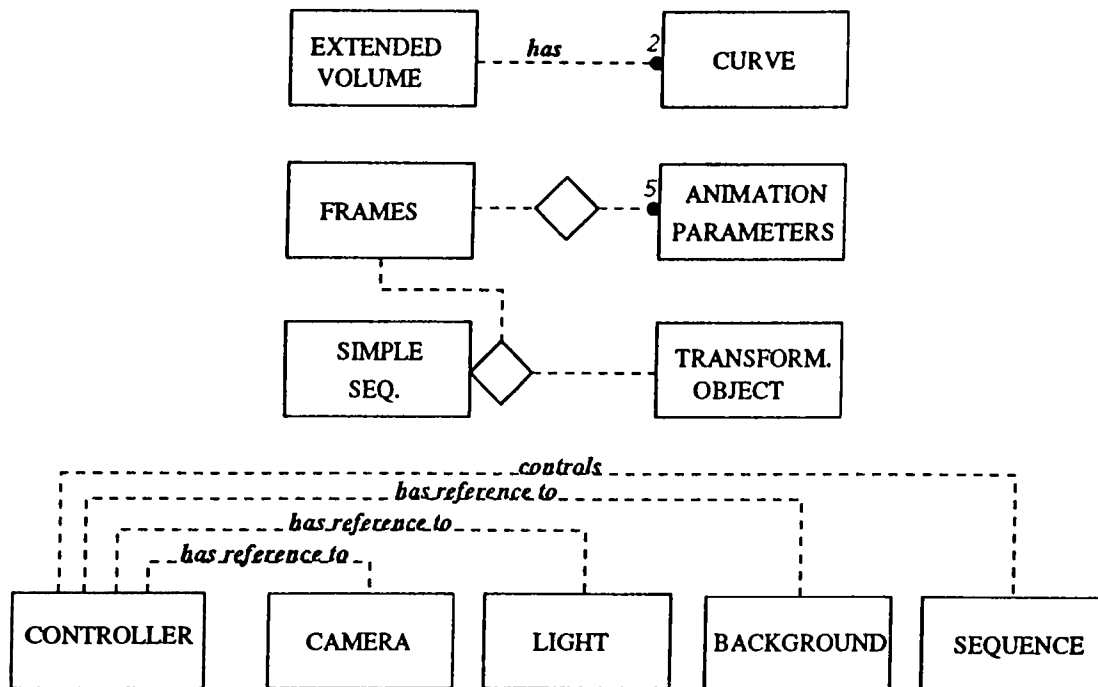


Figure 3.8: Relations among some of the classes

the recursive aggregation in the Figure 3.7. There are three subclasses of *temporal dependent sequence*, *concatenated*, *simultaneous* and *overlapped* sequences and two subclasses of *spatial dependent sequence*, *guided* and *hierarchic* sequences. Sequences will be explained in detail in Chapter 4.

In Figure 3.8, some important associations and aggregations between some classes are presented. Aggregation is a special form of association, it is not an independent concept. If two objects are tightly bound by a part-whole relationship then it is an aggregation but if they are considered independent even though they may be often linked then it is an association [21]. An *extended volume* has two *curves*, a profile and a trajectory, *animation parameters* are parts of a *frames* object, a frames object and a transformable object are parts of a simple sequence and finally, *controller* controls the sequences and has references to the current camera, light and background objects through the sequence.

3.2 Transformable Objects

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 has no geometry. The light source which has a geometry just as an ordinary actor, is for the purpose of rendering by radiosity methods, while the other one is for the other shading techniques like 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 [8]. An animation sequence rendered in this way can be seen in Figure B.5.

3.2.1 Actors

An actor can be defined either as a network of 3D data, a mesh volume, available by some other means, such as a data file containing the 3D network data of an object, or it can be defined as a 3D volume extended from 2D, such as a volume of swinging or volume of sweeping. If preferred the 3D geometry of an actor can be smoothed by fitting a NURBS on it.

There are a number of general advantages of NURBS [18] among which the following are especially useful in our application:

- manipulation of the control points and the weights provide the ease and flexibility to design different kinds of shapes with local control,
- the evaluation of NURBS is reasonably fast and computationally stable, and

- they are invariant under affine¹ transformations, and perspective and parallel projections.

These are some useful properties of NURBS that become advantageous in animation.

The definition of a NURBS curve is as follows:

$$C(u) = \frac{\sum_{i=0}^n w_i P_i N_{i,p}(u)}{\sum_{i=0}^n w_i N_{i,p}(u)} \quad (3.1)$$

where w_i are the weights, P_i are the control points and $N_{i,p}(u)$ are the normalized B-spline basis functions of degree p , the definition of which can be found in [18].

A NURBS surface can be defined as:

$$S(u, v) = \frac{\sum_{i=0}^n \sum_{j=0}^m w_{i,j} P_{i,j} N_{i,p}(u) N_{j,q}(v)}{\sum_{i=0}^n \sum_{j=0}^m w_{i,j} N_{i,p}(u) N_{j,q}(v)} \quad (3.2)$$

where $w_{i,j}$ are the weights, $P_{i,j}$ form a control net, and $N_{i,p}(u)$ and $N_{j,q}(v)$ are the normalized B-spline basis functions of degree p and q in the u and v directions respectively.

The definition of the NURBS curve can be rewritten in a simplified form as follows :

$$C(u) = \sum_{i=0}^n P_i R_{i,p}(u) \quad (3.3)$$

$$R_{i,p}(u) = \frac{w_i N_{i,p}(u)}{\sum_{j=0}^n w_j N_{j,p}(u)} \quad (3.4)$$

where $R_{i,p}(u)$ are rational basis functions and their analytic properties determine the geometric behavior of curves. These geometric behaviors are as follows:

- some special cases of NURBS curve are Bezier and B-spline curves,

¹Affine transformations are the linear transformations such as scaling, rotation and shearing followed by a translation.

- a NURBS curve can be locally² approximated,
- it lies within the convex hull of the control points,
- it is invariant under affine and perspective transformations,
- it has the same differentiability properties as with the basis functions,
- if a weight is set to zero, then that control point has no effect on the curve,
- if a weight diverges to infinity, then the curve, at the related parameter values, converges to that point.

Extended Volume Generation

An actor may be an extended volume obtained from surfaces of swinging or surfaces of sweeping [1]. For obtaining a surface of sweeping, a profile curve must be swept through a trajectory curve, and for obtaining a surface of revolution, a profile curve must be rotated around an axis of the plane it lies on. Revolution can be generalized under the word swinging. If there is a profile curve $P(u)$ on the xy plane and a trajectory curve $T(v)$ on the xz plane defined as NURBS curves as follows:

$$P(u) = \sum_{i=0}^n P_i R_{i,p}(u) \quad (3.5)$$

$$T(v) = \sum_{j=0}^m T_j R_{j,q}(v) \quad (3.6)$$

swinging the profile curve through the trajectory curve gives the following surface as seen in Figure 3.9:

$$S(u, v) = (P_x(u)T_x(v), P_y(u), P_x(u)T_z(v)) \quad (3.7)$$

²Local approximation is that, if a control point or the weight of a control point is modified, this modification affects the shape of the curve only in $p + 1$ knot spans where p is the degree of the curve.

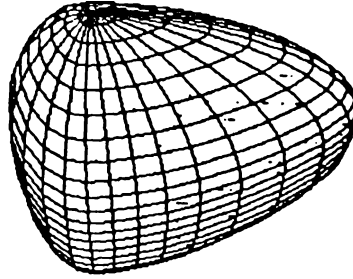


Figure 3.9: A volume of swinging

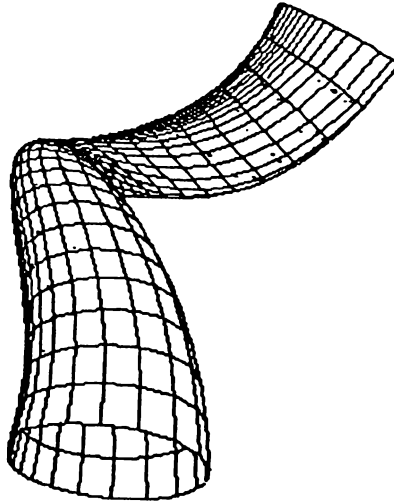


Figure 3.10: A volume of sweeping

where the control points for this surface are

$$Q_{i,j} = (P_{ix}T_{jx}, P_{iy}, P_{ix}T_{jz}) \quad (3.8)$$

and the weights are

$$w_{i,j} = w_i w_j \quad (3.9)$$

Sweeping the profile curve through the trajectory curve gives the following surface as seen in Figure 3.10:

$$S(u, v) = (P'_x(u) + T_x(v), P'_y(u) + T_y(v), P'_z(u) + T_z(v)) \quad (3.10)$$

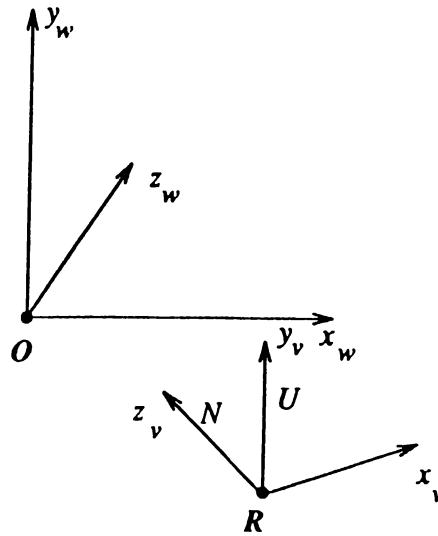


Figure 3.11: World and viewing coordinate systems.

where the control points for this surface are

$$Q_{i,j} = (P'_{ix} + T_{jx}, P'_{iy} + T_{jy}, P'_{iz} + T_{jz}) \quad (3.11)$$

and the weights are obtained as in the sweeping case. Here, the P'_i is the rotated version of the control point P_i according to the inclinations of the trajectory curve. So both for swinging and sweeping cases, we can use a profile curve and a trajectory curve, hence swinging volumes are generated giving more flexibility than revolutions volumes.

The network of control points for extended volumes are obtained in this way and NURBS is applied over it by equation 3.2.

3.2.2 Cameras

Both the world and the viewing coordinate systems are defined as left handed coordinate systems. Figure 3.11 shows the world and viewing coordinate systems according to each other. x_w, y_w, z_w are the unit vectors on the positive x, y, z

axes of the world coordinate system. O is the origin, $((0,0,0)$ point) of the world. x_v, y_v, z_v are the unit vectors on the positive x, y, z axes of the viewing coordinate system. z_v is the N (normal) and y_v is the U (up) vector of the camera. R is the reference point of the camera in world coordinates.

