1983

CAST: A System for color animation and scene transformation

Frederick Schebor

Rochester Institute of Technology
School of Computer Science and Technology

CAST
A System for Color Animation and Scene Transformation

by

Frederick S. Schebor

Approved by:

Evelyn Rozanski
Associate Professor Evelyn Rozanski

Guy Johnson
Associate Professor Guy Johnson

Lawrence A. Coon
Assistant Professor Lawrence Coon

October 1,1983

Permission to Reproduce

Title of Thesis: CAST a System for Color Animation and Scene Transformation

I, Frederick S. Schebor, hereby grant permission to the Wallace Memorial Library, of RIT, to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Frederick S. Schebor

Date: November 1, 1983

1.0 PRELIMINARY INFORMATION

1.1 Abstract

CAST is an acronym for Color Animation and Scene Transformation. The objective of this system is to give non-computing personnel the ability to create still and animated pictures with the aid of a digital computer.

Data generation and animation language are two key components that comprise an animation system. The CAST system incorporates an interactive dialogue to assist the user in generating and/or editing three dimensional scenes, and an animation language which uses a simple script like syntax to describe parallel motion.

The third key component of an animation system is the display. The CAST system incorporates hidden surface elimination and surface shading necessary to produce realistic images. The output is in "device independent" form so that it may be adapted to virtually any type of display.

This thesis covers the design and implementation of the CAST system. Basic concepts as well as requirements, design problems/tradeoffs, and implications of this system are also discussed.

1.2 Key Words And Phrases

Animation, three dimensional, language driven, interactive, device independence

1.3 CR Categories And Subject Descriptors

I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Animation; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - Curve, surface, solid and object representations; I.3.3 [Computer Graphics]: Picture/Image Generation - Viewing algorithms.

1.4 Acknowledgments

I wish to extend my appreciation to the thesis committee: Guy Johnson, Lawrence Coon and especially Evelyn Rozanski. Besides providing guidence, she waited patiently for me to complete this thesis.

I also wish to thank Mike Lutz and Warren Carithers. Without their systems programming support, the full implementation of the CAST system would have never taken place.

1.5 Table of Contents

2.0 INTRODUCTION AND BACKGROUND

2.1 Introduction

The goal of this thesis is to produce a working language driven animation system. The term "language driven" implies that the animator communicates his ideas (both form and movement) to the system through the use of a language. This type of animation system has two problem areas which must be addressed :

1. The system must provide the animator with a language that is easy to learn and use.

2. The animator must be able to generate adequately complex three dimensional objects with relative ease.

CAST solves the first by giving the user an interactive language. In addition to being easy to learn, the interaction process gives the user continuous visual feedback of his work. As for the second, CAST provides a rich set of graphical primitives from which the user can create complex scenes.

2.2 Background

The first animation is attributed to the Dutch mathematician Pieter Van Musscheubroek (1692-1791) [3]. His method for animation used painted glass plates. These plates, when slid past one another, showed ships on an active sea, windmill vanes turning and people greeting one another. In 1832 Professor S. Sample of Vienna invented a motion device called the stroboscope. Around the perimeter of a disk are painted figures in consecutive positions. Between each figure is a hole. When the disk is rotated in front of a mirror, a series of convincing movements is observed when viewed through the holes.

In the middle of the nineteenth century, motion machines were produced with names such as the phenakistoscope, thaumatrope, zoetrope and the praxinoscope.

The later instruments became more sophisticated with the addition of viewing lenses and mirrors. Since these devices are cyclic in nature (e.g., a revolving disk or a revolving cylinder), their animations must also be cyclic. The last movement in the sequence must reproduce the original starting position otherwise a disturbing jump will take place with each revolution.

"Thumbooks" conveniently solved this problem. They were simple images printed on separate pages, that when flipped could reproduce a motion effect. Longer motion could be reproduced by simply adding pages to the book.

