A TOOL FOR GENERATING THREE DIMENSIONAL ANIMATION ON COMPUTERS

A THESIS
SUBMITTED TO THE DEPARTMENT OF
GRAPHIC DESIGN
AND THE INSTITUTE OF FINE ARTS
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF FINE ARTS

By
Cemil Şinasi Türün
April, 1991

A TOOL FOR GENERATING THREE DIMENSIONAL ANIMATION ON COMPUTERS

A THESIS

SUBMITTED TO THE DEPARTMENT OF

GRAPHIC DESIGN

AND THE INSTITUTE OF FINE ARTS

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF FINE ARTS

By

Cemil Sinasi Türün

April, 1991

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 Fine Arts.

[illegible]

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 Fine Arts.

[illegible]

Assoc. Prof. Yıldırım Yavuz

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 Fine Arts.

[illegible]

Assist. Prof. Ihsan Derman

Approved by the Institute of Fine Arts

[illegible]

Prof. Bülent Özgüç, Director of the Institute of Fine Arts

ABSTRACT

A TOOL FOR GENERATING THREE DIMENSIONAL ANIMATION ON COMPUTERS

Cemil Sinasi Türün
M.F.A. in Graphical Arts
Supervisor: Prof. Dr. Bülent Özgüç
1991

In this work, a three dimensional computer animation system has been designed to be employed in schools, for the training of art students on basic three dimensional animation techniques. Puppet Theater, as we have called the system, utilizes the flexibility and effectiveness of the low-end hardware, namely IBM PC™ computers supported with Targa 16™ graphics board and gives special emphasis to user friendliness. It is basically a software to design three dimensional objects and choreograph the object data in the computer's memory, before rendering the resulting scenery with shading methods. The system is the result of reflecting the recent advances in the field of computer graphics and pushing the potentials of the existing platform. Software is implemented in C language, thus the code is transportable. A custom designed object oriented windowing system called WODNIW is used as the user interface. This open windowing system supports pull-down menus, interactive buttons, scalable windows and other popular user interface elements.

Keywords: Computer graphics, Animation, Three-dimensional computer animation, Rendering.

ÖZET

BİLGİSAYARLARDA ÜÇ BOYUTLU CANLANDIRMA FİLM ÜRETİMİ İÇİN BİR ARAÇ

Bu çalışmada, güzel sanatlar dalında eğitim veren bir okulda, öğrencilerin üç boyutlu canlandırmanın temellerini öğrenebilmelerini ve canlandırma filmi üretebilmelerini sağlamaya yönelik üç boyutlu bir bilgisayar yazılımı tasarlanmıştır. Kukla Tiyatrosu adını verdiğimiz bu sistem, Targa 16^{TM} grafik kartı takılan düşük maliyetli bilgisayar sistemlerinin esnekliğini ve etkililiğini kullanmakta ve kullanıcı arabirimini olabildiğince anlaşılır kılmaktadır. Temel olarak bilgisayarın belleğinde oluşturulan üç boyutlu modellerin hareketlendirilip sonuçta da gerçeğe yakın, renkli bir görünüme büründürülmesini amaçlayan bir yazılımdır. Bilgisayar grafiği konusundaki son gelişmeleri yansıtan ve eldeki platformun sınırlarını zorlayan sistem, taşınabiliriği ve esnekliği sağlamak için C programlama diliyle geliştirilmiştir. Bu programa arka plan olarak da WODNIW adı verilen bir pencereleme sistemi geliştirilmiştir. Bu açık pencereleme sistemi, en yeni kullanıcı arabirimi araçları olan pull-down menüleri, etkileşimli tuşları, boyu ayarlanabilir pencereleri ve nesneye yönelik operasyonları desteklemektedir.

Anahtar sözcükler: Bilgisayar grafiği, Canlandırma, Üç boyutlu canlandırma film, Renklendirme.

ACKNOWLEDGEMENT

Foremost, I would like to thank my advisor, Prof. Özgüç for his invaluable help, support, and tutorship, without which this thesis would not have been possible. I owe a large part of this thesis to him who has showed me immense patience throughout the last three years.

Secondly, I would like to thank Ferhat Büyükkökten for his continual support, friendship and the many hours he spent in front of the computer with me. His skills in computer programming are frankly exceptional and without him, I would not have been able to figure out how to solve many design problems on my own. My chairperson, Erdag Aksel deserves mention not because of his position but for his continuing trust in my vague ideas and sharing my interdisciplinary desires.

Last, but not least, I would like to thank my colleagues Ahmet Arslan, Veysi Isler, Ogan Ocali and Ugur Güdükbay for the nights spent researching and working on our respective theses. Although we are all concerned with different areas of Computer Graphics, we are, after all, on the same train...

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Computers have been used to generate animated films for over 25 years. Animation, as an art form existed since early 1920's, and hand drawn pictures were received with much enthusiasm and proved to be of educational value throughout our century. After the introduction of the digital computer in 1940's, this form of creativity found its way on the new medium.

The first interactive computer graphics software was written by Ivan Sutherland as his doctoral dissertation submitted to Massachusetts Institute of Technology in 1962. His system was titled "Sketchpad : a man-machine graphical communication system" and presented first computer graphics oriented user interface with a simple paint program. It was operated using a light pen and specified the relationships of graphic elements like angles, sizes, by means of a keyboard which was used with the other hand. Later it was improved to "Sketchpad III" by Timothy Johnson by allowing the operator to draw and visualize objects in three dimensions.

While technical innovations in the field of computer graphics were being introduced, another pioneer, John Whitney Sr. was interested in the aesthetic possibilities of using a computer in creating pictures. In 1961, the first computer animated picture, *Catalog* was produced by John Whitney using an analog computer and his experimental efforts ended up with many pioneering short films. In 1966, a well known computer animated film, *Lapis* was produced by his brother, James Whitney and this six minutes long work had a unique atmosphere that reflected the combination of analog computer technology with meticulous craftsmanship [1]. Later, many advances in the field followed, and systems like CAMPER, SCANIMATE, CAESAR, ANIMAC began to show up [2]. These were tools to create action by animating lines and simple surfaces on the two dimensional screen of the then popular vector display computers. In its pure form, animation means, to give life, movement and character to an artificial creation. After the emergence of computers, the medium of traditional animation shared a unique relationship with this new and powerful tool.

Today we see this relationship in two ways: First is computer's assisting the traditional generation process of animation or *Computer Assisted Animation*. This assistance, though limited in some ways, still continues with advances occurring in the field of computing. Second role of the computer in the generation of an imitation of the real world is a totally new form of creation process. It is the so called 3-D animation, or *Computer Generated Animation*. These two processes are described in the second chapter.

In its most general sense, 3-D computer animation is a process of creating the real world in the computer realm. Hence the work on computer graphics deals with eliminating the problems related with the visual complexity of the world; by simulating the effects of light on the objects, shades, shadows and texturing of them for better illusions of realism. These results are getting better with the increasing power of computers. However, there still lies the problem of giving life to all these simulated objects, which is not yet explored thoroughly by computer scientists. The procedural complexity of the world is still an obstacle in front of realistic animation.

Every new day, we see tools for making animation to overcome this complexity and each of them displays a different methodology and varying levels of intricacy concerning both ease of use and design. Thisthesis is concerned with designing a student level animation generation package to be used in an educational environment. The primary issue considered in the design of the tool is its flexibility, that is, giving as much control as possible to the hands of the user. This however, must be done without compromising on the side of user friendliness, so that the environment of the tool must be easily accessible by the non-technical user.

The package, or system, is called **Puppet Theater** and is designed to be employed in educational three dimensional computer animation production on IBM AT™ and compatible computers enhanced with a Targa 16™ graphic board [3]. However, the system is transportable to any computer environment since it is written in the C programming language and hardware dependent parts are structurally separated. With appropriate screen and interactive input device drivers, it can be used on many different platforms.

Puppet Theater is a complete tool designed to be used in a school environment for generating three dimensional animation for entertainment, promotional or educational purposes. It consists of three integrated parts:

i - Puppets, a three dimensional modeling package that generates complex objects using polygonal representation.

ii - Puppeteer, a package to define smooth motion of Puppets as well as complex transformations and translations.

iii - Tailor, a rendering package complete with a shader and a previewer.

The details of Puppet Theater and its integral parts are explained in sections 3 and 4.

2. METHODS TO GENERATE ANIMATION ON COMPUTERS

In this chapter, the ways of generating animated pictures using computers and their differences will be explored. A general view of the present situation of three dimensional animation and tools used will be explored.

2.1 Computer Animation

Animation, as described in the introduction, is to give life to an artificial creation. This can be done using traditional methods as seen through the works of numerous studios all over the globe. This type of animation is exploiting a natural property of the human eye: the simulation of motion in the form of sequential frames of consecutive pictures generates an illusion of real motion when displayed with a rate quicker than 16 frames per second. Traditionally, these consecutive pictures are hand drawn on a transparent, back-lit table and considered successful if the illusion of motion is created on the viewers. The nature of this method is tedious as well as time consuming if not impossible for many at all. Only talented and experienced animators can generate motion through this very