In order to obtain what is seen from the camera, everything defined in the world coordinate system must be redefined in the viewing coordinate system. For this purpose, the origin (O) of the world system is translated to R , then the world system is rotated to fit on the viewing system exactly. For the rotation transformation in this process, the following method as explained in [28] has been used. In this method, the transformation matrix from one coordinate system to another that has the same origin is obtained in the following way.

$$T_w^v = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix}$$

where T_w^v is the transformation matrix from world coordinate system to viewing coordinate system. The t_{ij} s are as follows:

$$\begin{aligned} t_{11} &= \vec{x}_w \vec{x}_v & t_{12} &= \vec{y}_w \vec{x}_v & t_{13} &= \vec{z}_w \vec{x}_v \\ t_{21} &= \vec{x}_w \vec{y}_v & t_{22} &= \vec{y}_w \vec{y}_v & t_{23} &= \vec{z}_w \vec{y}_v \\ t_{31} &= \vec{x}_w \vec{z}_v & t_{32} &= \vec{y}_w \vec{z}_v & t_{33} &= \vec{z}_w \vec{z}_v \end{aligned}$$

where $\vec{x}_i \vec{x}_j$ is the dot product of the two unit vectors and gives the cosine of the angle between them. A vector \vec{p} in the world coordinate system can be redefined in the viewing coordinate system as

$$\vec{p}_v = T_w^v \vec{p}_w \quad (3.12)$$

where p_v and p_w are the representations of the vector \vec{p} in the viewing and world coordinate systems respectively. Hence, T_w^v is the transformation matrix that maps a vector represented in the world coordinate system to a vector represented in the viewing coordinate system. After this matrix is calculated once for a particular orientation of a camera, it is converted to a unit quaternion that represents

this rotation, and this quaternion is used for the rotation purposes. Quaternions will be explained later in detail in Section 3.3.2, and the conversions from a rotation matrix to a unit quaternion and vice versa are shown in Appendix A.

3.3 Animation Parameters and Their In-betweening

An animation sequence, must have a frame data where the key values for the animation parameters and their key frame numbers (times) are defined.

There are some differences on the parameters applied to the transformable objects. The parameters applicable to a camera are position and orientation parameters, and to an actor are size and color parameters as well as position and orientation ones. Position, orientation and size parameters are defined as triples x, y, z . Color parameters are reflectivity and emission values defined as triples of R, G, B , where these parameters are used in rendering by radiosity.

For actors, if they are created as volumes of swinging or sweeping, then morphing (shape change) is also possible and the shape of the actor is also a parameter of the animation.

The key values of these parameters are interpolated to generate the in-between values and resultantly, the in-between frames.

3.3.1 Interpolation Scheme

A very common technique used in computer animation is the key frame animation where the in-between frames are generated automatically based on a series of key frames supplied by the animator. If linear interpolation is used during

the generation of the in-between frames then undesirable side effects occur giving the animation a mechanical look. These problems are lack of smoothness, discontinuities in speed of motion, and distortion during rotations.

The most important advantage of an interpolation technique compared to an approximation technique is that in an interpolation scheme, the interpolated values for the parameters pass exactly through the key values, while an approximation scheme only approximates the key values but the approximated values do not necessarily pass through them. This property of the interpolation techniques is mostly preferred in animation because it guarantees that the the key-frames given by the animator are respected.

The best approach is to use a non-linear interpolation technique for the generation of the in-between values, both to avoid the problems of linear interpolation and the inadequacy of the approximation techniques.

Polynomial interpolation is the most fundamental of all interpolation concepts. It is mostly of theoretical value because nowadays, faster and more accurate methods have been developed. These methods are *piecewise polynomial* methods which rely on the polynomial methods. Polynomial interpolation is not restricted to interpolation on point data; derivative data can also be interpolated. This leads to an interpolation scheme called *Hermite interpolation*. The piecewise polynomial scheme constructs curves that consist of polynomial pieces of the same degree and that have an overall smoothness.

C^1 *piecewise cubic Hermite interpolation* is one of these schemes [11]. It is a simple interpolation technique although not the most practical one, but has some advantages for the purposes of animation. It solves the following problem: Given some data points x_0, \dots, x_L , their corresponding parameter values u_0, \dots, u_L and tangent vectors m_0, \dots, m_L , find a C^1 piecewise cubic polynomial S that interpolates to these data points, i.e,

$$S(u_i) = x_i \quad , \quad \frac{d}{du}S(u_i) = m_i \quad ; \quad i = 0, \dots, L. \quad (3.13)$$

Over interval $t \in [u_i, u_{i+1}]$ the representation of the interpolant S in terms of the *cubic Hermite polynomials* $h_i^3(t)$ is

$$S(t) = x_i h_0^3(t) + m_i h_1^3(t) + m_{i+1} h_2^3(t) + x_{i+1} h_3^3(t) \quad 0 \leq t \leq 1 \quad (3.14)$$

where t is the local parameter of the interval $[u_i, u_{i+1}]$ [2].

The Hermite basis polynomials are defined as

$$\begin{aligned} h_0^3(t) &= 2t^3 - 3t^2 + 1 \\ h_1^3(t) &= t^3 - 2t^2 + t \\ h_2^3(t) &= t^3 - t^2 \\ h_3^3(t) &= -2t^3 + 3t^2 \end{aligned} \quad (3.15)$$

As the low level motion in our system is supplied by parametric key-frame interpolation method, the key values for the position, size, color and shape parameters given at the key frames are interpolated through this interpolation which provides the respection of the key values. The orientation parameter is interpolated through a special interpolation scheme for quaternions explained in the next section.

Each of the component polynomials of the animation parameters are interpolated by *piecewise cubic Hermite interpolation*. Two constraints are given by the interpolation conditions and other two constraints are given by specifying tangent vectors at $t = 0$ and $t = 1$. Thus $Q_i(t)$ is completely determined by the data points, x_i, x_{i+1} , and the tangent vectors at them, m_i, m_{i+1} .

The tangent vectors at the key positions can be calculated purely from local geometry information as

$$\begin{aligned} m_i &= \frac{1}{2}[x_{i+1} - x_{i-1}] \\ &= \frac{1}{2}[(x_{i+1} - x_i) + (x_i - x_{i-1})] \end{aligned} \quad (3.16)$$

which is simply the average of the source chord $x_i - x_{i-1}$ and the destination chord $x_{i+1} - x_i$.

A smooth animation through a given set of key values does not always produce the desired effect that the animator expects. Sometimes a wider, more exaggerated motion, or a motion that overshoots a key value may be desired [16]. For these purposes, adjustments on tension, bias and continuity are very helpful and generate good and challenging results for the animation. Especially, bias control easily simulates the traditional animation effects “following through after an action” by *overshooting* the key value or “exaggerating a movement” by *under-shooting* a key value. These adjustments are introduced to the piecewise cubic Hermite interpolation process by means of some parameters that are used in the calculation of the resultant values of the tangent vectors that are put into the calculation of the interpolant as values m_i, m_{i+1} , at the key positions [2]. These parameters bring the advantage of this interpolation scheme for animation purposes. They are introduced to the interpolation process by separating each tangent vector into two parts, an incoming part, from source chord and an outgoing part, to destination chord. The single value for tangent vector m_i in the default interpolant at each x_i is replaced by these two parts.

The complete values of tangent vectors of source chord ms_i and destination chord md_i are as follows with the use of the continuity(c), bias(b) and tension(t) parameters.

$$ms_i = \frac{(1-t_i)(1-c_i)(1+b_i)}{2}(x_i - x_{i-1}) + \frac{(1-t_i)(1+c_i)(1-b_i)}{2}(x_{i+1} - x_i) \quad (3.17)$$

$$md_i = \frac{(1-t_i)(1+c_i)(1+b_i)}{2}(x_i - x_{i-1}) + \frac{(1-t_i)(1-c_i)(1-b_i)}{2}(x_{i+1} - x_i) \quad (3.18)$$

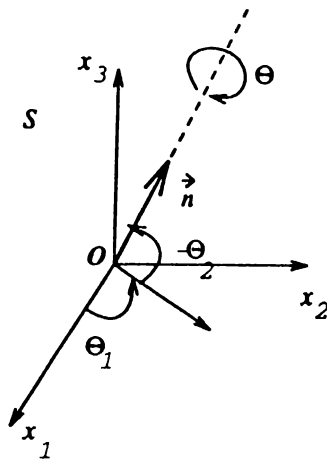
Here are some effects of the parameters on the interpolant:

- $t_i \rightarrow 1$, reduces the length of the tangent vector to zero and tightens the curve to a corner,
- $t_i \rightarrow -1$, increases the tangent vector to twice its default length and produces more slack in the curve,
- $t_i > 1$, produces loops.

- $c_i = 0$, tangent continuity,
- increasing $|c_i|$ results in two tangent vectors becoming increasingly distinct,
- $c_i = -1$, ms_i reduces to the source chord, md_i reduces to the destination chord, producing a pronounced corner in the curve,
- $c_i \rightarrow -\infty$, corner more acute and curve buckles inward,
- $c_i \rightarrow \infty$, corner pointing opposite direction.

- $b_i = 0$, two chords are weighted equally,
- $b_i = -1$, tangent vector is determined by the destination chord,
- $b_i = 1$, tangent vector is determined by the source chord,
- $b_i \rightarrow -\infty$, more the trajectory bends to one side of P_i ,
- $b_i \rightarrow \infty$, more the trajectory bends to the other side of P_i ,

The composite formulas, ms_i and md_i provide a considerable flexibility in the construction of the resultant interpolant.

Figure 3.12: The specification of vector \vec{n}

3.3.2 Representation and Interpolation of the Orientations by Quaternions

Quaternions were discovered by Sir William R. Hamilton in October, 1843, while he was trying to extend the complex plane to three dimensional space. He was trying to find a way to multiply triples so that the norm was preserved for 14 years, and finally realized that quadruples would work. By an odd quirk of mathematics, only 2, 4 or 8 components are norm preserving as Sir Hamilton desired [23].

The common solution to the problem of representing general rotations and in-betweening them is using the Euler's angles interpolated independently. This is not an ideal solution because the Euler angles given for rotations around different axes must be used in the given order as rotations do not commute. Instead, quaternions with interpolation on the unit sphere is better for representing and in-betweening rotations [19, 23]. Quaternions are compact and require less number of calculations for concatenating and applying rotations than the matrix representation. An homogeneous transformation matrix in three dimensions representing a rotation is 4×4 , hence requires 16 floating point numbers, while

a quaternion requires only 4 of them. Quaternions are not widely known, but conversion from Euler angles to quaternions and vice versa are well defined [23].