By the end of the nineteenth century the motion picture camera was introduced. The first animated films were created at the beginning of the twentieth century. In these first films, each frame had to be drawn by hand. This was a very tedious task since each frame consisted of a figure and a background. This was improved in 1914 when cell animation was developed. The idea was to separate the moving figures from the usually static background. The figures are drawn on clear pieces of acetate and are then photographed against the background. Since this technique reduced the time required to produce an animation, efforts could be directed towards further artistic developments. Most experts agree that animation reached it's peek at the Walt Disney studios in the 1940's. Films of that period still awe the moviegoer of the 80's. Yet their beauty did not come easily or cheap. A one hour animation required almost 90,000 unique drawings that had to be traced, inked, painted onto cells and then photographed.

With the advent of the computer, came the realization of the benefits gained by combining the computer with the animation process. The major benefits identified by Csuri [6] are:

1. Mathematical Content

   A professional animator would have a difficult time producing a film in which objects, positions and/or their movements have precise mathematical definitions. Subjects such as hydraulic flow and structure deformations which would prove difficult for an animator, can be easily handled by the computer.

2. Repetition.

   By varying parameters in a animation program, it is possible for the computer to generate a host of

different films. In conventional animation, the amount of time and effort that goes into a second film will be almost the same to that which went into the first film.

3. Time

   Computer controlled cathode ray tubes, plotters and film recorders can produce drawings much faster than their human counterpart. In addition, if the system can be left unattended, the animation process could run twenty four hours a day thereby increasing overall output.

4. Cost

   Since the amount of work produced by the computer over a specified unit of time will be greater than that of a professional animator, it should be cheaper to use.

The realization of these benefits came quickly. The first computer-generated movie can be attributed to Cecil Leith and George Michael [18]. The film was produced for a classified research project in hydrodynamics around 1956. One of the problems in generating computer films is getting the data off the computer and onto film. This was solved in 1960 with the manufacture of a computer controlled microfilm recorder. This device was intended for permanent high density data copying onto microfilm, but with appropriate software changes, it could be made to produce 16 mm movies. BEFLIX (a corruption of Bell FLicks) was the first animation system to make use of the microfilm recorder [11]. Written at Bell Labs in the early 60's it gave the user two levels of language. The lowest and most detailed allowed the programmer to vary gray levels in a 252 X 184 array of spots. At the higher level the programmer can instruct the system to draw lines, arcs, letters; copy, shift and translate spots; zoom, dissolve and fill areas.

BEFLIX falls into a category of animation systems that are called "language driven" systems [13]. This type of system achieves its flexibility by giving the animator a wide inventory of picture manipulation commands. Also included in this group is the CAFE system [21]. This system incorporates an easily learned non-procedural language that allows single axis rotations and is designed for on-line use. The Graphics Symbiosis System (GRASS) [7] system uses dials and joysticks instead of a language. It permits real time, wire frame, three dimensional animation.

There are two problems that systems in this group must overcome. The first deals with data generation i.e., the artist must be able to generate complex three dimensional objects with sufficient ease. The second problem area is based on the language itself. Since it is the use of the animation language which manipulates the objects, the artist must acquire some familiarity with the language syntax. The Synthavision system [5] solves the first problem by giving the animator a set of geometric primitives (e.g., cube, cone, torus) which can be combined to form more complicated objects. This system is heavily used by the industrial-education market.

Another class of animation systems are known as "picture driven" or "key frame" animation systems. In this class of systems, the animator prepares key images in the sequence and enters them into the computer. The computer then generates the in-between images. The first system, called KARMA [1], used only linear interpolation to proceed from key frame to key frame. No shading, rotation, scaling or three dimensional effect was provided. The most notable of these systems was the system written by Burtnyk and Wein [2]. This system was used by the National Research Counsel of Canada to produce many animated films. The advantage to key frame systems is that they mimic conventional animation where key positions are established on a story board. However, these systems are not without their own set of problems. Although the images may represent the three dimensional world, they must be entered into the computer in two dimensional form. Thus it is extremely difficult to animate movements such as complex rotations. Secondly, if the key images are not closely related, interpolation problems may result.