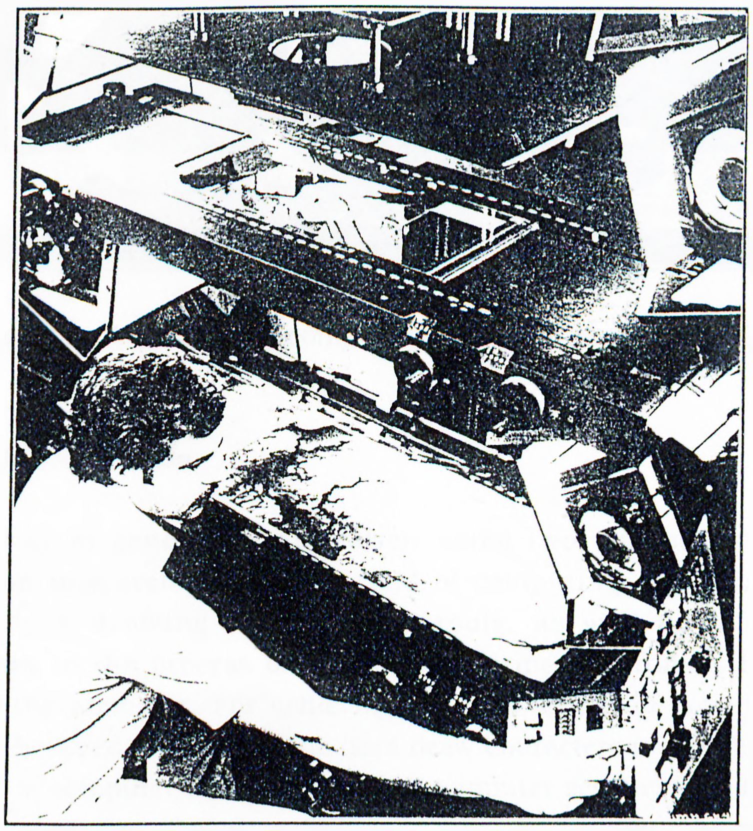

Figure 1 . A traditional animation production studio.

expensive process. However, this medium also proves to be effective when it comes to illustrate or entertain. In Figure 1, a traditional animation studio is shown [4].
The nature of the traditional process necessitates drawing of a character or an object on every frame of the animation. This totals up to at least 12 pictures for just one second of believable animation. Consequently, for a 10-minute hand drawn animation, 7200 pictures must be generated. With the introduction of the digital computer however, this process has been altered in a revolutionary way. Computer animation, as a term, refers to two types of processes:

i - Computer aided animation,
ii- Computer generated animation.

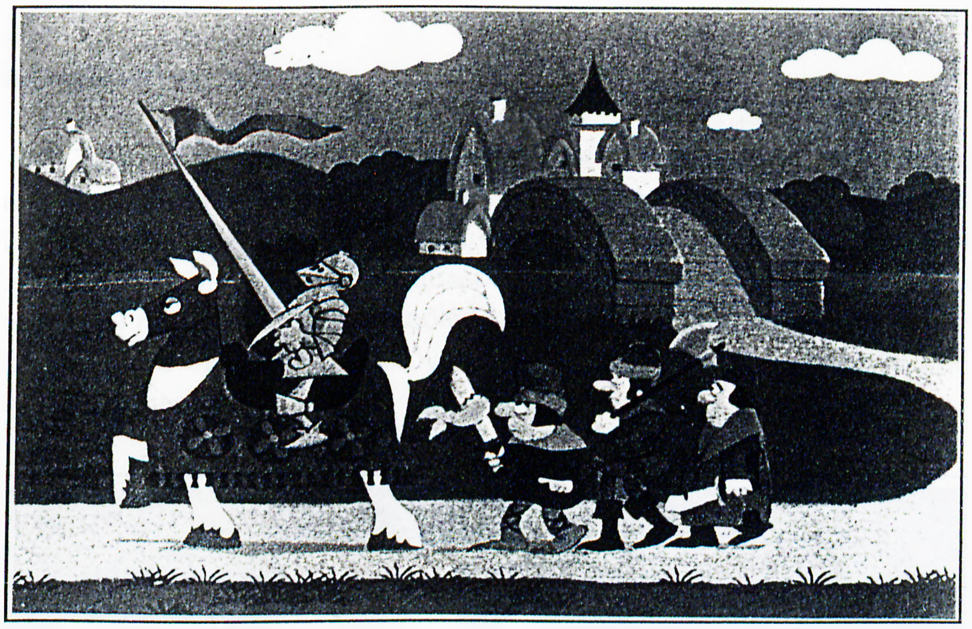
Figure 2 . A frame from a computer aided, two dimensional film.

2.2 Computer Aided Animation

The traditional way of generating animation using two dimensional drawings has benefited from the improvements in the field of computing. Systems for the use of animators employed drawing and painting tools, as well as inbetweening aid. Inbetweening refers to the process of filling the intermediate frames between two key frames drawn by the animator. For achieving different speeds of action, the number of pictures placed in between is varied. Animators draw characters (or rather actors, as used in the terminology of computer animation) on the computer screen using the drawing tools supplied and then paint them before recording on another medium, film or video. Figure 2 shows one frame from a two dimensional computer aided animation produced by Edwin Catmull. Depending on the capacity of the computer used, the frames may even be kept on the computer's own storage device for recalling back in real time, that is, close to speeds of 24 frames per second. Computers were expected to replace the animators at first, but problems had not allowed for a total relief. As described in [5], the problems of computer assisted animation are visible especially when the actors change their positions with respect to the point of view. While the computers prove to be assistive during the input of the drawings, coloring, composing, background painting, pencil testing and timing, they become ineffective when it comes to inbetweening. Several approaches are advised in the mentioned paper but the obvious difficulty of inferring a missing information in a drawing stands as an obstacle.

As a concluding remark, two dimensional, computer aided animation is not bringing a radical change in the process, but still proves to be an efficient way of decreasing production time and costs.

2.3 Computer Generated Animation

A totally new form of creativity appeared after the introduction of computer generated animation, since it utilized a concept belonging entirely to the new occurrence. Instead of regenerating an object's representation with each image, now a three dimensional, worldly representation of the object or actor in question is kept in the computer's data base. This three dimensional representation is handled by the agents of a digital computer and any change in our point of view is quickly reflected by a new rendering of the object's representation in question.

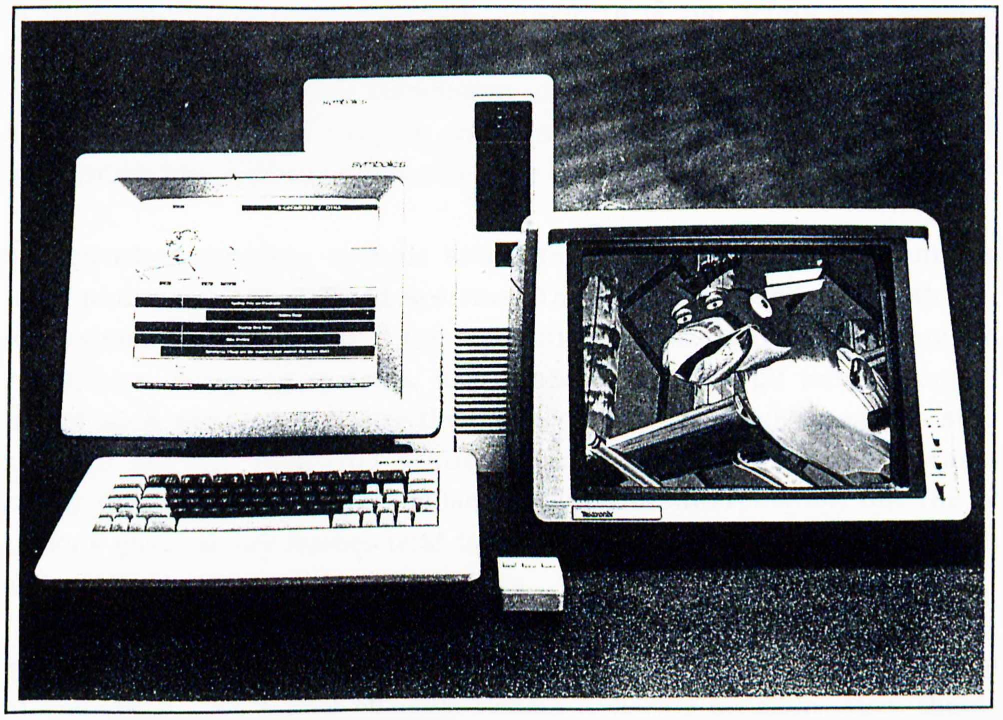

Figure 3. A system for generating computer animation

Now the viewer has became a part of this virtual representation and all visual references to the outer world are handled by means of formulations of the physical laws inside the computer's memory. Figure 3 shows such a computer, the Symbolics Paint and Animation system [29].

After representing an object inside the data base of the computer, it is possible to apply the physics of the world to this virtual entity. All physical qualities, light, reflections, refractions, colors are simulated in this micro cosmos. In addition to these, motion could also be simulated by altering the virtual three dimensional coordinates of the object with respect to an origin that can be handled as another object. The origin in question is the origin point of a three dimensional Cartesian coordinate system, that is the (0,0,0) point of a virtual world. While changing the virtual coordinates of the object, its worldly qualities such as color reflection, shades and shadows are recalculated for each new position.

All the above mentioned operations can be done in real time or stored in storage for a later preview of the combined action, depending on the power of the computer used. Representations can be animated following one of the two methods, either by defining the physical laws of motion as part of the system or supplying key figures and letting the computer do the inbetweening. They are called, physical motion definition and key frame animation respectively.

2.3.1 Physical Motion Definition

In this type of motion description, all physical laws governing the movements of bodies are integrated with the virtual representations. Movement of the actors are computed using laws of nature; gravitational constraints, frictions, tension between arms and the like. This kind of animation is open to automation and may be a big help to producers, since motion description may easily become an operation of ordering to move in a specified manner, very much like ordering a robot to move [6].

Physically described systems include simulated or model driven systems, scripting systems and parametrically defined systems. They are discussed in length in [7]. The simulated systems are best suited for scientific visualizations, a growing branch of computer graphics. Scripting systems, like ASAS [8] or MIRA [2] were among the first applications of such systems but later they were considered disadvantageous since it was not possible to see the resulting sequence until the whole program was written. Parametrically defined systems are similar to key frame interpolation but the nature of the instructions given at key frames tend to be complex and are sometimes insufficient to supply the positional information of the objects.

With the present state of the computer technology however, to integrate physical behavioral animation with a cost effective system is hardly possible. Also, there may be objections to a strictly physical description and an animation system may be considered incomplete without unworldly motion description. Therefore, in many texts a system employing both physical constraints and key framing is considered ideal. This last type of hybrid systems are not mature enough to be standard way of defining motions.

2.3.2 Key Frame Animation

In this type of motion definition, the central role belongs to the animator who supplies the key frames of the animation. His role is just like a choreographer's telling the dancers where and when to move. During the first years of computer animation, two dimensional animation systems used key framing extensively. In [9] the ways to produce two dimensional key framed animations are discussed. Recently, with advances in three dimensional applications, key framed animation when mentioned, almost always refers to the three dimensional case.