Euler's Theorem: If two coordinate systems S and Σ have the same origin O , then the system S can be fitted on the system Σ by a rotation α around a line δ which is the axis of rotation passing from the origin [32]. The line δ , explicitly the vector that determines the line δ and lies over it, can be determined by two angles (θ_1 and θ_2) in the system S as seen in Figure 3.12. Therefore, the orientation of the system Σ according to S can be determined by three angles, θ_1 , θ_2 and the amount of rotation α . These three independent angles that provide the transformation from S to Σ are called *Euler's angles*. The matrix that transforms S to Σ can be obtained by the concatenation of the rotations by these angles. In literature, different groups of Euler's angles are used that provide this transformation. In [28], it can be found that the transformation matrix T_{Σ}^S that transforms Σ to S , is as follows for the group of 313 Euler's angles with rotations of ϕ around x_3 , θ around x_1 and ψ around x_3 again.

$$\begin{bmatrix} \cos \psi \cos \phi - \sin \psi \cos \theta \sin \phi & -\sin \psi \cos \phi - \cos \psi \cos \theta \sin \phi & \sin \theta \sin \phi \\ \cos \psi \sin \phi + \sin \psi \cos \theta \cos \phi & -\sin \psi \cos \phi + \cos \psi \cos \theta \cos \phi & -\sin \theta \cos \phi \\ \sin \psi \sin \theta & \cos \psi \sin \theta & \cos \theta \end{bmatrix} \quad (3.19)$$

Again in [28], it is found that the transformation matrix can also be shown as

$$T(\vec{n}, \alpha) = \cos \alpha I + (1 - \cos \alpha) \vec{n} \vec{n}^T + \sin \alpha N \quad (3.20)$$

where \vec{n} is the unit vector that determines the axis of rotation, α is the angle of rotation as explained in Euler's theorem, I is the identity matrix, and N is the skew symmetric matrix generated from vector \vec{n} .

Therefore, these two expressions can be solved to obtain the axis and angle of rotation in terms of Euler's angles. Instead of using the Euler's angles in these expressions, which are very complicated, the Euler's parameters can be used, which are much more compact [32]. Now, the axis and angle of rotation can be

defined in terms of four parameters called Euler's parameters as follows.

$$\vec{v} = \begin{bmatrix} \xi \\ \eta \\ \zeta \end{bmatrix} = \sin(\alpha/2) \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix}, \quad \chi = \cos(\alpha/2) \quad (3.21)$$

As the vector of rotation \vec{n} is unit, then

$$\vec{v}^T \vec{v} + \chi^2 = \xi^2 + \eta^2 + \zeta^2 + \chi^2 = 1 \quad (3.22)$$

When 3.20 is rewritten in terms of Euler's parameters, it is obtained that

$$T(\vec{n}, \alpha) = (2\chi^2 - \vec{v}^T \vec{v})I + 2\vec{v}\vec{v}^T + 2\chi V \quad (3.23)$$

When this matrix is written explicitly and compared with the matrix in 3.19 the Euler's parameters can be found in terms of Euler's angles. These are

$$\xi = \pm \sin \frac{\theta}{2} \cos \frac{\theta - \psi}{2} \quad (3.24)$$

$$\eta = \pm \sin \frac{\theta}{2} \sin \frac{\theta - \psi}{2} \quad (3.25)$$

$$\zeta = \pm \cos \frac{\theta}{2} \sin \frac{\theta + \psi}{2} \quad (3.26)$$

$$\chi = \pm \cos \frac{\theta}{2} \cos \frac{\theta + \psi}{2} \quad (3.27)$$

and the signs must be taken the same for all the Euler's parameters.

When the Euler's parameters are used in a quadruple, it is seen that this quadruple is a quaternion and it is unit because of the relation in equation 3.22. Hence, a unit quaternion represents a rotation.

$$q = \begin{bmatrix} \xi \\ \eta \\ \zeta \\ \chi \end{bmatrix} \quad (3.28)$$

The key values for orientation which are quaternions, are interpolated by *slerp* (spherical linear interpolation) which is an interpolation scheme for quaternions [23]. It generates in-between quaternions of two unit key quaternions by the method of great arc in-betweening on the unit sphere where unit quaternions lie.

As a brief summary of quaternions; a quaternion has four components, three of them give the axis of rotation and the fourth is obtained by the angle rotated about that axis. So, $q = [s, (x, y, z)]^T$ is a quaternion, where the vector $\vec{v} = (x, y, z)^T$ is the axis of rotation and $s = \cos \frac{\theta}{2}$ where θ is the angle of rotation. The multiplication of two quaternions is as follows:

$$[s_1, \vec{v}_1][s_2, \vec{v}_2] = [(s_1s_2 - \vec{v}_1\vec{v}_2), (s_1\vec{v}_2 + s_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2)] \quad (3.29)$$

A vector, \vec{v} , can be rotated by a quaternion, q , as follows, which is shown in detail in Appendix A.

$$\hat{\vec{v}} = Rot(\vec{v}) = q[0, \vec{v}]q^{-1} \quad (3.30)$$

assuming that the vector can be represented as a quaternion of $[0, \vec{v}]$ of which the rotational part is eliminated. The inverse of a quaternion is obtained as follows

$$q = [s, \vec{v}]^T$$

$$q^{-1} = \frac{1}{\|q\|^2} [s, -\vec{v}]^T \quad (3.31)$$

$$\|q\|^2 = s^2 + \vec{v} \cdot \vec{v} \quad (3.32)$$

The quaternion that transforms q_1 into q_2 by analogy to the vector translation is

$$\Delta q = q_1^{-1}q_2 \quad (3.33)$$

and spherical linear interpolation in the unit quaternion space is

$$q = slerp(q_1, q_2, u) \quad 0 \leq u \leq 1$$

$$= q_1(q_1^{-1}q_2)^u \quad (3.34)$$

$$= \frac{\sin(1-u)\theta}{\sin\theta} q_1 + \frac{\sin u\theta}{\sin\theta} q_2 \quad (3.35)$$

where $q_1 q_2 = \cos \theta$ and the interpolated quaternion goes along a great arc on a $4D$ unit sphere which is the shortest way between q_1 and q_2 . The equation 3.35 is simpler for practical purposes and its derivation is shown in Appendix A. Other related operations on quaternions and conversions between quaternions and Euler angles can be found in [23].

3.3.3 Interpolation of Shape

As mentioned previously, the actors that are created as volumes of swinging or sweeping can have changes in their shapes. These are extended volumes based on two curves, a profile and a trajectory. For generating a volume of swinging, the profile curve is swung over the trajectory curve, and for volume of sweeping, the profile curve is swept through the trajectory curve, hence extended volumes are obtained as explained in Section 3.2.1 and seen in Figures 3.9 and 3.10. A vertical cut on this volume has the shape of its profile curve and a horizontal cut has the shape of its trajectory curve. As extended volumes are based on their profile and trajectory curves, a shape change applied to these curves will generate a similar shape change on the volume itself.

Our approach to shape change is based on the two-dimensional profile and three-dimensional trajectory curves that make up the three-dimensional extended volumes. A curve is kept as its control points and when an extended volume is obtained from its profile and trajectory curves, it is kept as its control points as well. As mentioned in Section 3.1, a transformable object has an attribute for specifying its display type, smoothed or unsmoothed. If it is smoothed, then NURBS will be applied on the control points of the volume and curve, based on the weights of the control points, so a smooth curve and volume will be obtained. The weights of the volume are obtained from the profile and trajectory curves as mentioned in Section 3.2.1. If the display type is unsmoothed then the volume will be displayed as its control points. The shape change of a volume has been

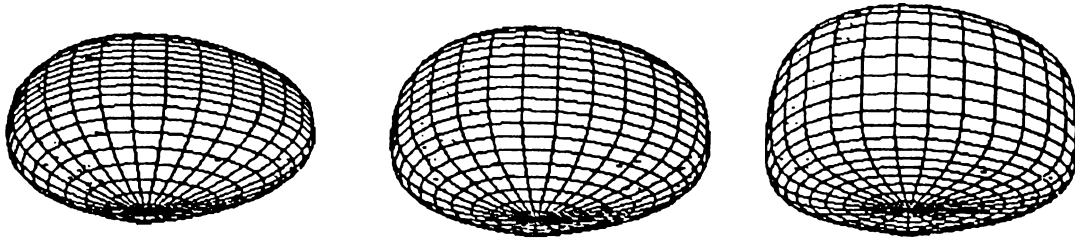


Figure 3.13: Shape interpolation by changing the weights of the control points

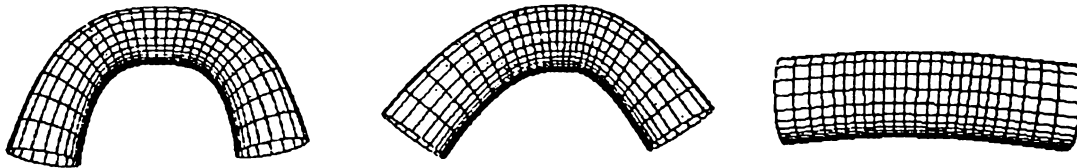


Figure 3.14: Shape interpolation by changing the control points

accomplished in two ways.

In the first approach, by the help of the weights and NURBS, it is possible to change the shape of a volume without changing the control points of its profile and trajectory curves. In this approach, the shape change is performed by changing the weights of the control points of its profile and/or trajectory curves, thus when NURBS is applied to the old control points with new weights, a new shape will be obtained as seen in Figure 3.13.

In the second approach to shape change, not only the weights of the control points but also the control points themselves can be changed. The control points will have geometrically different positions, so that the shape of the volume will change obviously as seen in Figure 3.14. In this approach, a problem of mapping two curves to each other occurs where one curve is for the current shape of the volume and the other one is for the next shape. The profile curve of the current shape of the volume must be mapped to the profile curve of the next shape and the trajectory curves must also be mapped accordingly.

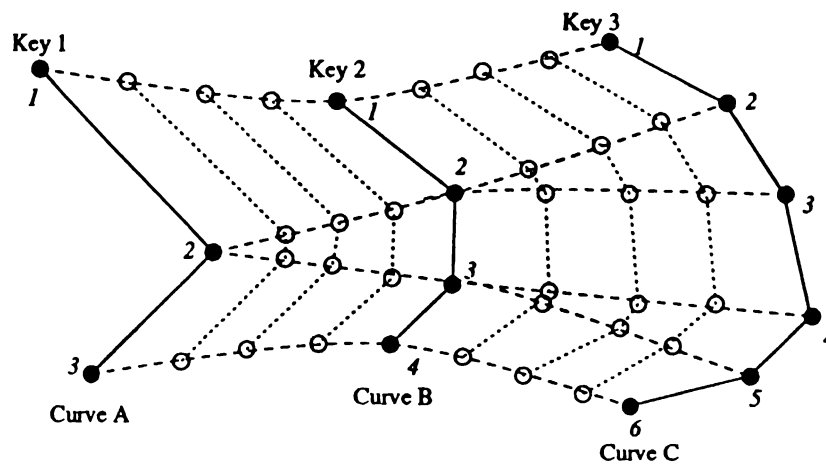


Figure 3.15: Mapping of three curves and their in-betweening

For obtaining a smooth shape change, some key shape values are given for the shape parameter in addition to the other animation parameters. A key shape value is an extended volume that is to be interpolated. Hence, some extended volumes are given as key shape values at key frames and their profile and trajectory curves are mapped. The new key curves generated by this mapping are interpolated by the same interpolation mechanism explained in Section 3.3.1, as the other animation parameters do, to obtain a smooth shape change, both for their control points and the weights of their control points.