The third class of animation systems consist of the "color table" animation systems [19]. These systems create the impression of movement by only changing an object's color. The object itself remains stationary. For example, suppose the animator wanted to create a bouncing ball. He would create individual balls and position them along the ball's path. Their initial color would be the same as the background i.e., invisible. To produce the animation, he would change the color of the balls one at a time and in order. The resulting animation would mimic the original ball's path. The problems with this system center around the limited effects that can be generated. Since the system only changes the color of an object, successive positions of the objects cannot intersect one another.

The past ten years have produced the current crop of "state of the art" systems. These systems produce animated three dimensional graphics with the aid of high performance algorithms implemented in hardware. Most of this hardware was developed for use in flight simulators. Flight simulators are devices that allow air crews to train without using actual aircraft. They simulate the out-of-the-window views. The first major application was General Electric's multi-million dollar color display system built for NASA to aid the Apollo program. Concave three dimensional objects with 250-300 edges were displayed in real time [16]. In 1980, G.E. delivered their fourth generation flight simulator system. Designed to simulate a B-52, it contains a 250,000 square mile data base that can be displayed using day, dusk and night conditions. It is capable of displaying 8000 edges in real-time.

2.3 Theoretical And Conceptual Development

The CAST system is built on three key components:

1. Data generation
2. Display
3. Animation

2.3.1 Data Generation -

Data generation is the process which enables the animator to input the objects required for the animation. In order to be effective, data generation must be built around a method that will permit the user to create three dimensional objects with relative ease. Many systems require the operator to input a large number of point locations that lie on the surface of the object they describe. Although flexible and precise, it is a very time consuming process. The method used by CAST was pioneered by the Synthavision system. This involves giving the user a standard set of geometric primitives. The process of constructing a three dimensional scene amounts to defining the location (in the Cartesian coordinate system), the dimensions and orientation of the primitives. These primitives are chosen from a library of twelve types and are the basic building blocks of the CAST system.

The primitives include:

1. Point
2. Line
3. Triangle
4. Plane
5. Arc
6. Circle
7. Box
8. Wedge
9. Pyramid
10. Sphere
11. Cylinder
12. Cone

For example, suppose the animator wished to create a scene which contained a house. A simple house could be described as a wedge sitting on top of a box. A lawn would be modeled as a green plane positioned under the box. Doors and windows would be rectangular planes attached to the sides of the box.

One problem is apparent with this type of structure. Suppose the user wished to apply a transformation (e.g., rotation) to the house during the animation. Normally this would require that he transform all of the objects (box, wedge and plane) separately but in step with one another. CAST solves this problem by giving the user a joining operator. With this operator, primitives may be logically joined together and treated as one object. The operator requires that one object in a group assumes the role of "parent" and the others are "descendants". Any operation performed on the parent will affect all of its descendants. Any operation performed on a descendant will not affect its parent or siblings.

The parent descendant relationship can be described by a tree structure. The house example could be described by the following tree:

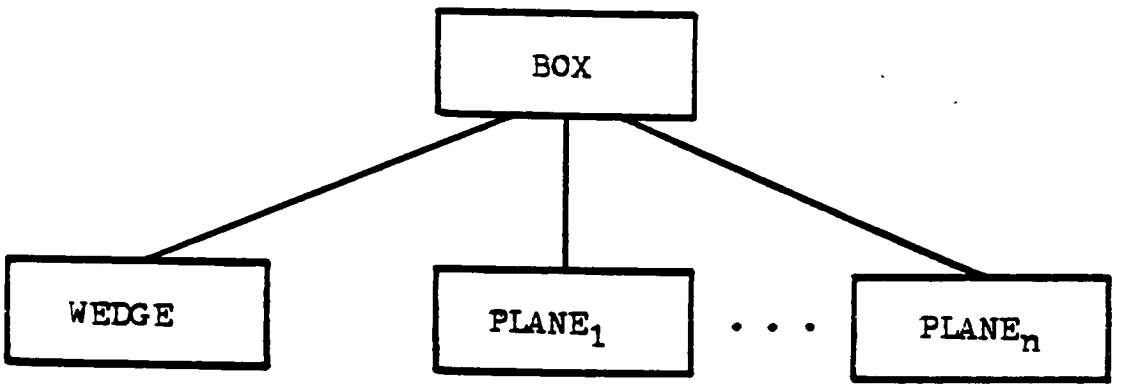


Figure 1

Suppose the scene consisted of many houses. The box primitives could be treated as descendants of a node called the Scene Node. The resulting tree would then be:

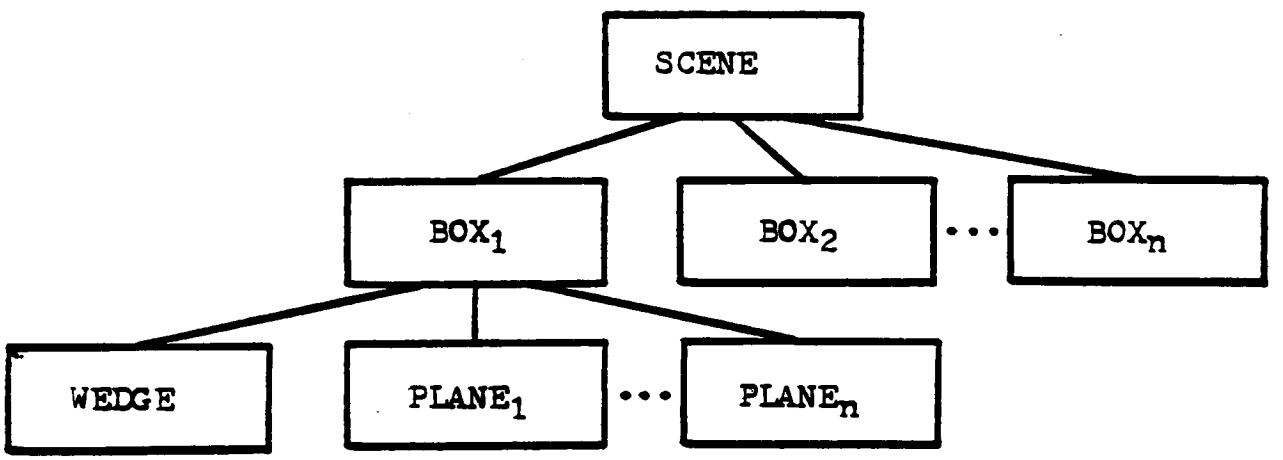


Figure 2

With this structure (called the Scene Tree), the user could transform every object in the scene by transforming the Scene Node. The joining operator is a key feature of the CAST system.

All primitives are described by a list of attributes. These attributes consist of:

1. Object name - 1 to 10 characters supplied by the user.

2. Defining points - Points which define the size and orientation of the object.

3. Object color

4. Object presentation - Either solid or wire frame

5. Object shade - A flag indication whether the object is to be displayed with shading.

If an object contains more than one face, each face can have its own color, presentation and shade. Object presentation and shade attributes do not apply to points and lines.

CAST incorporates an interactive BUILD/EDIT subsystem to aid the animator in creating scenes. The BUILD subsystem is used for creating new scenes and the EDIT subsystem for editing previously built scenes. When the BUILD subsystem is entered, the user is prompted to enter the scene name, the background color, and the stage size. Colors include red, yellow, green, cyan, blue, magenta, black and white. The stage size is the distance from the viewing location at which objects will not be displayed i.e., beyond a back clipping plane. Providing the animator with a finite stage allows him to "hide" objects that are not required on stage at some point in the animation.

Once the initial scene information is entered, the user can then add objects to the scene. When adding objects, the user is prompted for the object name, the name of its parent, point and face information. The size and orientation of each primitive is described by a set of points. The operator is prompted to enter the coordinates of each point in the set. Figure 3 shows the primitives, their points and faces.