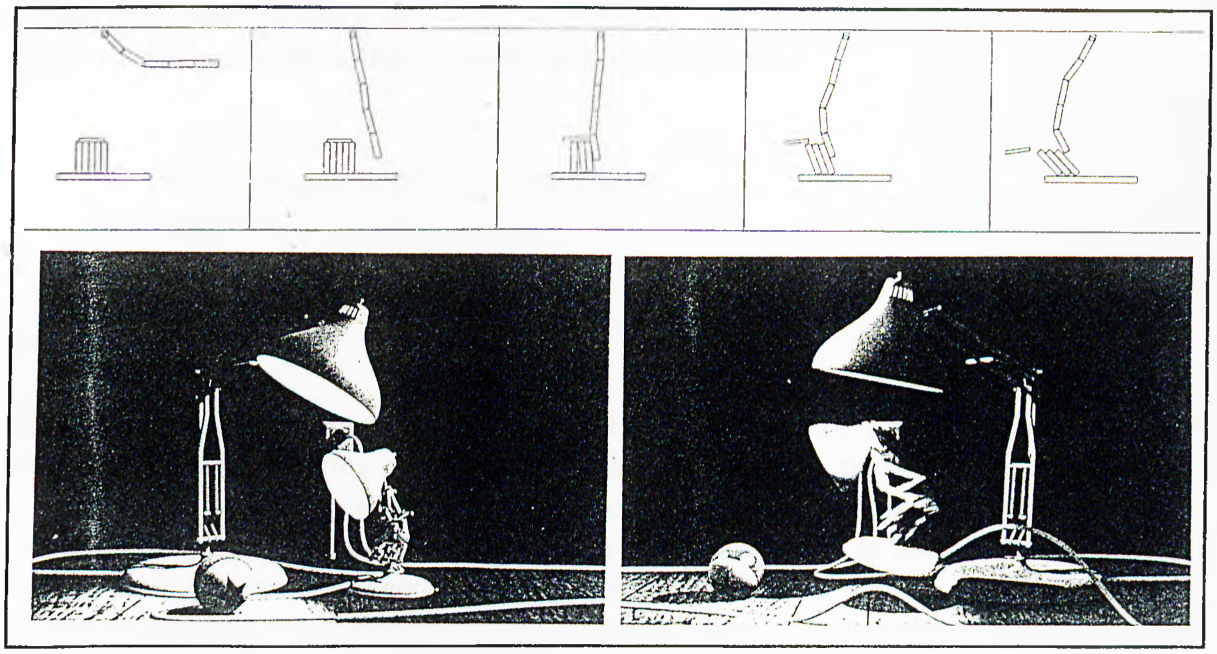

Figure 4 . Comparison of two approaches : physically defined vs. key framed.

In three dimensional animation systems with choreographic motion definitions, after the actors are generated they are manually animated by the directions of the animator. He gives the position of an actor with respect to a scene for the first frame of the movement piece and defines the course of its action using one of the many possible methods. The computer computes the missing frames of the actor's movement using the beginning and end values or incremental definitions and stores the scene description for future regeneration of the motion. These missing frames are called *inbetweens* after the traditional animation term, as described earlier.

Key frame systems also let the animator control the kinetic characteristics of the motion explicitly. The velocity and acceleration-deceleration of the objects can be controlled by changing the number of in-between frames without altering the key frames. However, as discussed in Lesseter's paper [10], just giving motion to rigid objects does not necessarily mean animating. Actually, traditional animation control methods like ease-in ease-outs can be simulated for better animation and the control of the kinetic characteristics implicitly gives such options to the hands of the computer animator. In the top part of Figure 4, a physically described movement of a chain and a set of blocks, made by David Baraff [11] is shown. For comparison, at the bottom part two frames from the key framed short film *Luxo Jr.*, animated by John Lesseter, shows a lamp jump.

2.4 Basics Of Three Dimensional Computer Animation

To develop three dimensional animation on the computer, there are three distinct operations to follow. First, the virtual representation of the objects must be generated in the computer data base. Then, these representations must be altered according to the desired motion description and at the end they must be brought to a worldly appearance by means of coloring, lighting etc.
Thus 3-D computer animation consists of three basic steps:

i - Object modeling,
ii - Motion specification or choreography,
iii- Rendering.

2.4.1 Object Modeling

Object modeling is the process of constructing objects to be handled afterwards. This construction is by entering the (x, y, z) coordinates of points that generate a skeleton, or wireframe representation of the actor (in Figure 16, wireframe representation of an actor modeled using Puppet Theater is depicted). A wireframe model is not adequate enough to give realistic appearance, therefore only serves as an initial step in generating a solid, colored or rendered object (for an example of a solid representation refer to Figure 5). When the animator wants to see the proposed move after some developments, this wireframe representation aids him. Modeling is done by surface descriptions, depending on the object shape: It is either by describing a set of polygons, by the equation of an algebraic surface or by surface patches [12].

2.4.2 Motion Specification

In three dimensional computer animation applications, the computer screen acts like a viewing window. This window can also be conceptualized as the objective of a movable camera. First the focus is on creating an environment, then the actors are treated by defining their three dimensional path within this environment.

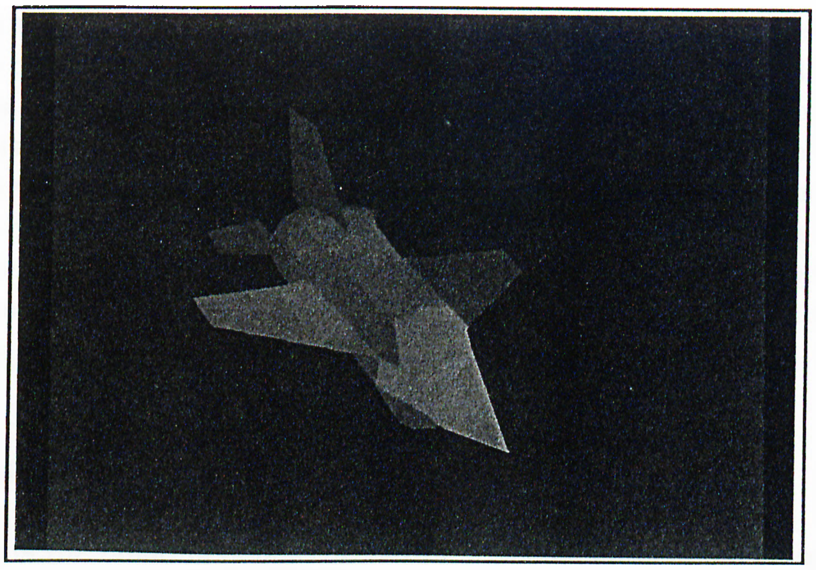

Figure 5 . An object modeled using Puppet Theater's sub-program Puppets

Controlling the motion of actors has been categorized in three levels: guiding, animator and task levels [13]. At the task level, the animation system must schedule the execution of motor programs to control characters. At the animator level, the systems are designed to allow the animator to specify motion algorithmically. Guiding level systems are those with no mechanisms for user-defined abstraction or adaptive motion. The efforts spent on this field are mainly due to produce an efficient system that combines three levels of complexion.

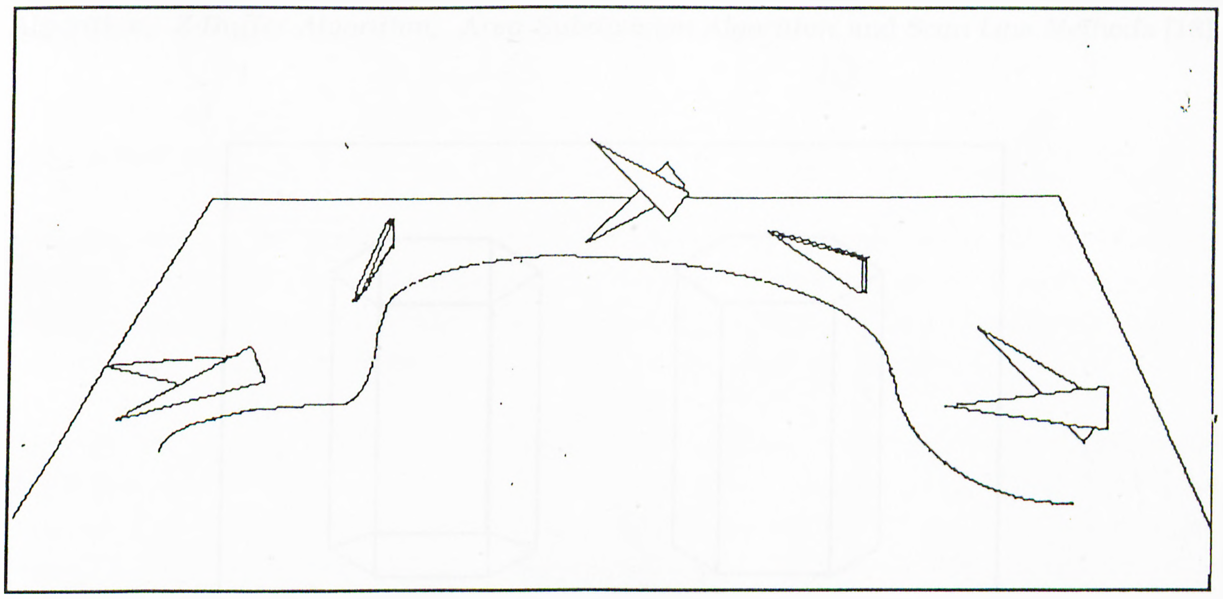

Figure 6 . An animator defined path of an object. Generated using Puppeteer .

Motion in traditional cartoon animation is often defined by squash, stretch, bending and changing of shapes [10]. In three dimensional animation, in addition to these actions, animators employ smooth curves and interpolating functions to simulate a behavior. A behavior is the total of all the animate acts assigned to any actor to be animated. These acts, stretches, squashs etc. are named by the Walt Disney studio and are now standard terms in the field. Technically, acts like stretching, squashing, bends and shape changes are obtained by transformation and deformation operations on the object. Smooth actions are obtained by employing splines [14], interpolations are handled by the iterative alterations of the object data base [15].