In Figure 3.15 curves *A*, *B* and *C* are the profile curves of the key shape values given at successive key frames. In this figure the large dotted curves show the mapping and the small dotted curves are the smoothed in-between shapes of the curves. The key shape values of each successive key frame must be mapped in such a way that the curve will be transformed into the next curve as smoothly as possible. The smoothness must be achieved in two manners. Firstly, when one curve is put over the other curve, the control points that stand nearest and have similar convexity or concavity must be mapped to the each other. The algorithm for this mapping can be seen in Figure 3.17. Secondly, after this mapping is done, the control points that are mapped to each other must be transformed from one to the other, passing through the key values smoothly. For example, Figure 3.15

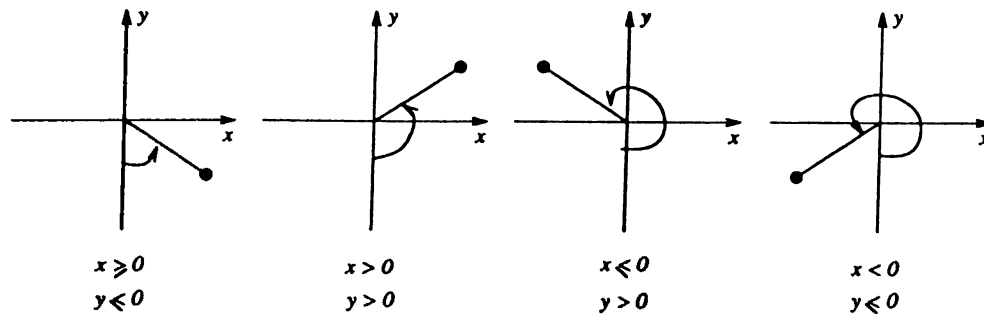


Figure 3.16: Determination of the angle of a control point

shows a suitable mapping that achieves the first smoothness concern. Control point 1 of curve A is mapped to control point 1 of curve B and to 1 of curve C . Control point 2 of A is mapped to control points 2 and 3 of B , control point 2 of B is mapped to control points 2 and 3 of C , and the other control points are mapped in a similar manner. The second smoothness concern is achieved by generating the in-between values of the control points of the curves using the interpolation scheme mentioned previously. The mapped control points are given to the interpolation process as key values and the in-between values are obtained that are shown as holes in the figure. The in-between profile and trajectory curves obtained in this way are used to generate the in-between extended volumes so that when displayed one after the other, shows a smooth shape change on the volume.

A preprocessing is done on the curves for mapping and some angles are obtained for each control point of the curve. These angles are then used in the mapping algorithm. In this preprocessing step, if the curve is not in two-dimensions then it is mapped into two-dimensions. Then for each vertex, an angle is obtained in the range of $0-360$ degrees as seen in Figure 3.16 and the index of the smallest angle is kept as starting index. These angles are used in the mapping algorithm of two curves as seen in Figure 3.17, where first curve has more number of control points than the second one. This algorithm does not guarantee that two curves that have great discriminations will be mapped suitably. But it gives acceptable

```

curve1 = first curve
curve2 = second curve
i = starting index of curve1
j = starting index of curve2
{In the following lines, subscript i is used for a vertex of
curve1, subscript j is used for a vertex of curve2}
map vertexj to vertexi
for each vertexi
  repeat
    if angle(vertexi) > angle(vertexi-1) then
      find a vertexj that has a greater angle from the
      lastly mapped vertex of curve2
    else if angle(vertexi) < angle(vertexi-1) then
      find a vertexj that has a smaller angle from the
      lastly mapped vertex of curve2
  until abs(angle(vertexj) - angle(vertexi)) <
        abs(angle(vertexj+1) - angle(vertexi))
  map the vertexj to vertexi
  if some vertices are left unmapped on curve2 then
    map those remaining vertices to vertexi

```

Figure 3.17: Algorithm for mapping two curves

results in mapping of two curves that do not have great discriminations. Therefore, giving key shape values with small incremental changes will produce a better mapping.

3.3.4 Interpolation of Color

In our system, the color parameters are represented by RGB color model. This model uses a Cartesian coordinate system. The RGB primaries are additive such that every color can be represented as an ordered triple (*r*, *g*, *b*) whose components indicate the relative amounts of each primary color to be added

together [30]. The RGB color model is of interest because it is the color model used in color TV monitors and in many raster displays [13]. The Young-Helmholz theory of color [5] states that there are three basic colors (red, green, and blue), and three kinds of cones in the eye, each most sensitive to one of these three primary colors [30], hence this is another reason for the popularity of the RGB color model.

Color interpolation is done for smooth-shadings, like Gouraud [15] shading, as follows:

$$\begin{aligned} C &= (1 - u)S + uE \\ &= S + u(E - S); \quad 0 \leq u \leq 1 \end{aligned} \quad (3.36)$$

where S is the starting color, E is the end color and C is the interpolated color obtained.

Since the colors red, green and blue are linearly independent, meaning that none of them can be formed by adding appropriate amounts of the other two colors, they can be interpolated as follows extended from equation 3.36 [30].

$$\begin{bmatrix} r_c \\ g_c \\ b_c \end{bmatrix} = \begin{bmatrix} r_s \\ g_s \\ b_s \end{bmatrix} + u \left\{ \begin{bmatrix} r_e \\ g_e \\ b_e \end{bmatrix} - \begin{bmatrix} r_s \\ g_s \\ b_s \end{bmatrix} \right\} = \begin{bmatrix} r_s + u(r_e - r_s) \\ g_s + u(g_e - g_s) \\ b_s + u(b_e - b_s) \end{bmatrix} \quad (3.37)$$

Based on the linear independence of the r, g, b components, we applied the interpolation schema, piecewise cubic Hermite interpolation, explained in Section 3.3.1, to the key values, for each color component and obtained smooth in-between colors passing through the key values.

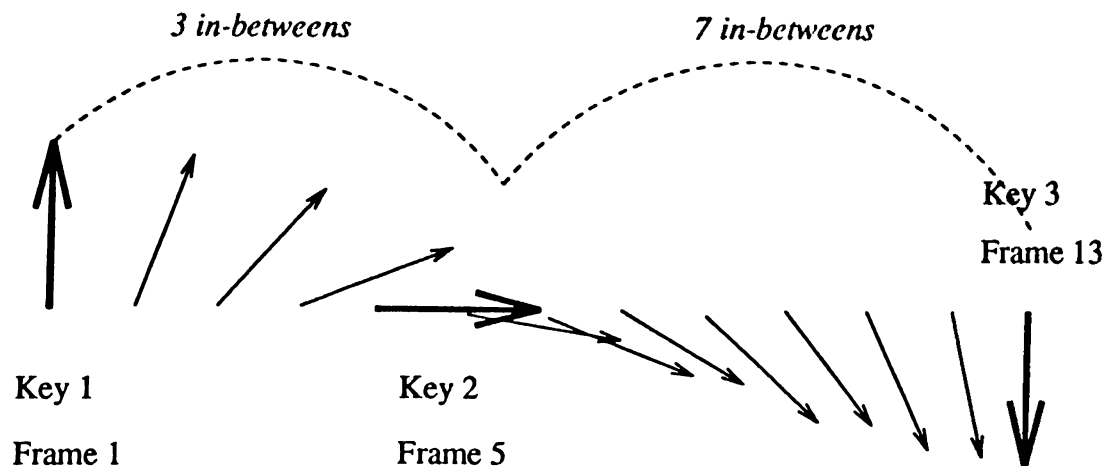


Figure 3.18: Varying number of in-betweens

3.4 Kinetic Control

Sufficient and good kinetic control, without any dynamic concepts such as forces, masses, torques and their expensive calculations for an animation, can create a good illusion of dynamics in the motion. A motion may appear correct spatially but be inappropriate temporally. Therefore an animation system must provide good and easy kinetic control over the animation. The possibility of altering the kinetics of the animation parameters independent from each other is a challenge in an animation system [24].

In our system, the animation parameters are interpolated and their in-between values are generated as explained in the previous section. The resultant in-between values of the parameters differ according to the kinetic control applied on them. We provide constant and varying number of in-between generations between two key values in kinetic control.

In varying number of in-between generations, the key values of animation parameters are given at key times. Different parameters may have different key times for their key values. The only limitation in their key times is that, the key

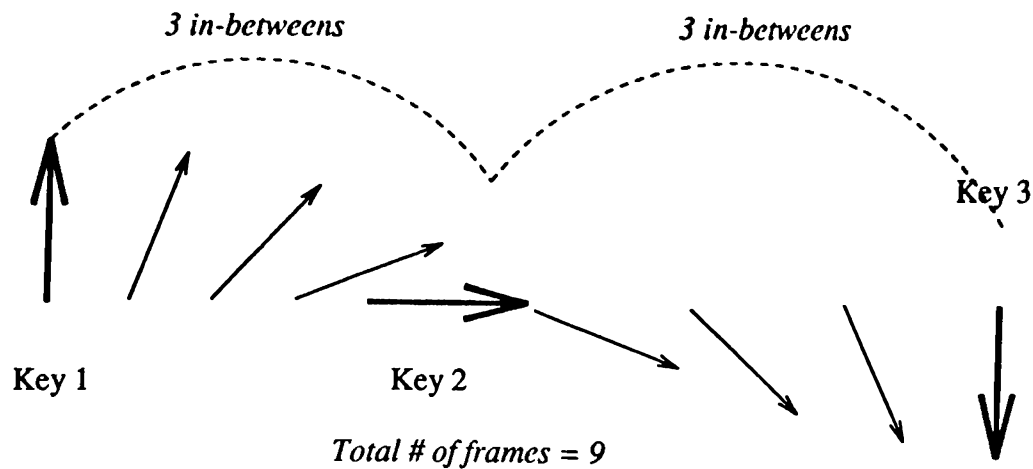


Figure 3.19: Constant number of in-betweens

values of every animation parameter used in that specific animation sequence must be given at the first and last key times of the sequence. Then, for each parameter in the animation, the key values are interpolated as explained previously, but the in-between values generated depend on the key times of the key values, as seen in Figure 3.18.

In constant number of in-between generations, the key values of the parameters are interpolated according to the total number of frames to be displayed for this sequence, without dealing with any key times. Equal number of in-between values are generated between each key value for each parameter, as seen in Figure 3.19.

In both of the kinetic types, the notions of timing (speed) and slow-in, slow-out are included. As explained in [16], *timing*, or speed of an action, is an important principle in animation because the meaning of the action comes from its speed. The speed reflects the weight of the object being animated and it can even give an emotional meaning to the action such that according to its speed a character may seem excited, nervous or relaxed. Two objects with identical size and shape can appear to have very different weights according to the timing of their motion. A heavy body moves slower than a light body. The weight effect that an animating

object gives depend directly on the spacing of its in-between values, not on the in-between values themselves. This spacing of in-between values is specified and determined according to the timing information. *Slow-in* and *out* deals with the spacing of in-between values between the key values. In early animation, the motion was limited to mainly fast and slow moves with even spacing between the in-betweens. Later, the animators found that grouping the in-betweens closer to each key with only one fleeting in-between halfway between, they could achieve an interesting result, slowing out from one key pose, then slowing in to the next key pose. This situation is achieved by the tension, continuity and bias controls on the interpolation scheme as explained in Section 3.3.1.