2.4.3 Rendering

Rendering is the tailoring of objects, having a planned and accepted motion. This process includes the methods to render a three dimensional object on the two dimensional screen space. These are, hidden surface elimination, shading and texturing, all of which can be included in a single rendering equation as in [16]. Two new techniques called ray tracing and radiosity give best results but the use of powerful machines is essential [17].

2.4.3.1 Hidden Surface Elimination

To render an object, one needs to project views of the object onto the screen as seen by a viewer. If the object in question is defined as surfaces filled with color or shading patterns, then hidden surface methods are employed to hide all back surfaces that are not visible from that particular point of view, as in Figure 7. These methods include *Depth Sorting Algorithm*, *Z-Buffer Algorithm*, *Area Subdivision Algorithm*, and *Scan Line Methods* [18].

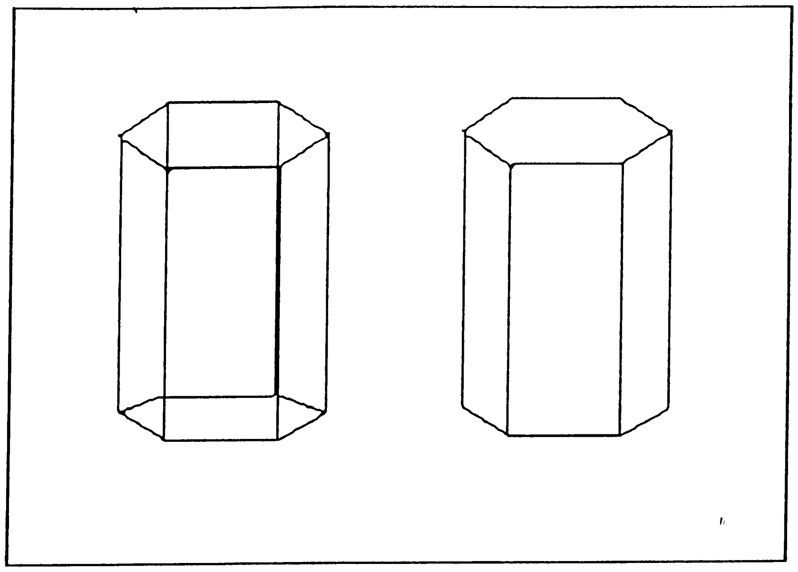

Figure 7 . Hidden surface elimination.

The Depth Sorting Algorithm was developed by Newell and approaches the problem first by sorting all polygons according to the largest z-coordinate of each [19]. Then, the algorithm resolves any ambiguities that may come up when the polygons' z-extents overlap. At the end, the algorithm reads each polygon in descending order of the largest z-coordinate and prints them in that order when finished.

In the Z-Buffer Method, first, z-coordinate values are stored for each pixel. The z-buffer, which is actually an array of real values, is initialized to the largest representable z value, while the refresh buffer, the array that contains actual display values, is initialized to the background pixel value.Then each polygon is scan-converted, that is scanned and processed into the refresh buffer, but without the initial sorting required of the depth-sorting algorithm. In recent graphics machines however, these operations have become a part of the hardware.

Area subdivision technique makes use of a 'divide and conquer' method: an area of projection plane is examined and checked if it is easy to decide which polygons are visible in the area. After this decision, the appropriate polygons or parts of them are displayed.

The scan-line methods operate in image space to create an image one scan line at a time. This method is a little more complex than the previous algorithms and involves the creation of an edge table for all non-horizontal edges of all polygons. Entries in this edge table are sorted into buckets based on each edge's smaller y-coordinate and within buckets based on the x-coordinate and an inverse slope.

The method used in Puppet Theater is a modified z-buffer algorithm. This method and the reasons for its preference are explained in the fourth chapter.

2.4.3.2 Shading

Shading refers to the operation of building a solid looking coverage on the objects that were defined by supplying three dimensional coordinates, by applying physical rules of nature. Shading an object is the last step of the rendering operation and may or may not include texture mapping, an advanced step in creating realistic looking scenes.

Different methods and ways of defining the light effects on the objects can dictate the shading methods. The simplest shading method is called constant shading (refer to Figure 8.) With it, the polygonal representation of the objects are illuminated with one or more virtual light sources and are assigned intensity levels which do not change within a polygonal surface. This makes the object look unworldly as the edges separating surfaces become obvious. However, for applications requiring fast renderings, and thus shades, this method is often satisfactory. This method may also prove to be reasonably realistic if the surfaces defining an object are very small.

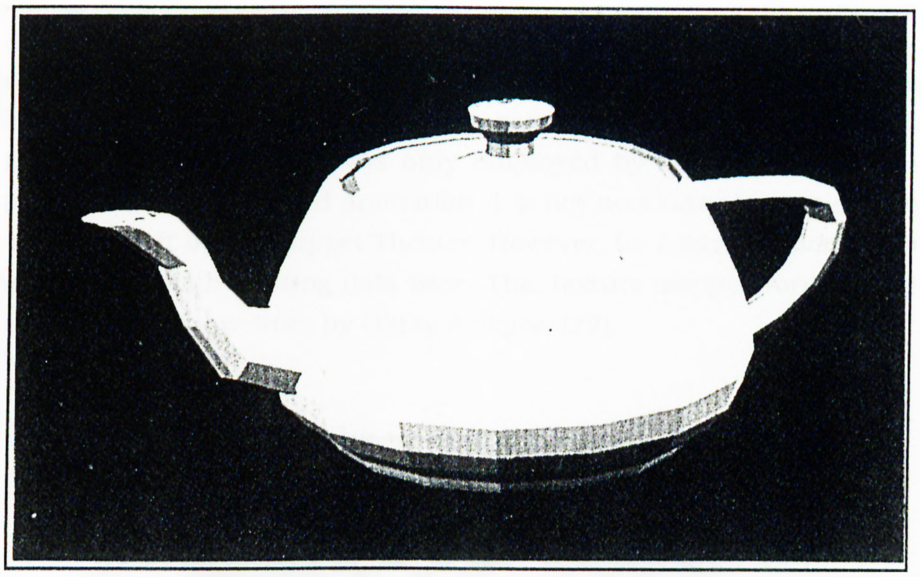

Figure 8 . A constant shading example

Advanced methods are named after their originators and called Gouraud and Phong shadings, with respect to increasing difficulty of rendering [20, 21]. In Figure 9, both methods are illustrated for comparison. Especially with Phong shading, it is possible to get more realistic looking scenes by varying the intensity levels within a surface by calculating the varying normal vectors within that surface. Here the information coming from the neighboring surfaces are taken into account and the effect of the light rays thus calculated. The overhead caused by this method is high and unapplicable for low-end applications, at least within the present level of efficiency of computers.

In this thesis, constant shading method is used. The method, as well as reasons for its selection are given in the fourth chapter.

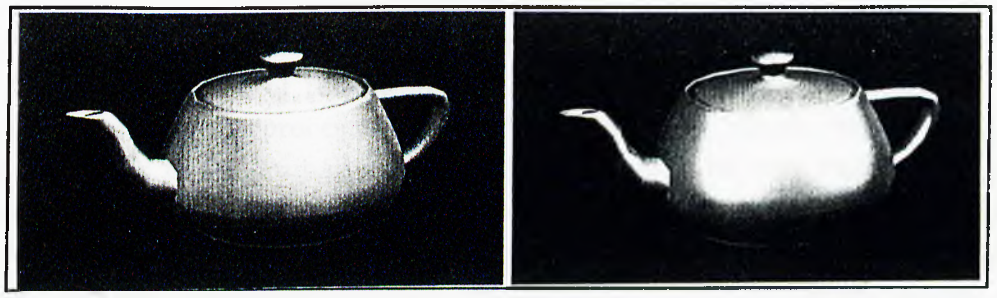

Figure 9 . Phong and Gouraud shaded teapots.

2.4.3.3 Texturing

Texturing is a further step of photorealism and is basically used to cover shaded objects with predefined textures. Texture mapping, as it is usually called, involves a long lasting operation and until very recently was only employed by systems for generating static images. In a system designed to aid animation it is not necessary to employ such a tedium and thus it has been left out in Puppet Theater. However, for future versions of our system it can easily be employed with existing data base. The texture mapped torus in Figure 10 was generated using a software written by Oktay Açıkgöz [22].

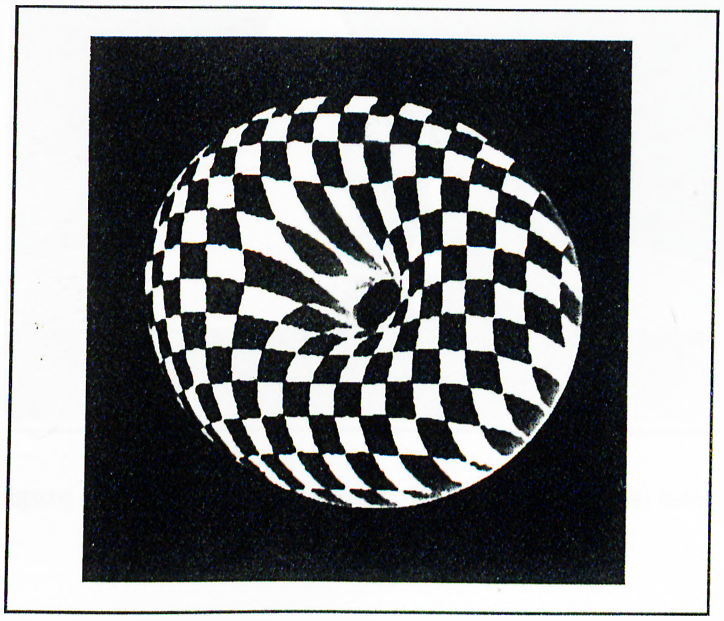

Figure 10 . A textured object.

2.4.3.4 Ray Tracing

This is one of the highest levels of computer achievement towards photorealism. In ray tracing, the natural laws of reflection and refraction are employed however, with an ingenious swapping of the process's parties. While in nature objects reflect or refract the incoming light rays to the human eye, the ray tracing method does the inverse: a ray for each point of the image space is traced back to a light source and the interfering objects are lighted. So, instead of tracing infinitely many rays emanating from created objects, only a few million rays are traced and checked. For all picture elements on the screen, a ray is sent inside the scenery and its interaction with the virtual objects is computed and the intensity value this picture element assumes is thus inferred according to the path of the ray. While highly realistic scenes are possible with ray tracing, this method is far beyond the present day capabilities of personal computers as computing the above takes tremendous computing time exceeding the speeds of a PC. A ray traced image created in

Bilkent University is shown in Figure 11 [17]. It took several minutes to compute this simple scene on a Sun computer.

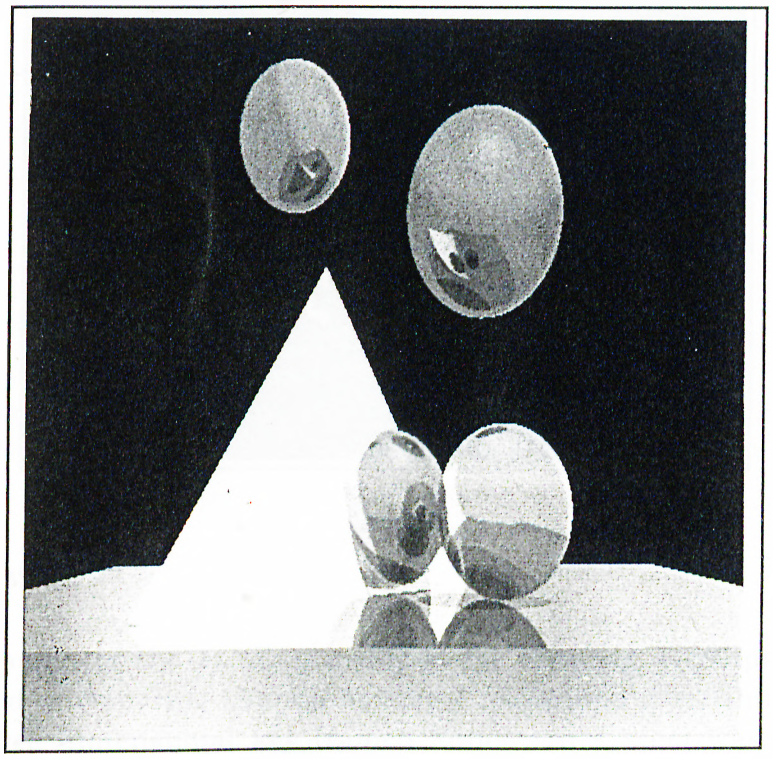

Figure 11 . A ray traced image . (Courtesy of Veysi Isler.)

3. PUPPET THEATER

After the discussion of the existing methods for creating a computer animation environment and considering the applicability of various algorithms to available hardware, a three dimensional key framed animation system was thought to be conceivable. It could integrate a modeling sub-program and employ a hierarchical object data base, that will allow users to define a motion either in terms of a single object, or by assembling two or more objects within a group.

A three dimensional approach to animation was favored because of its being a relatively untouched field of design and its unique position in challenging existing methods of expression. Currently, the generic exercise is using one of the available software tools for doing the task. These tools, albeit retaining a wide range of algorithms to create models and do calculations on them, lack the flexibility of a home-grown system. While confronting the user with high costs, (both time-wise and economically) results easily become self-replicating. The resulting imagery become dull and repetitive as inflexible and hard coded tools become outdated. The solution to these problems appears simply as a need to an in-house software open to all users who are capable of adding new software tools to existing ones. So the problem turned into an operation of selecting the viable methods of creating a three dimensional computer animation.

After considering and learning the existing methods, a polygon based, hierarchically defined object data base was selected. Objects are defined in three dimensions and their polygons are entered into the data base. Polygons, on the other hand are either three or four sided and data base keeps their vertex coordinates. Objects are said to be in a hierarchical definition, because they can be defined in terms of existing objects. That is, a set of corner points defining a polygon can be an object and one or more objects can form a group. Then groups can be treated just like objects. This type of a definition gives the user a flexibility of choreographing complex movements. The user can easily define a complex motion with multi levels of complexion. We must note that this kind of a multi level motion description was not available in the literature covered during this work. A hierarchical motion definition is probably kept away from popular literature for reasons of its market considerations.

For creating an environment for defining three dimensional movements, it soon became obvious that a windowing environment was necessary for the platform we were to generate. Usually high and medium level computer systems employ a built-in windowing environment and desired graphical primitives are readily available within that environment. Graphical primitives are the basic elements of graphical user interfaces, lines, boxes and pull-down menus are some examples. The fact that, our system, namely the Targa 16™, lacked such an environment confronted us in the beginning. An already existing operating environment [23] was considered and substantially redesigned for existing hardware and a unique windowing system was created. This system is called WODNIW (Wonderful Objects Developer with Nearly Interactive Windows) by its designers.

3.1 WODNIW

The windowing system our hardware employs, is an original one created to fulfill the needs of the tool designer. The principles of this windowing system are not very different than of the commercially existing windowing platforms. WODNIW is an event-driven windowing environment in which user can open windows, menus, filled or empty boxes, buttons and other user interface tools while tracking the mouse's motion in an event driven fashion [24]. A menu operation, for example is defined once in the program and WODNIW tracks the events every time an operation is performed. Since this menu operation is not controlled in every part of the software, the code is compact and readable. An environment with this kind of operation tracking is called an event driven environment [30].

This specific windowing environment lets the programmer define events and define them just once in the beginning of the programs. An event may be one for tracking the motion of the mouse or pen, or it can be a series of instructions for detecting whether the mouse button is pushed inside an icon; a graphical user interface element. Then during an application session, all the events defined at the beginning are controlled continuously by WODNIW. This schema is called "event driven operation" and is very helpful to the designers of software. Obviously, the non-technical user will not see or feel any such kind of programming particularities.

The details of WODNIW and programming guidelines are supplied in the Appendix.

In the following section, the key elements and considerations of Puppet Theater and the reasons for their preference are explained.

3.2 Reasons For Selecting A 3-D Environment

Some of the reasons behind our preference of a three dimensional animation software were given above. These included the three dimensional methods' power in describing an event within a scenery and their techniques' being radically new when compared to existing ones. Also, for creating educational and entertaining films, this field has proven to be more effective than two dimensional media. Creating the replica of a three dimensional world, complete with object relations and movements frees the designer from thinking about relative perspectives, lighting and similar tedium. He can concentrate on telling his story whether it be an educational, entertaining or a commercial one.

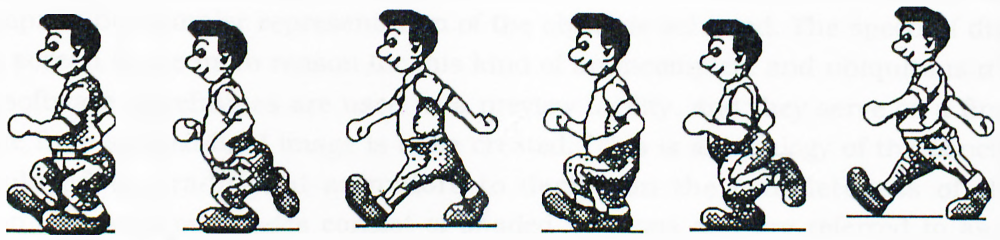

Figure 12. A two dimensional walk sequence created by VideoWorks™ .

There exist two-dimensional tools for creating animated films, like AutoDesk's Animator™ for PC's, or Macro Mind's Video Works™ for the Macintosh computers (Figure 12.) They help users to define two dimensional objects and backgrounds before letting them move these objects on the screen. Later, the user can playback what he has done and interactively change any part of the animation. To write a three dimensional software similar to these 'two dimensional popular software was the starting point of Puppet Theater. Like its two dimensional counterparts, Puppet Theater lets the user define his three dimensional objects and scenery. Then the user can animate these objects with supplied animating tools and can observe the result within minutes. If the results are satisfactory, he can integrate these actions with others on the same screen and keep their definitions for a later recording.

Puppet Theater is a three dimensional environment but the screen to view results and tools to interact with software are strictly two dimensional. To overcome this difficulty, special design considerations were thought and applied. While creating a three dimensional object to be animated, two approaches were used. First is a straightforward and somewhat low-level approach: to enter three dimensional coordinates of the corner (vertex) points of all polygons used. This approach, although tedious for novice and not error free, is actually a very interactive and aiding method for low-level user. Non symmetrical and highly complex objects can be created by this method, and in other contexts too, this basic approach was employed; modelers and animators measure the actual object they want to model and store the measured data inside computer's data base. The second approach to create three dimensional objects is by using the symmetrical nature of them : users trace a line on the two dimensional screen and the computer software rotates it to generate a three dimensional object data base. This object, after being created is subject to other operations supplied by the tool. By this method one can overcome the difficulty arising from the two dimensional nature of the inputting devices. The method of sculpting three dimensional objects by rotating around a fixed axis has been employed by many existing packages and described in several sources [7,25]. For such a task, it was decided to depend on local sources and take an ongoing project in our university as our reference [26].

3.3 Representations of Objects

A very popular representation of three dimensional objects is a wireframe model. In this model, the object is displayed as a set of straight lines connecting the vertex points. Thus, an incomplete but simpler representation of the object is achieved. The speed of displaying them on screen is the main reason for this kind of an incomplete and ubiquitous model. In many a software, wireframes are used as a preview facility, and they serve as a first check before the fully shaded final image is to be created. (This is an analogy of the pencil or line test employed by traditional animators to decide on the completeness of the final animation.) These final images consist of shaded polygons and are referred to as surface representations.