An on off parameter is introduced to the simple sequence during the specification of its key-frames. If this parameter is set to off then the in-between frames will not be displayed at that time interval. This control is especially useful when there are more than one camera or light in a sequence. Different cameras and lights can be set on or off by the help of this parameter during their sequences at desired intervals. One more advantage of this parameter is seen in hierarchic sequences explained in Section 4.2.2.

Chapter 4

Motion Abstraction System: High Level Controls

In our system high level controls are provided by the help of motion abstraction and object orientation. These high level controls provide the user 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. A detailed explanation was given about what we mean by motion abstraction in Section 2.2.

The highest level controls available for the user are forwarding and backwarding a sequence and framing to the specified frames through the sequence. By forwarding a sequence setting the camera to display in wire frame, a quick shooting can be done to verify the animation. Forward, backward and frame commands achieve their job according to the properties of the sequence under consideration. These properties are explained in detail in the following sections. The forward, backward and frame commands are controlled by the highest authority in the system, the *controller*. Forwarding and backwarding of a sequence are called by the controller so that at each call one frame is generated. As long as the sequence

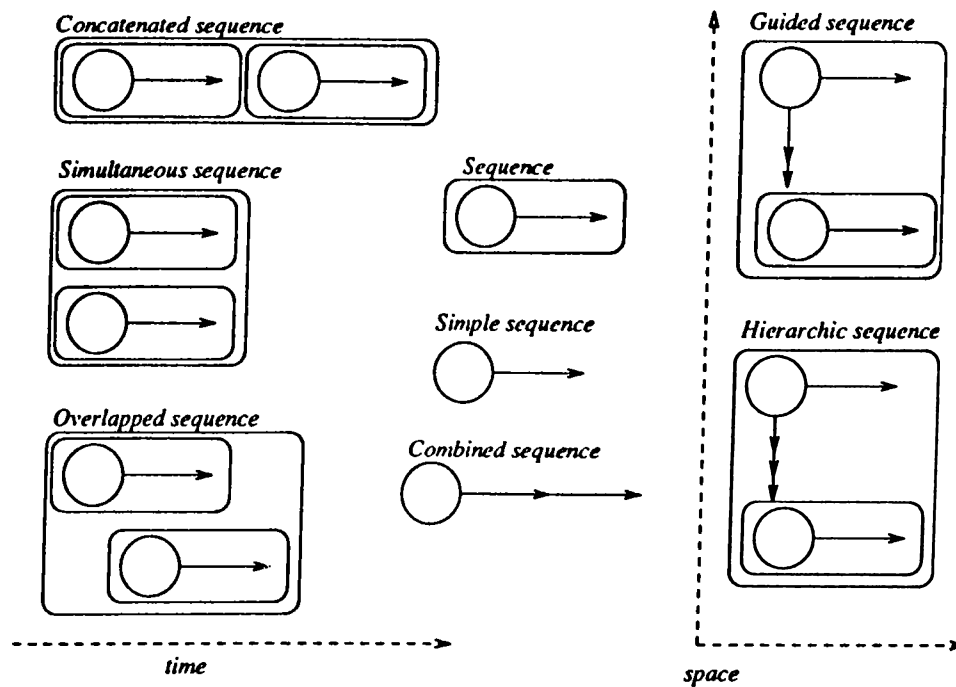


Figure 4.1: Visual representations of sequences

has not ended, it will be forwarded or backwarded again and again until it ends, and forward or backward returns a zero result. The forwarding and backwarding of a sequence is achieved in a recursive manner according to the properties of its building blocks as will be explained in the following sections.

Some brief explanation about *sequence* class was given in Section 3.1. This is the most important class in our system because it supports the motion abstraction concept. In Figure 4.1, 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.

A sequence has some important methods as follows:

- *forward()*, runs the sequence in forward direction,
- *backward()*, runs the sequence in backward direction,

- *framer(frame_no)*, displays the specified frame of the sequence,
- *total_frame_number()*, returns the total number of frames that this sequence will last, and
- *current_frame_number()*, returns the current frame number that this sequence is at.

All these methods differ in their implementation in different sequence classes of the sequence hierarchy. These methods, for each sequence, will be explained in their corresponding sections.

As seen in Figure 3.7, there are basically two kinds of sequences, *simple sequence* and *compound sequence*.

4.1 Simple Sequence

A simple sequence consists of a transformable object, actor, light or camera, and a frame data where the key values and the key times of the animation parameters used in this sequence are specified, as seen in Figure 3.8. A simple sequence generated for a transformable object can be used for any other transformable object as long as there is no shape parameter in the frames data of the sequence. The reason for this limitation is that, the key values for the shape parameter generally depend on the transformable object of the sequence so using such a sequence for a very different transformable object may produce an undesirable result.

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

When a simple sequence is forwarded or backwarded, its attribute, current frame number is incremented or decremented. 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 frames data of the sequence, and these parameter values are applied to the transformable object that this sequence acts on. This process continues until the local current frame number of the sequence reaches to its total frame number and hence the sequence ends. When the simple sequence has ended it returns 0 from forward and backward calls. A simple sequence that has reached to its end does nothing when it is forwarded or backwarded again and again, just returns 0 from those calls, until it is reset to its beginning.

When the simple sequence is framed to a particular frame number, the process performed is similar to forwarding or backwarding, except it is assumed that, the sequence has been run until the previous frame. To manage this, the current frame number is set to the previous frame number artificially and the next frame is displayed, and the process stops.

4.1.1 Combined Sequence

This is a simple sequence generated by the combination of two simple sequences. There are two ways of combining them, denoted by an attribute in this class named **attached**. 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 nonattached 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. In the first way, attached combination, the second sequence is modified to fit to the end of the first sequence while in the second way, nonattached combination, the second sequence is not affected but some in-between values are introduced between the sequences to manage a smooth transition between them. This concept of smooth transition between motions has been dealt in [24], in a manner called *phrased transition*. By phrased transition, it is meant that an alteration in the nature of the motions is done so that they become perceptually integrated into one single motion producing a transition that flows smoothly.

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 made of any two independent sequences, either simple or compound. As seen in Figure 3.7, 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 be again any compound sequence or a simple sequence. Hence there is a recursive definition on this class that leads to a recursive definition again 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. By recursively instantiating all the sequences in this manner, resultantly the lowest level sequences, simple sequences, are 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 behaviors of their building block sequences. The building blocks of such a sequence have their own spatiotemporal behavior and behave according to their local information. However, this local behavior of the building sequences is converted into a global behavior with temporal dependence. This is achieved in terms of time. The temporal behavior 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 behavior is generated at this level without touching the spatial behaviors of the individual sequences. Fiume et al. have developed a temporal scripting language for object oriented animation [12] as we have mentioned in Section 2.3 briefly. Our work has some similarities with theirs in terms of temporal behavior, however we did not implement any scripting language that has some overheads as mentioned previously.

Concatenated Sequence

This is a temporally dependent compound sequence, made up of two independent sequences, inheriting the features of its parent class, compound sequence. The two independent sequences are made temporally dependent under the abstraction of concatenated sequence where the second sequence starts its execution just after the first one ends.


```
forward()
  if sequence1->forward() != 0 then
    return(-1)
  if sequence2->forward() != 0 then
    return(-1)
  else
    return(0)
```

As seen above, a concatenated sequence is forwarded by forwarding its first sequence until it ends. After it is ended returning 0 from its forward call, the second sequence starts execution. The concatenated sequence ends when its second sequence ends, and returns 0 from forward.

```
current_frame_number()
  if sequence1->current_frame_number() <
    sequence1->total_frame_number() then
    return (sequence1->current_frame_number())
  else
    return (sequence1->total_frame_number() +
           sequence2->current_frame_number())
```

The current frame number of a concatenated sequence is the current frame number of its first sequence if it did not reach to its end yet. 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 executing its second sequence.

```
total_frame_number()
    return (sequence1->total_frame_number() +
           sequence2->total_frame_number())
```

Total frame number that a concatenated sequence has is the sum of the total frame numbers of its sequences.

Simultaneous Sequence

This is a temporally dependent compound sequence where the two sequences start execution at the same time.

```
forward()
    if sequence1->forward() != 0 ||
       sequence2->forward() != 0 then
        return(-1)
    else
        return(0)
```

A simultaneous sequence is forwarded by forwarding both of its sequences at the same time. Such a sequence ends when the longer of its sequences ends.

```
current_frame_number()  
  if sequence1->total_frame_number() >  
    sequence2->total_frame_number() then  
      return (sequence1->current_frame_number())  
  else  
    return (sequence2->current_frame_number())
```

The current frame number of a simultaneous sequence is the current frame number of its longer sequence.

```
total_frame_number()  
  if sequence1->total_frame_number() >  
    sequence2->total_frame_number() then  
      return (sequence1->total_frame_number())  
  else  
    return (sequence2->total_frame_number())
```

And the total frame number of a simultaneous sequence is the total frame number of its longer sequence.

Overlapped Sequence

Overlapped sequence is the last temporally dependent compound sequence that our system provides. In such a sequence, the second sequence overlaps the first sequence starting execution at the specified overlapping frame number.

```

forward()
  if sequence1->current_frame_no() < overlap_frm_no then
    return (sequence1->forward())
  else if sequence1->forward() != 0 || sequence2->forward() != 0 then
    return(-1)
  else
    return(0)

```

It is forwarded by only forwarding its first sequence until the overlapping frame is reached. After that point, it does not have any difference from the simultaneous forwarding. It ends when the longer of its sequences ends.

```

current_frame_number()
  if sequence1->current_frame_number() <
    sequence1->total_frame_number() then
    return (sequence1->current_frame_number())
  else
    return (overlap_frm_no + sequence2->current_frame_number())

```

If it is currently executing its first sequence, then the current frame number of a simultaneous 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.

```
total_frame_number()
  if sequence1->total_frame_number() >
    overlap_frm_no + sequence2->total_frame_number() then
    return (sequence1->total_frame_number())
  else
    return (overlap_frm_no + sequence2->current_frame_number())
```

If the total frame number of its first sequence is greater than the sum of its overlapping frame number and the total frame number of its second sequence then the total frame number is the total frame number of its first sequence. Otherwise, it is the sum of its overlapping frame number and the total frame number of its second sequence.