The first reason to use a polygonal representation is its wide applicability. Almost all of the existing packages adhere to the polygon approach as opposed to constructive solid geometry representations. A set of three dimensional primitives, that is, basic objects like blocks, cylinders, pyramids etc. are provided to the user and by using logical operations defined for three dimensional space, new solids are generated. The results are called constructive solid geometry representations [25]. Time has proven that polygonal representations are better adjustable and more flexible when it comes to animation, even though some constraint based motion simulators prefer constructive solid geometry [27]. After selecting the polygonal representation, it is the designer's problem to decide on how these polygons are kept in the graphical data base. Most widely used implementation is to keep the vertex data for each polygon and then refer to them using pointers.

3.4 Hierarchical Object Definition

A popular definition of objects is a hierarchical definition. In this type of complexion, all elements of objects are linked to each other in a hierarchical manner. One way to represent a polygonal structure is to define all surfaces into a list of polygons and each polygon into a list of vertices using pointers. Therefore each vertex data should be stored once and will be used only by means of pointers. Although the usual practice is to define objects by surfaces and surfaces by polygons, we adhered to a simpler version of this representation.

In our implementation, objects are composed of four sided polygons and the basic data stored in the data base is the data of vertex points. Then each object is made up of a list of polygons. A further hierarchy applied is of the object order: objects are grouped or glued together to form compound objects or *comps*. In our data base, simpler objects forming groups are called *atoms*. Once grouped together, all operations applicable to objects also work for comps. The atoms that have been used in a comp can be used as part of another comp or may be a standalone object. This grouping is very helpful for choreographing complex motions, since the hierarchical structure of objects is reflected out as hierarchical motion. More about the hierarchical definition is in section 4.1.

3.5 B-Splines and Their Use in Animation

B-Splines are curves in space that are defined by a set of parametric equations. (Technically, B-Splines are piecewise cubic polynomials defined by a set of parametric equations). The curve in Figure 13 is a cubic B-Spline curve with control points as shown. Parametric representation of cubic curves is an explicit way of defining points in space and is an inferior approach next to using quadratics, a more complex mathematical representation. The computer graphics field prefers cubics simply because cubics are computationally feasible and easier to handle. The points on a curve can be computed sequentially, rather than by solving non-linear equations for each point in an implicit definition. Besides that, parametric curves are easily transformable, since the transforming function is applied to the control points and not implicitly on the curve.

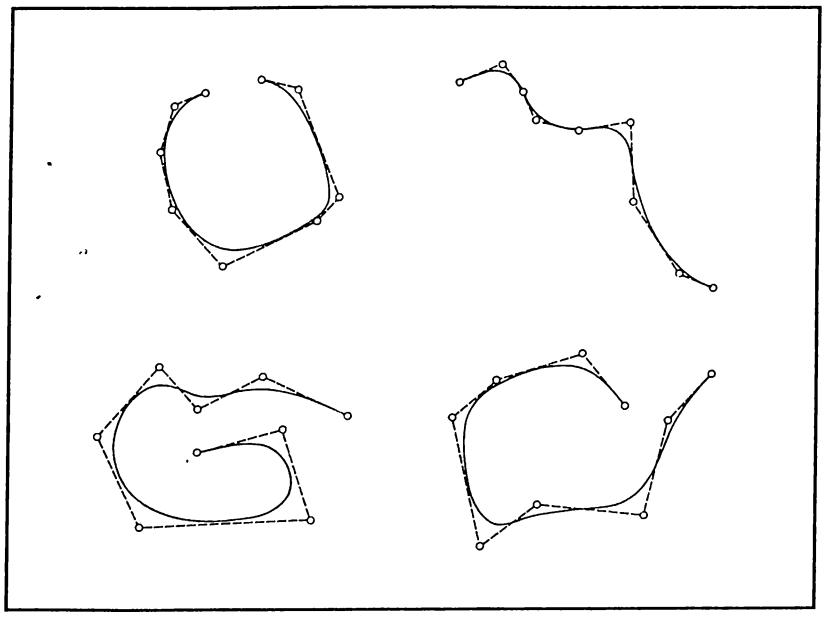

Figure 13. Cubic B-Spline curves with control points.

B-Splines are used in computer animations for achieving smooth motions of objects through a curve. As the points which form the B-Spline are generated, they form as a track for the objects to be animated. A further help of B-Splines is their property of supplying support on the accelerations and decelerations. If the control points around a necking region is stressed, then the object, while following the curve points will slow down and make a believable pass through the neck.

3.6 Rendering for Previewing.

After all descriptions for movements are given, the objects will be rendered according to the shading model used. Puppet Theater uses constant shading for preview, since it is mainly an animation tool. This automatically solves the hidden surface problem, because this level is implicitly included in the rendering equation. The method used for constant shading is a *scan line Z-buffer* algorithm. This method is used in almost all computer graphics applications regardless of the cost of equipment.

A Z-buffer is an array structure used to store sequential data. In advanced computer graphics systems, there is a special screen buffer next to the frame buffer for keeping the z-coordinate values of objects at each screen point. Our implementation necessitates keeping such a buffer within software. Puppet Theater's Tailor sub-program optimizes a scan-line z-buffer algorithm. In this approach, the redundancy of keeping memory locations for each point on the screen is eliminated by scanning a line from top to bottom of the effective display area. While scanning the line, a z-buffer stores the real values of z-coordinates and displays the closest. When the scan line reaches the end, it jumps to the next line and re-initializes the z-buffer and replaces the old contents with the new values of the second scan line.

The advantage of keeping only one scan line is efficient for our purposes, but to re-initialize the contents every scan line may be disadvantageous since this means not being able to use values in the future. So the ideal case is to use the Z-buffer algorithm, keeping all screen values at a time. The latter is not suitable for PC systems with insufficient memory.

4. A CLOSER LOOK AT PUPPET THEATER

4.1 Details of the Data Base

The objects in Puppet Theater are kept with their polygon data and color identity. The polygons are four sided and this is the first item in their data structure of type integer. Next comes the color attribute. The hardware running Puppet Theater is a 16 bit frame buffer and 15 of them are reserved for color. In an RGB (stands for Red, Green, Blue) system, the color intensities are generated by changing the percentage of these three basic components. Thus for example, 100 % Red + 0 % Green + 100 % Blue give a full magenta. Zero values for all of the three gives black and full percentage of all gives white. Each color is represented with 32 shades of the respective color, therefore there can be 32x32x32 = 32767 colors displayed simultaneously. Each color in our system has a value between 0 and 32767, 0 being black and 32767 being pure white. The integer variable **color** is a number showing the color of that specific polygon.

Next are the double floating point values of the polygon.

```
typedef struct polygon
    {
    int type;                  /* either 3 or 4 sided polygons*/
    int color;
    double x[4],y[4],z[4];    /* points for polygon*/
    } POLYGON;
```

Previously created objects or puppets are then transferred into an object record that keeps a linked list of sub objects with pointers to their respective polygons. Thus the system relies on sub objects (atoms) and super objects which are compounds of these atoms. The atoms are connected to other atoms or compounds via a hierarchical, object oriented structure. Thus any kind of movement action or a translation can be defined for either an atomic object or a compound super object. This simplifies the matter, as for example a complex motion, like walking and cutting scissors, that moves on a table (Figure 6) can be modeled and animated easily.

The combined data structure details are below:

```
typedef struct polylist
    (
    POLYGON poly;
    struct polylist *next;          /* pointer to next polygon */
    )POLYLIST;

  typedef struct atom
    (
    POLYLIST *plist;
    int x, y, z;                    /* coordinates of the representing pt.*/
    double alpha, beta, gamma;      /* rotation angles */
    )ATOM;

  typedef struct comp
    (
    int objectid;
    struct comp *next;              /* pointer to next comp */
    )COMPOSITE;
```

This structure is illustrated in Figure 14, and movement examples are in the figures of Section 4.3 and recorded in the accompanying video tape.

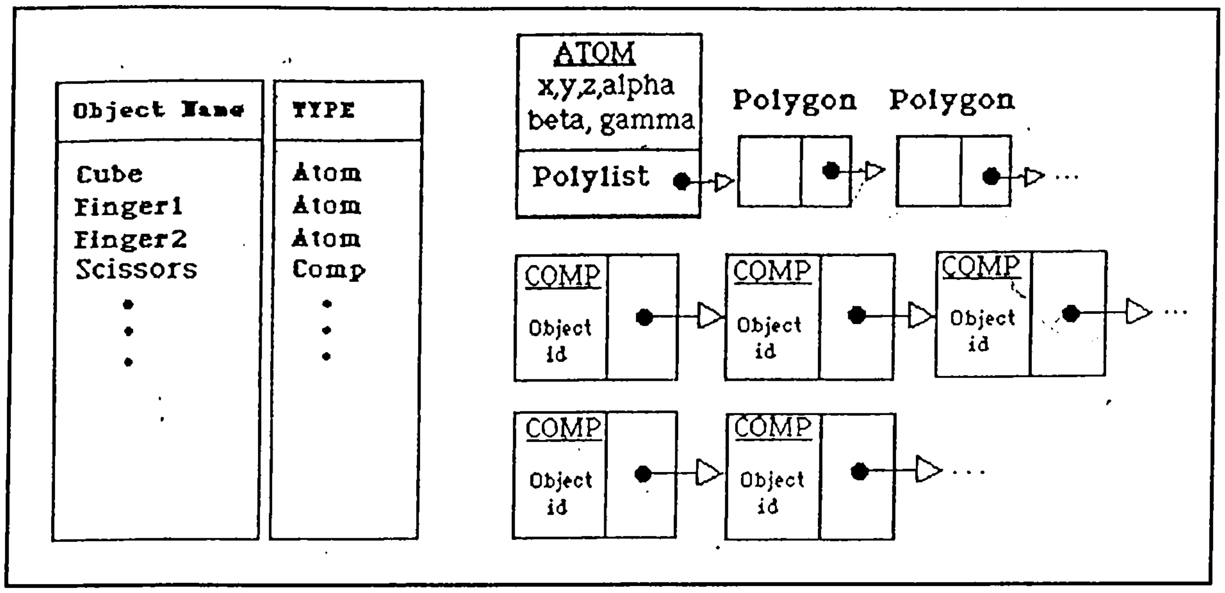

Figure 14 . Structuring of the hierarchical data base.

4.2 Creation of a Puppet

In Puppet Theater, creation of puppets can be handled in two ways. First is to digitize the approximate measurements of an object and second, using volume sweeping. Digitizing the actual data of objects is a standard application usually forced by the limitations of present day input devices. The users measure the dimensions of objects to be represented and create a three dimensional mesh of points. These points serve as vertices of the polygonal surfaces. For curved objects, a polygonal approximation of the surface is entered and shaded with advanced rendering methods like Phong's. The surface is better simulated. If the object to be animated is a fictitious one that has no solid world representative , then the data of the object will be a mathematical abstraction, like a fractal mountain for example [28].

Another very efficient way of representing objects is *volume sweeping.* Volume sweeping refers to the operation of sweeping a curve or a volume around or on a curve to generate three dimensional values. Puppets sub-program uses a curve sweeping algorithm. It can vary the complexity of the object by varying the number of polygons generated after the sweep. An example object generated using this method is shown in Figure 15 and the details of producing an object is shown in the accompanying video tape.

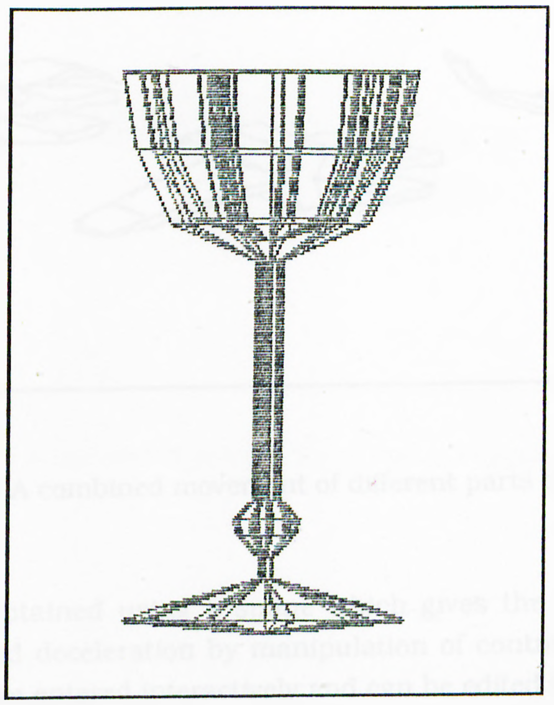

Figure 15 . An object created by the sweeping operation performed by Puppets.

The algorithm is very straightforward . First, a number of points are entered via a mouse or pen relative to a straight line, serving as the axis of rotation. Then these points are connected with lines and the resulting line is rotated around that axis. The complexity is variable since the user can interactively change the number of points of both operations. This approach was based on [26].

4.3 Animation of Puppets

The sub-package called Puppeteer is responsible for the movements of puppets. Here, as described in the data base section, puppets can be either atoms or grouped atoms, comps. A puppet, whether a comp or atom, performs translations, scalings and rotations. After having an initial position in the scene, the object can be rotated in space. The rotation angles are given incrementally and for each frame in the sequence, Puppet Theater computes every action resulting from this increment. Simultaneously, the atom or comp can be scaled during the sequence and the result is a complicated move. There are 24 frames for previewing, and they can be concatenated for longer sequences. In this sequence, the incremental values may be kept going or changed in between. Therefore a total control of the action can be achieved. Later, this combined transformation is performed with the spline trace.

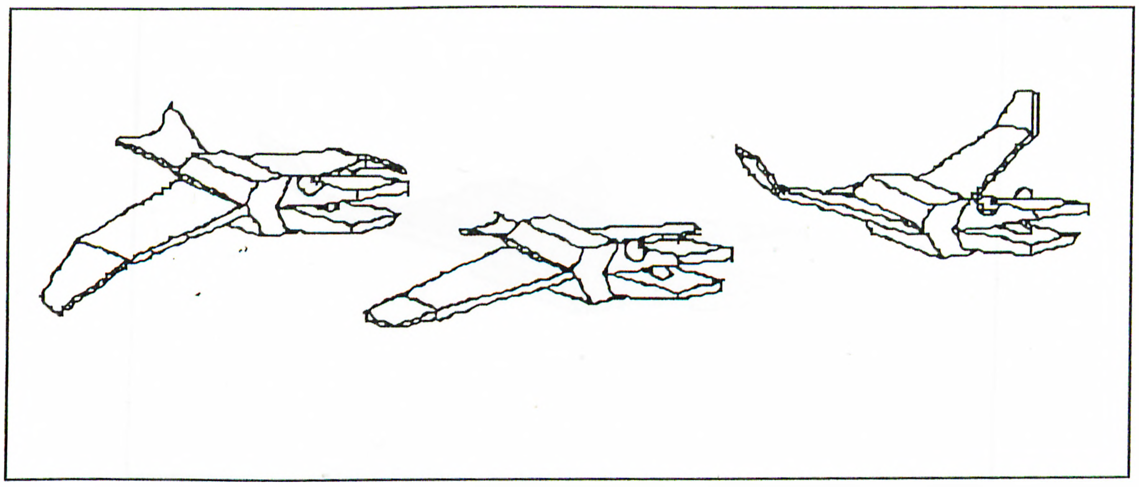

Figure 16 . A combined movement of different parts of a puppet.

Smooth movement is obtained using a spline which gives the user an extra ability of defining acceleration and deceleration by manipulation of control points as described in [14]. The spline points are entered interactively and can be edited for new splines. Then the object in question is swept on the spline curve. This operation includes the combined transformation action of the puppet, therefore, the puppet performs more than one movement simultaneously.

4.4 Tailoring

This rendering sub package is developed for the tailoring and shading operations. Complete scene descriptions for each frame is shaded, utilizing a z-buffer constant shading algorithm and taking the place of light source(s) into account. [Camera view is computed to give the impression of active scene.] In the future this shading function will be improved by replacing it with Gouraud or Phong shading options but this will include an overhead, as the system in question is a low cost station (especially time wise). This part of the system controls the driving of a connected video recorder if automatic controllers are available. Hence at this last stage an option of including video effects, like fade-ins fade-outs or page turn effects or matte operations that could composite computer input with real video recordings is possible. All these effects can be programmed externally, since all of them are external raster operations and have no connection with internal structuring of objects or movements. Figure 17 shows a tailored object "bird" with shades calculated according to a light source.

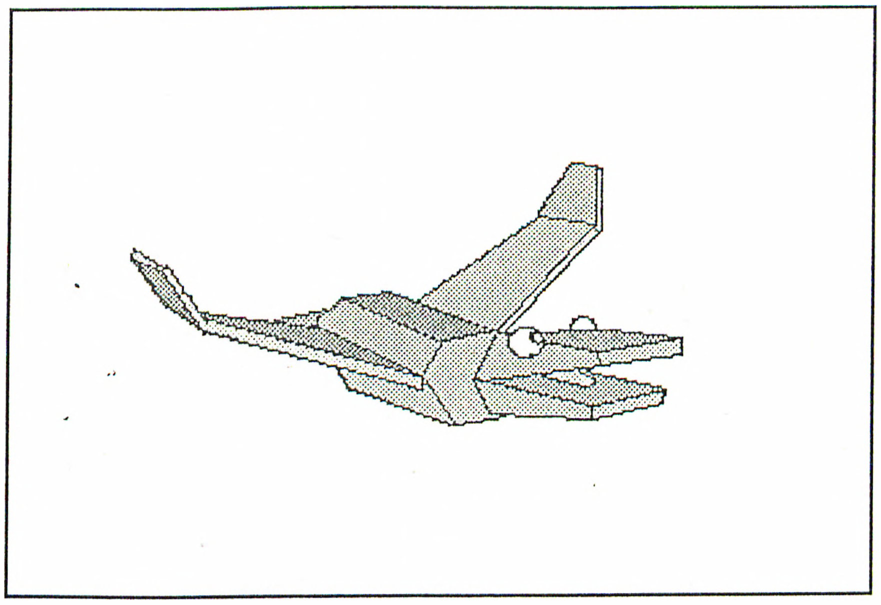

Figure 17 . A tailored object produced using Puppet Theater.

4.5 Previewing

Although wireframe is the most popular representation its speed and simplicity it gives, in our case it only suffices to display images with a certain speed that is not enough to use for action tests. The hardware limits us to less than 16 frames per second while displaying. For the same reason, objects with a greater number of vertex points are displayed at slower speeds. The solution to this problem came up as keeping a small portion of the screen fully as a raster image and then display 24 of them in a cyclic manner. The number 24 comes as a result of memory limits, it is the number of 100x100x16 bit images that fit in our memory.

When animator finishes a scene description with respective objects, light sources, camera and their movements, he can preview a portion of the action at a standard rate of 24 frames per second or at higher or lower rates. This helps him in giving an idea of the finished product and supplies feedback. If the action is too slow at some parts of the track, he can rearrange it playing with the controlling points on the spline.

5. CONCLUSION

The reason behind this work lies in our exigence for creating animated pictures using the existing hardware within the premises of the Faculty of Art, Design and Architecture, and making the tools to create animations accessible to fellow students. Our faculty owns a computer graphics laboratory with computers having capable graphics cards. One of them, the Targa 16™, is an established standard in American art schools. This graphics hardware can simultaneously display many colors on a large color monitor.

For this set up, a tool for preparing animated sequences was conceived possible and a universal programming language, C, was selected as the programming environment. After the initial preparation, it was understood that there existed no tools for even displaying a single colored pixel on the screen. Thus, the first part of the thesis necessarily turned out to solving the problem of graphical primitives, as they are called in the jargon. Consequently, first some programming tools for writing an animation software was designed. These included routines (programs) to define a line, then a rectangle, then a filled rectangle, etc. At the end of the first part, we ended up with a windowing environment, which is a derivation of the ideas gained while working with the SunVIEW™ windowing environment.This software, which serves as a backbone to other programs was called WODNIW. Then, for the core part of the task, existing animation packages and technical papers were sought. Upon our surprise, many important details of writing such a tool were left unexplained in almost all of the sources. This is due to a market condition: Animation packages, regardless of their size, costs tens of thousands of dollars. Therefore, it was not easy to find aids to proceed with writing an animation software. However, we think the hardware we own was used efficiently and this job proved very rewarding and educational, as many sub-disciplines of computer graphics have been surveyed for a self sufficient tool.