4.2.2 Spatially Dependent Sequences

These sequences are compound sequences created according to the spatial behaviors of their building block sequences. The building blocks of these sequences have their own spatiotemporal behavior but under the control of such a sequence, the spatial behaviors of them get modified. The spatially dependent sequences have one of their building blocks as a simple sequence. This simple sequence plays the parent role while the other building block can be any kind of sequence playing the child role. The spatial behavior of the child sequence is modified according to the spatial behavior of the parent sequence. The first sequence is the parent sequence and the second one is the child sequence. The temporal behavior of such sequences are straight forward as seen below.

```

forward()
  if sequence1->forward() != 0 then
    sequence2->forward()
    return(-1)
  else
    return(0)

```

The sequences are forwarded simultaneously until the parent sequence ends.

```

current_frame_number()
  return (sequence1->current_frame_number())

```

```

total_frame_number()
  return (sequence1->total_frame_number())

```

The current and total frame numbers depend only on the parent sequence.

Guided Sequence

Guided sequence is a spatially dependent compound sequence where the spatial behavior of the child sequence is modified so that it is being guided by its parent. Figure 4.2 shows examples of spatially dependent sequences. In this figure the sequences drawn by the dotted lines are the child sequences. As seen in that figure, in 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

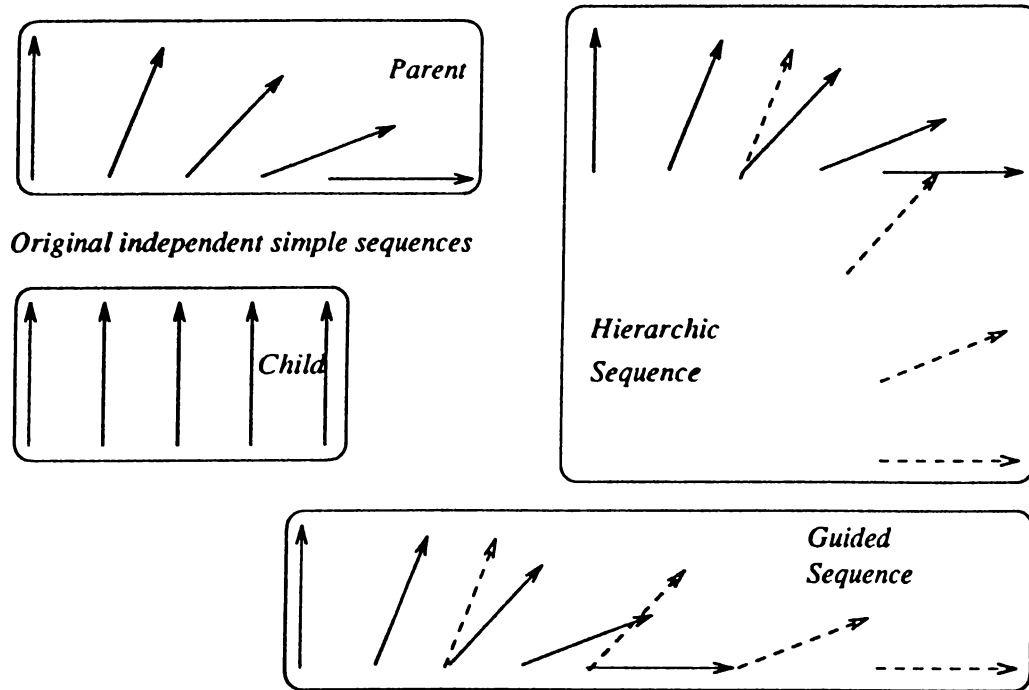


Figure 4.2: Spatially dependent sequences

a frame i is as follows:

$$\begin{aligned}
 result_size_i &= size_i \times parent_size_i \\
 result_orientation_i &= orientation_i \times parent_orientation_i \\
 result_position_i &= position_i + parent_position_i
 \end{aligned} \tag{4.1}$$

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

$$((point \times result_size_i) \% result_orientation_i) + result_position_i \tag{4.2}$$

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

Hierarchic Sequence

This is the other spatially dependent compound sequence. This behavior is exactly the hierarchic behavior seen at a linked body. However in our system, the

definition for linking bodies to generate hierarchic action is done in 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 by the help of motion abstractions at a sequence. Thus geometric structure can change during a sequence. One more advantage is the easy and reusable motion definitions as well as other sequence types that are supported in our system. The definition of the child sequence is very easy by the help of 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 4.2, in 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:

$$T = [orientation_i][position_i][parent_orientation_i][parent_position_i] \quad (4.3)$$

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 way.

By the help of the hierarchic sequence definition and the 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.

Chapter 5

Conclusion

Our system is a motion abstraction system designed and implemented using object oriented techniques. Due to object orientation in its design, it contains modular, extensible and re-usable code. Object-orientation is advantageous for animation both in the implementation of the system and generation of the animation sequences. New sequence or transformable object classes can be inserted to the system in their hierarchy.

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 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 lowest levels as well. This brings re-usability of existing sequences and flexibility in the creation of new ones with concise and precise definitions. For the specification of the animation sequences

no scripting language is needed which has some overheads like parsing and experience of programming. Instead, an interactive environment is provided at all levels that gives ease, flexibility and speed during specification. The created sequences, key-frames or graphical objects, briefly everything created in the system, can be saved in their specific libraries to support re-usability. 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.

Extended volumes are used in the system that can have shape change either by changing the weights used in the smoothing NURBS approximation applied over them or by changing the shapes of their profile or trajectory curves. Shape interpolation done in this way produces better results if it is applied on small incrementally changed shapes.

The lowest level sequences are generated by parametric key-frame interpolation and Hermite spline interpolation is applied on the key values with tension, continuity and bias parameters that provides slow-in, slow-out effects.

Quaternions are used instead of Euler's angles and matrix representation of rotations that cause some overheads. Although the common approach to rotations is by using Euler's angles, it is not an ideal solution since Euler's angles must be used in the given order as rotations are not commutative. Instead quaternions with interpolation on the four-dimensional unit sphere, is a better way of representing and applying rotations than doing it with Euler's angles using the matrix representation. Quaternions are compact and require less number of 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 behaviored sequences are joined and their global temporal behavior

is generated according to their temporal dependency. In the second one, not the temporal behavior but the spatial behavior of the sequences is affected by their dependency.

As a result, this motion abstraction system provides an environment to generate flexible, re-usable, concise and precise animation sequences. As a future work, new sequence or transformable object classes, other interpolation techniques and new animation parameters can be introduced to the system, and shape interpolation can be improved. These can be done without much effort because of the object-oriented design of the system.

Appendix A

Derivations

In this appendix the capital letters denote the skew symmetric matrix of the vectors denoted by small letters. Representations like $A\vec{b}$ means $\vec{a} \times \vec{b}$. Other rules about vector algebra can be found in [28]. All the derivations in this appendix are from the course notes MATH 672 [29].

A.1 Conversion from a rotation matrix to a unit quaternion

Let T be an orthogonal 3×3 matrix representing a rotation. It can be written as in equation A.1 where \vec{n} and α are the axis and angle of rotation respectively.

$$T(\vec{n}, \alpha) = \cos \alpha I + (1 - \cos \alpha)\vec{n}\vec{n}^T + \sin \alpha N \quad \|\vec{n}\| = 1 \quad (\text{A.1})$$

S and K are the symmetric and skew symmetric parts of T respectively.

$$\begin{aligned} T &= S + K \\ S &= \frac{1}{2}(T + T^T) \end{aligned} \quad (\text{A.2})$$

$$\begin{aligned}
&= \cos \alpha I + (1 - \cos \alpha) \vec{n} \vec{n}^T \\
K &= \frac{1}{2}(T - T^T) \\
&= \sin \alpha N
\end{aligned}$$

The axis of rotation, \vec{n} , can be obtained from the skew symmetric part as

$$\begin{aligned}
\vec{k} &= \sin \alpha \vec{n} \\
\vec{n} &= \frac{1}{\sin \alpha} \vec{k}
\end{aligned} \tag{A.3}$$

and the angle of rotation, α , can be obtained from the trace of the matrix T .

$$\begin{aligned}
\text{trace}(T(\vec{n}, \alpha)) &= \text{trace}(S) + \text{trace}(K) \\
&= \cos \alpha \text{trace}(I) + (1 - \cos \alpha) \text{trace}(\vec{n} \vec{n}^T) + \sin \alpha \text{trace}(N) \\
&= 3 \cos \alpha + (1 - \cos \alpha) + 0 \\
&= 1 + 2 \cos \alpha \\
\cos \alpha &= \frac{1}{2}(\text{trace}(T(\vec{n}, \alpha)) - 1)
\end{aligned} \tag{A.4}$$

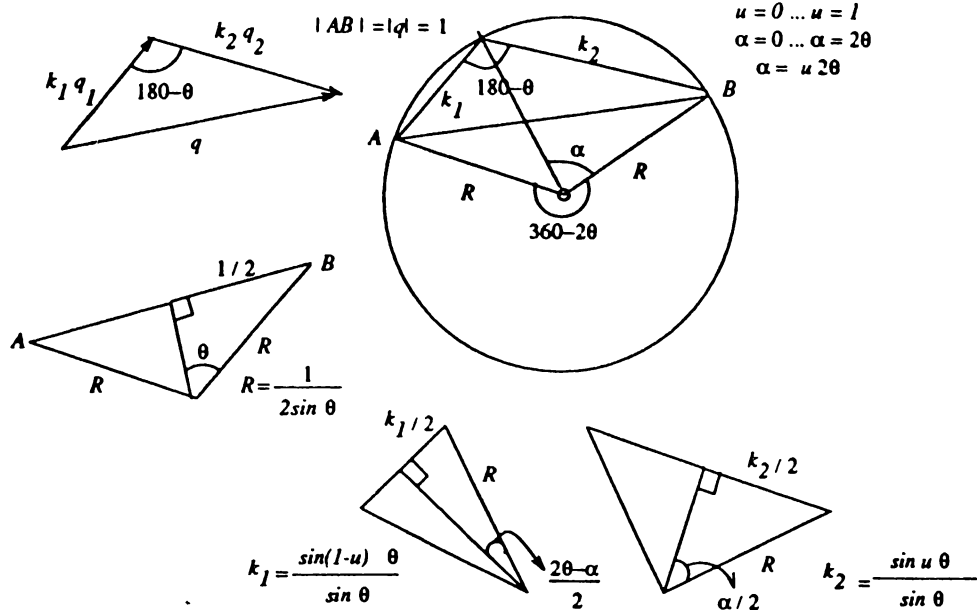
A quaternion $q = [s, \vec{v}]^T$ that represents a rotation with angle α and rotation axis \vec{n} is

$$\vec{v} = \sin(\alpha/2) \vec{n} \quad , \quad s = \cos(\alpha/2) \tag{A.5}$$

A.2 Conversion from a unit quaternion to a rotation matrix

Given a quaternion $q = [\xi, \eta, \zeta, \chi]^T$ it can be converted to a rotation matrix as follows:

$$T(\vec{n}, \alpha) = \begin{bmatrix} 2(\xi^2 + \chi^2) - 1 & 2(\xi\eta - \zeta\chi) & 2(\xi\zeta + \eta\chi) \\ 2(\xi\eta + \zeta\chi) & 2(\eta^2 + \chi^2) - 1 & 2(\eta\zeta - \xi\chi) \\ 2(\xi\zeta - \eta\chi) & 2(\eta\zeta + \xi\chi) & 2(\zeta^2 + \chi^2) - 1 \end{bmatrix} \tag{A.6}$$


 Figure A.1: Derivation of the *slerp* formula

A.3 Derivation of the spherical linear interpolation formula

The interpolated unit quaternion q of two unit quaternions q_1, q_2 can be obtained as follows:

$$\begin{aligned} q &= \text{slerp}(q_1, q_2, u) & 0 \leq u \leq 1 \\ &= k_1 q_1 + k_2 q_2 & \|q\| = \|q_1\| = \|q_2\| = 1 \end{aligned} \quad (\text{A.7})$$

where k_1 and k_2 are functions of u . As $\|q\| = 1$ we can continue as

$$\begin{aligned} q^T q &= (k_1 q_1^T + k_2 q_2^T)(k_1 q_1 + k_2 q_2) \\ 1 &= k_1^2 q_1^T q_1 + k_1 k_2 q_1^T q_2 + k_2 k_1 q_2^T q_1 + k_2^2 q_2^T q_2 \\ 1 &= k_1^2 \|q_1\|^2 + k_2^2 \|q_2\|^2 + 2k_1 k_2 q_1^T q_2 \\ 1 &= k_1^2 \|q_1\|^2 + k_2^2 \|q_2\|^2 + 2k_1 k_2 \|q_1\| \|q_2\| \cos \theta \\ 1 &= k_1^2 + k_2^2 - 2k_1 k_2 \cos(180 - \theta) \end{aligned}$$

From Figure A.1, we obtain that

$$k_1(u) = \frac{\sin(1-u)\theta}{\sin\theta} \quad k_2(u) = \frac{\sin u\theta}{\sin\theta} \quad (\text{A.8})$$

A.4 Rotation of a vector by a quaternion

We can rewrite equation A.1 using the information in equation A.5 as

$$T(\vec{n}, \alpha) = (s^2 - \vec{v}^T \vec{v})I + 2\vec{v}\vec{v}^T + 2sV \quad (\text{A.9})$$

A vector \vec{r} can be rotated by a quaternion $q = [s, \vec{v}]^T$ to obtain \vec{r}' as seen below. In order to do quaternion multiplication, the vector can be rewritten as a quaternion which has no rotational part as $[0, \vec{r}]^T$.

$$\begin{aligned} [0, \vec{r}'] &= q[0, \vec{r}]q^{-1} \\ &= [s, \vec{v}][0, \vec{r}][s, -\vec{v}] \\ &= [s, \vec{v}][\vec{v}^T \vec{r}, s\vec{r} + V\vec{r}] \\ &= [s\vec{v}^T \vec{r} - s\vec{v}^T \vec{r} - \vec{v}^T V\vec{r}, s^2 \vec{r} + sV\vec{r} + \vec{v}\vec{v}^T \vec{r} + sV\vec{r} + VV\vec{r}] \\ &= [0, s^2 \vec{r} + 2sV\vec{r} + \vec{v}\vec{v}^T \vec{r} + (\vec{v}\vec{v}^T - (\vec{v}^T \vec{v})I)\vec{r}] \\ &= [0, s^2 \vec{r} + 2sV\vec{r} + \vec{v}\vec{v}^T \vec{r} + \vec{v}\vec{v}^T \vec{r} - \vec{v}^T \vec{v} \vec{r}] \\ &= [0, (s^2 - \vec{v}^T \vec{v})\vec{r} + 2sV\vec{r} + 2\vec{v}\vec{v}^T \vec{r}] \\ &= [0, ((s^2 - \vec{v}^T \vec{v})I + 2\vec{v}\vec{v}^T)\vec{r} + 2sV] \end{aligned} \quad (\text{A.10})$$

Hence the rotated vector \vec{r}' is

$$\begin{aligned} \vec{r}' &= ((s^2 - \vec{v}^T \vec{v})I + 2\vec{v}\vec{v}^T + 2sV)\vec{r} \\ \vec{r}' &= T(\vec{n}, \alpha)\vec{r} \end{aligned} \quad (\text{A.11})$$

from equation A.9.

Appendix B

Colored Images

In this appendix, some colored images are presented for giving a visual presentation of the animation sequences generated by our system. The frames of the sequences seen in the first four figures were rendered by Phong shading that is available in our motion abstraction system. The last one was rendered by radiosity methods [8].

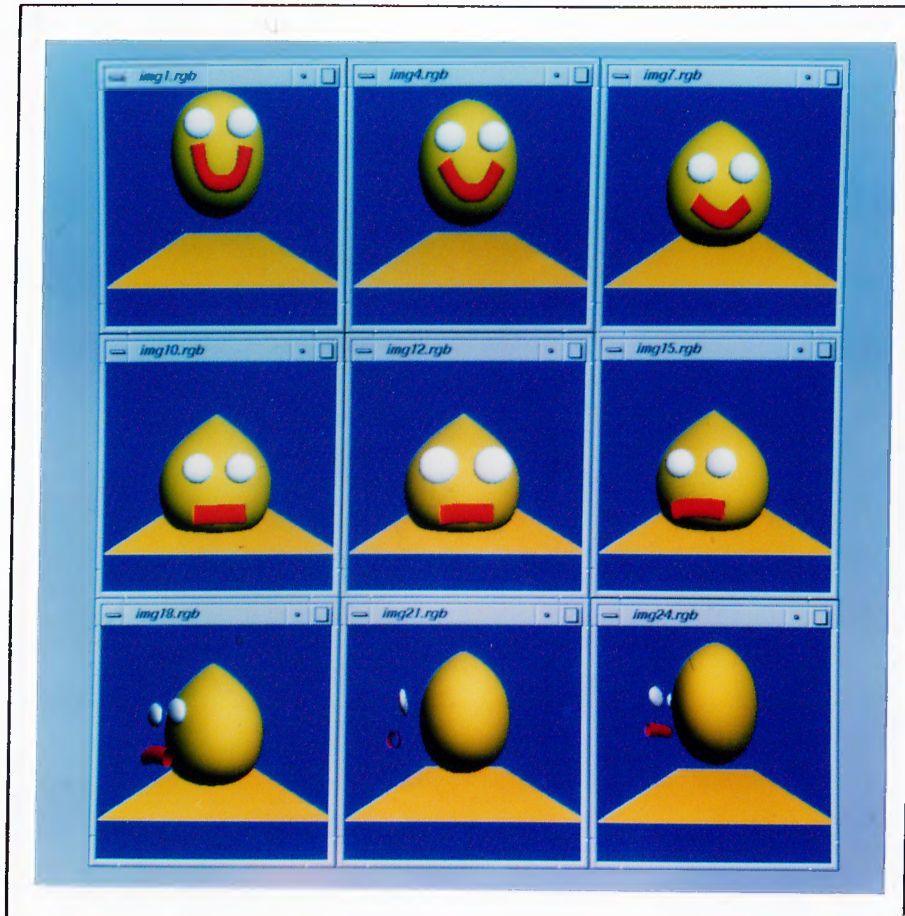


Figure B.1: Frames from the animation Jellybon

The sequence seen in Figure B.1 presents an hierarchic sequence generated by making the simple sequences of mouth and eyes hierarchically dependent to the simple sequence of the head. Then this hierarchic sequence is made simultaneous with the simple sequence of the floor. Hence the resultant sequence is a simultaneous sequence generated by making a simple sequence and a hierarchic sequence temporally dependent as a simultaneous sequence. The head and mouth have shape changes in their simple sequences.



Figure B.2: Frames from the animation Spin

The sequence seen in Figure B.2 presents a simple sequence with color interpolation. The spin rotates around its vertical axis and after some time it gets slower and stops.

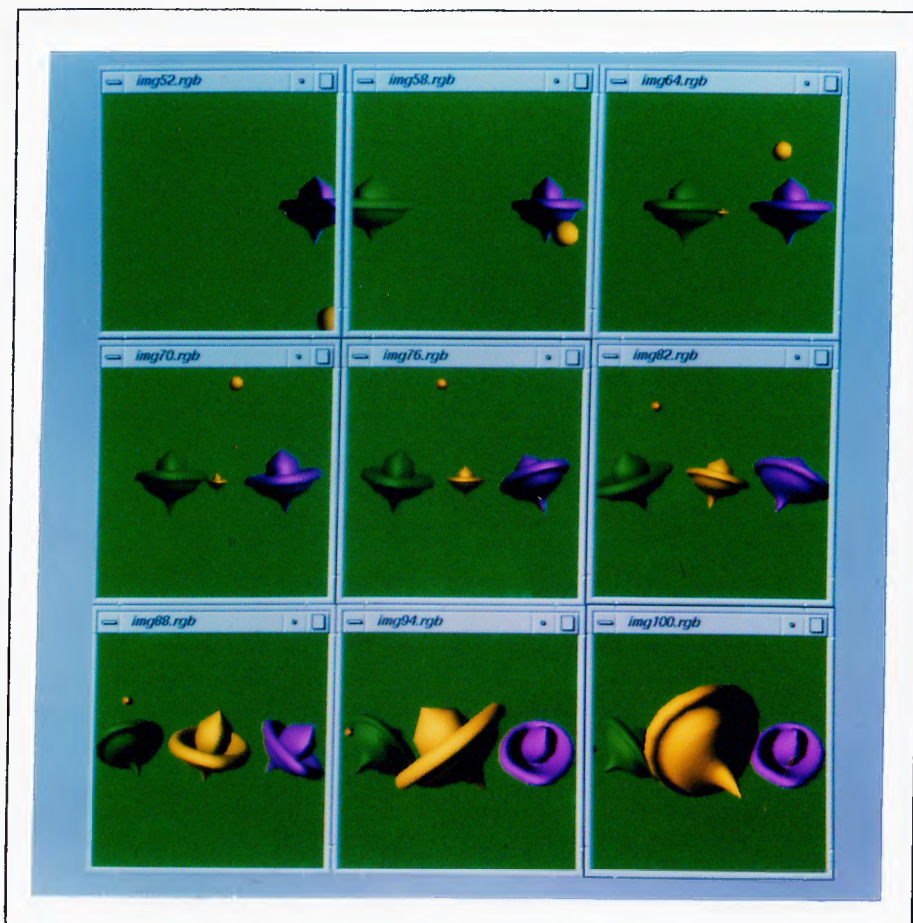


Figure B.3: Frames from the animation Sunny

In Figure B.3, the sequence of spin seen in Figure B.2 is used as building block and an overlapped sequence is generated by two of these sequences. Then this sequence is overlapped with the spin sequence once again. The initial positions and orientations of the spins are given by hierarchic dependence to initializer simple sequences 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 into lower left to pointing into lower right.