This tool, Puppet Theater, may be an initial step of a bigger project: computer graphics research at Bilkent University is fruitful and different subjects are explored. All of these projects may be embedded in a single tool and become a complete creativity tool, with texture mapping, object deformations, ray tracing and radiosity. I think this work may help combine such research into one package.

APPENDIX

WODNIW is composed of subroutines to handle basic events, like mouse, text, window etc. To write programs using WODNIW is straightforward once the programmer understands the event driven style of the software.

All screen attributes are handled via a rectangle definition, therefore it is best to start with defining base rectangles.

1- Rect Handling:

These structures and routines are used to handle rectangular drawing areas. The data type used is **Rect**. It has the components x,y,width and height.

Create a rect by	: rect_identifier = rect (x, y, width, height);
Save a rect to disk	: rect_save_disk (file_handle, Rect);
Load a rect from disk	: rect_load_disk (file_handle, Rect);
Save rect to memory	: rect_save_mem (char *mem, Rect);
Load rect from memory	: rect_load_mem (char *mem, Rect);

Note that there is no allocation of memory.

Invert all bits in a rect on the screen (frequently used for signaling a massage to the user) : rect_invert(Rect);

For checking whether the mouse is inside a region on the screen, use : mouse_inrect(Rect) returns 0 or 1 depending on the position of mouse. 1 means the mouse is inside the rect.

2- Window Handling:

Window handling is done by opening a main window first. An id number (or name) is given to the main window and it is used for later reference. The other routines give (return) various information about the window.

```
winid=win_open( rect ( x, y , width, height ) , storage_type , color_of _frame, color _
        of _canvas ) ;
win_close (winid ) ;
win_clear (winid ) ;
win_get_x (winid ) ;          /* returns top left x coordinate of window */
win_get_y (winid ) ;          /* returns top left y coordinate of window */
win_get_width (winid ) ;      /* returns width of window */
win_get_height ( winid ) ;    /* returns height of window */
```

* storage _type is ST_MEM for windows to be opened in memory
  is ST_DISK for windows to be opened on disk
  is ST_NOSTORAGE if window is not to be closed
* winid is of type Window
* color_of _frame and color_of_canvas are integer values constructed as
  blue+32*green+1024*red
  0 <= red, green, blue <32

3- Mouse Handling:

For using a mouse, the command mouse_on() is used before all mouse operations. When the tip of the mouse is not to be seen, arrow_off() is used.

Initialization	: mouse_on ( ) ;
Closing mouse handler	: mouse_off ( ) ;
Hiding mouse pointer	: arrow_off ( ) ;
Recover mouse pointer after hide operation	: arrow_on ( ) ;

The mouse_parameters are automatically updated by an interrupt handler that sets

mouse_click	: Boolean (char)
mouse_release	: Boolean (char)
mouse_x	: integer
mouse_y	: integer

4- Drawing Routines:

For drawing pixels on the screen, the most basic command is pixput. Using it, one can draw a single pixel on the (x,y) coordinate of screen with variable "color", that can take values between 0 (black) and 32767 (white). Pixget reads the color from the given point and pixline draws a line between two points. The last two operators are for window and menu handling or other operations such as button plotting.

```
pixput ( x, y, color ) ;                 /* Draw single pixel */
pixget ( x, y ) ;                        /*  Returns color of pixel  */
pixline ( x1, y1, x2, y2, color ) ;      /*  Draws line between two points  */
pixbox (  Rect, color ) ;                /*  Draws filled box  */
pixrect (  Rect, color ) ;               /*  Draws rectangle  */
```

5-Text Handling:

WODNIW uses a proprietary font format and can use many different characters. There is a system font defined by the programmer and named SYS.FON. The first two operators work with the system font whereas last two use a new font specified with text_setfont.

```
To write  a string with the system font use
              gprintf ( x, y, string, color ) ;
To write a single character use
              gprintchar ( x, y, cher, color ) ;
To use aspecial font file first load font by
              text_setfont  ( "font file name" ) ;
then use it with the two routines
              text_putstr ( x, y, string, color ) ;  and
              text_putchar ( x, y, char, color ) ;
```

6-Menu Handling:

To create a menu, define a menu variable of type 'Menu', then initialize the structure by

```
        menu_create ( &mymenu ) ;
To add items, use:
        menu_create_item ( &mymenu, "ITEM 1", submenu_name ) ;
        menu_create_item ( &mymenu, "ITEM 2", submenu_name ) ;
        .
        .
        menu_create_item ( &mymenu, "LAST ITEM", submenu_name ) ;
```

To invoke (that is display and get selection) a menu, use

```
menu_invoke(&mymenu, x, y) ;
```

*x, y is the top left position on the menu box

*This procedure returns an integer that represents the item number of the selection

*If menu was closed by Esc or mouse was clicked outside the menu box, return value is 0

After using a menu one can dispose the occupied memory by

```
menu_destroy (&mymenu);
```

7- Event Handling:

This is the most outer shell of the windowing system. All operations defined until now work as an event, so they must be defined to be operated. Two types of events are defined: menu event and keyboard event.

For processing the mouse events, an event list is used. To add an event to the event list, use:

```
event_defm(Window, Rect, &flag_variable);
event_defk(int key value, &flag_variable);
```

This directs the system to set the flag variable if the mouse is clicked in the relative region given on the specified window. For example, if a Boolean flag exit_flag is defined and if we want to set it when the top left portion of the window is selected (closebox) we write: event_def (window_id, rect (0,0,16,16), & exit_flag);

To wait until an event arrives, use wait_for_event
So a template event loop is as follows:

```
        while (!exit_flag)
        (
wait_for_event ();
if (event1_flag)...
if (event2_flag)...
if (event3_flag)...
if (keyvalx32)=`a`&& keyval x 32 <= `z`)...
```

References

1 Jankel A, Morton R ., *Creative Computer Graphics* , New York:Cambridge University Press (1984).

2 Magnenat-Thalmann N. and Thalmann D., *Principles of Computer Animation.* Tokyo: Springer (1985).

3 Targa 16 Reference Manual, Truevision Inc (1986).

4 Paterson R., Animation techniques for the secret of NIMH, *American Cinematographer,* (1982), 808-12.

5 Catmull E. The Problems of Computer-Assisted Animation, *Tutorial:Computer Graphics,* IEEE Computer Society, (1982), 495-500.

6 Miller G., The Motion Dynamics of Snakes and Worms, *Computer Graphics,* **22** (4) (1988), 347-358.

7 Watt A.,*Three-Dimensional Computer Graphics.* Avon: Addison-Wesley,(1989).

8 Reynolds C.W. Computer animation with scripts and actors. *Computer Graphics,* **16** (3), (1982), 289-96.

9 Steketee S. N. and Badler N. I., Parametric keyframe interpolation incorporating kinetic adjustment and phrasing control. *Computer Graphics,* **19** (3), (1985), 255-62.

10 Lesseter J., Principles of Traditional Animation Applied to 3D Computer Animation. *Computer Graphics,* **21** (4), (1987), 35-44.

11 Baraff D., Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Computer Graphics,* **23** (3), (1989), 223-32.

12 Newman M. E. and Sproull R. F., *Principles of Interactive Computer Graphics.* New York: McGraw-Hill, (1981).

13 Zeltzer D., Towards an Integrated View of 3-D Computer Animation. *Computer Generated Images, edited by Magnenat-Thalmann and Thalmann.* Tokyo: Springer, (1985), 230-248.

14 Kochanek D:H.U. and Bartels R. H., Interpolating Splines with Local Tension, Continuity, and Bias Control. *Computer Graphics,* **18** (3), (1984), 33-41.

15 Chadwick J. E .et al., Layered Construction for Deformable Animated Characters. *Computer Graphics,* **23** (3), (1989), 243-252.

16 Kajiya Y. J., The rendering equation. *Computer Graphics,* **20** (4), (1986), 143-150.

17 Isler V. and Özgüç B., Ray tracing geometric models. *4th International Symposium on Computers and Information Sciences,* METU,(1989), 493-503.

18 Foley J.D. and Van Dam A., *Fundamentals of Interactive Computer Graphics.* Reading MA: Addison-Wesley, (1982).

19 Gharachorloo N., Gupta S., Sproull R.F., Sutherland I.E., A characterization of ten rasterization techniques, *Computer Graphics,* **23** (3), (1989), 355-368.

20 Gouraud H., Continuous shading of curved surfaces. *IEEE Transactions on Computers,* **20** (6), (1971), 623-629.

21 Phong B., Illumination for computer generated pictures. *Comm. ACM,* **18** (6), (1975), 311-317.

22 Açıkgöz O. and Özgüç B., Texture mapping on geometrical models. *4th International Symposium on Computers and Information Sciences*, ODTU,(1989), 515-523.

23 Büyükkökten F., Isıklı O., Halıcı U., Halıcı E., EGS: Etkilesimli grafik sistemi. *TBD 8. Ulusal Bilisim Kurultayi*, (1990)..

24 Türün C.S. and Özgüç B., Concepts on digital animation. *5th International Symposium on Computers and Information Sciences*, ITU, (1990), 913-922.

25 Hearn D. and Baker M.P., *Computer Graphics*. Englewood Cliffs, NJ: Prentice-Hall, (1986).

26 Arslan A., Isler V., Akman V., A procedure to sweep arbitrary curves,*5th International Symposium on Computers and Information Sciences*, ITU, (1990), 895-904.

27 Güdükbay U. and Özgüç B., Deformation of solid models,*4th International Symposium on Computers and Information Sciences*, ODTU, (1989), 525-534.

28 Mandelbrot B., *The Fractal Geometry of Nature*. San Fransisco CA: Freeman, (1982).

29 Symbolics Paint and Animation System advertising brochure.

30 Özgüç B. , Thoughts on user interface design for multi window environments, *Second International Symposium on Computer and Information Sciences*, Istanbul, (1987), 477-488.