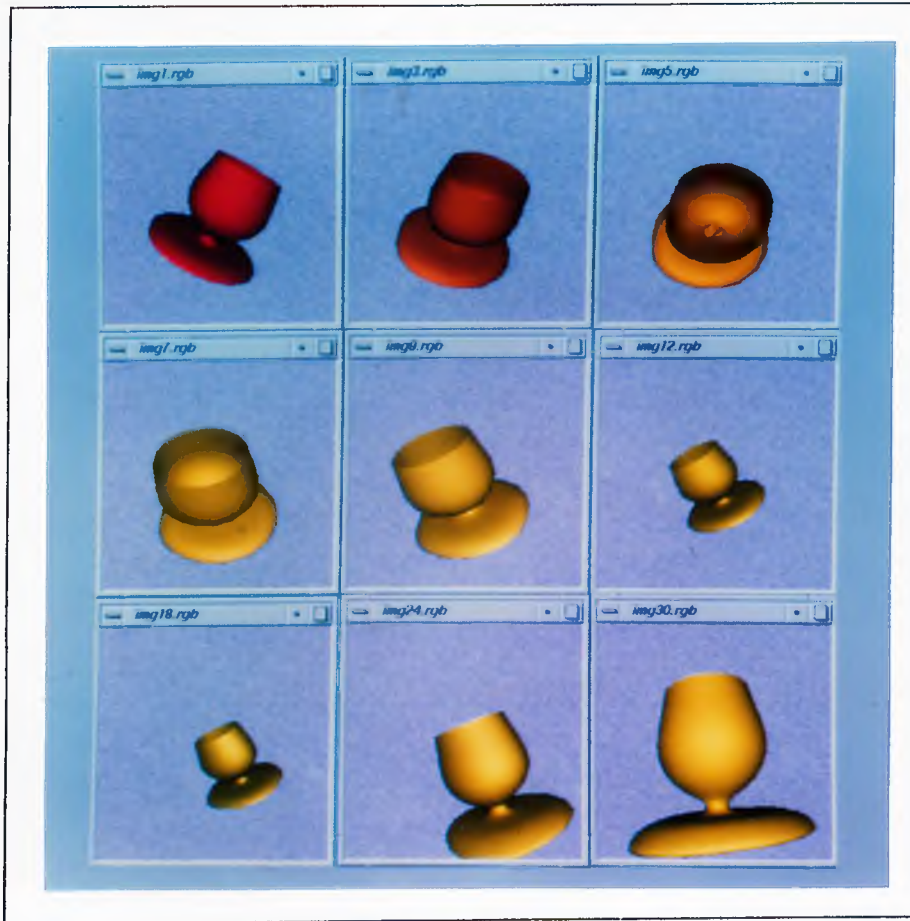


Figure B.4: Frames from the animation Glass

In Figure B.4, a simultaneous sequence generated by the sequences of camera and glass is presented. The glass rotates and changes color, then the camera changes position and orientation.

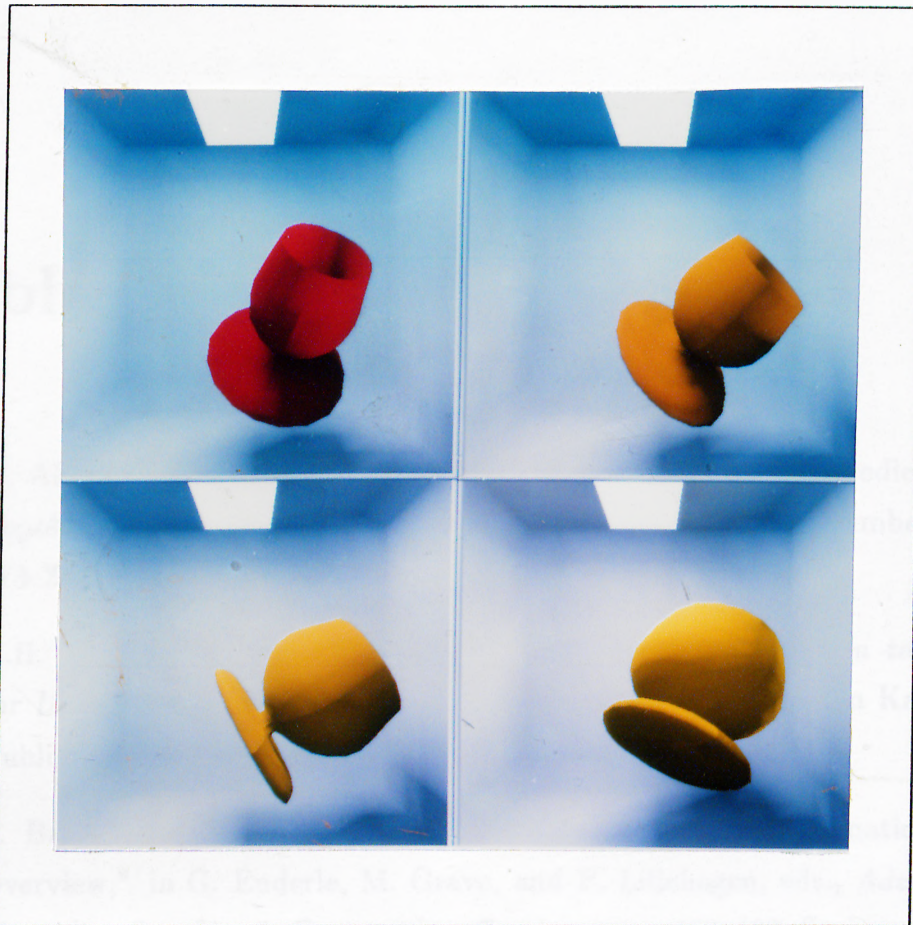


Figure B.5: Frames from the animation Glass rendered by radiosity

Figure B.5 presents the sequence of glass rendered by the radiosity renderer developed on the IPSC/2 Hypercube in Bilkent University [8].

Bibliography

- [1] V. Akman and A. Arslan, "Sweeping with all Graphical Ingredients in a Topological Picturebook," *Computers & Graphics*, vol. 16, number 3, pp. 273–281, 1992.
- [2] R.H. Bartels, J.C. Beatty, and B.A. Barsky, *An Introduction to Splines for Use in Computer Graphics & Geometric Modeling*. Morgan Kaufmann Publishers, Inc., 1987.
- [3] P. Baudelaire and M. Gangnet, "Computer Assisted Animation - An Overview," in G. Enderle, M. Grave, and F. Lillehagen, eds., *Advances in Computer Graphics 1, Eurographics Seminars*, pp. 469–498. Springer Verlag, 1986.
- [4] S. Bergman and A. Kaufman, "BGRAF2: A Real-time Graphics Language with Modular Objects and Implicit Dynamics," *SIGGRAPH '76, Computer Graphics*, pp. 133–138, July 1976.
- [5] F. M. Branley, *Color from Rainbows to Lasers*. Thomas Y. Crowell Co., New York, 1978.
- [6] 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*, pp. 275–282. North Holland, 1987.

- [7] D.G. Cachola and G.F. Schrack, "Modeling and Animating Three-Dimensional Articulate Figures," *Graphics Interface '86 Vision Interface '86*, pp. 152–157, 1986.
- [8] 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)*, 1993.
- [9] M. Chmilar, B. Wyvill, and C. Herr, "A Software Architecture for Integrating Modeling with Kinematic and Dynamic Animation," *The Visual Computer*, vol. 7, pp. 122–137, 1991.
- [10] 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*, pp. 15–26. Springer Verlag, 1988.
- [11] G. Farin, *Curves and Surfaces for Computer Aided Geometric Design, A Practical Guide, Second Edition*. Academic Press, Inc., 1990.
- [12] E. Fiume, D. Tsichritzis, and L. Dami, "A Temporal Scripting Language for Object-Oriented Animation," in G. Marechal, ed., *Eurographics '87*, pp. 283–294. North Holland, 1987.
- [13] J. D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, 1982.
- [14] P. Getto and D. Breen, "An Object-Oriented Architecture for a Computer Animation System," *The Visual Computer*, vol. 6, pp. 79–92, 1990.
- [15] H. Gouraud, "Continuous Shading of Curved Surfaces," *IEEE Transactions on Computers*, vol. C-20, number 6, pp. 623–628, June 1971.
- [16] J. Lasseter, "Principles of Traditional Animation Applied to 3D Computer Animation," *SIGGRAPH '87, Computer Graphics*, vol. 21, number 4, pp. 35–44, July 1987.

- [17] R. Maiocchi and B. Pernici, "Directing an Animated Scene with Autonomous Actors," *The Visual Computer*, vol. 6, pp. 359–371, 1990.
- [18] L. Piegl, "On NURBS: A Survey," *IEEE Computer Graphics & Applications*, vol. 11, number 1, pp. 55–71, January 1991.
- [19] D. Pletincks, "The Use of Quaternions for Animation, Modeling and Rendering," in N. Magnenat-Thalmann and D. Thalmann, eds., *New Trends in Computer Graphics, Proceedings of CG International*, pp. 44–53. Springer Verlag, 1988.
- [20] C. Reynolds, "Computer Animation with Scripts and Actors," *SIGGRAPH '82, Computer Graphics*, vol. 16, number 3, pp. 289–296, July 1982.
- [21] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*. Prentice Hall, 1991.
- [22] M. Shaw, "The Impacts of Abstractions Concern on Modern Programming Languages," *Proceedings of IEEE '68*, pp. 1119–1130, 1968.
- [23] K. Shoemake, "Animating Rotation with Quaternion Curves," *SIGGRAPH '85, Computer Graphics*, vol. 19, number 3, pp. 245–254, July 1985.
- [24] S.N. Steketee and N.I. Badler, "Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control," *SIGGRAPH '85, Computer Graphics*, vol. 19, number 3, pp. 255–262, July 1985.
- [25] B. Stroustrup, *The C++ Programming Language, 2nd Edition*. Addison-Wesley, 1991.
- [26] N. M. Thalmann and D. Thalmann, *Computer Animation, Theory and Practice*. Springer Verlag, 1985.
- [27] 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*, vol. 5, number 3, pp. 62–73, March 1985.

- [28] Y. Tokad, *Analysis of Engineering Systems*. Bilkent University, 1990.
- [29] Y. Tokad, "MATH 672: Three Dimensional Mechanical Systems," Spring 1992, Bilkent University.
- [30] K. Turkowski, "Anti-Aliasing in Topological Color Spaces," *SIGGRAPH '86, Computer Graphics*, vol. 20, number 4, pp. 307-314, August 1986.
- [31] C. W. A. M. van Overveld, "The Generalized Display Processor as an Approach to Real-time Interactive 3-D Computer Animation," *The Journal of Visualization and Computer Animation*, vol. 2, pp. 16-25, 1991.
- [32] E.T. Whittaker, *Analytical Dynamics*. Cambridge University Press, 1964.
- [33] 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*, vol. 25, number 4, pp. 105-112, July 1991.