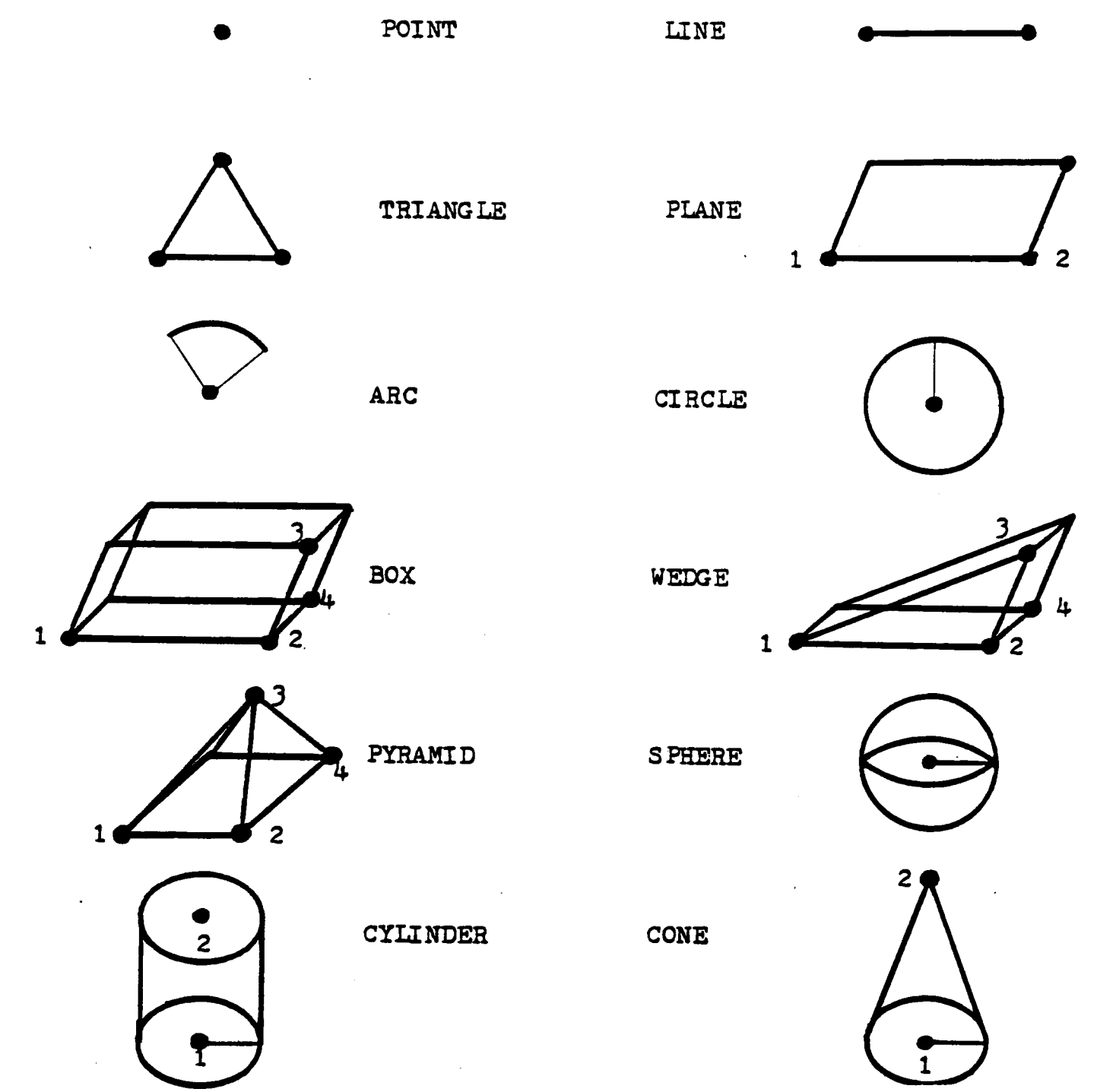


Figure 3

Once the points have been entered, he is prompted for the face information. After a primitive has been entered, any of these attributes may be changed through the use of

the MODIFY command. If a primitive is not needed it may be removed from the scene by using the DELETE command. DELETE makes use of the parent-descendant relationship. If a parent is deleted, all of its descendants are also deleted. When the user is satisfied with the scene, he records it with the SAVE command. This command creates a disk file in which the scene is saved. The scene file is used as input for the EDIT or ANIMATE subsystem.

In addition to primitives, the user has control over a CAMERA and a LIGHT. It is through the camera that the user views his scene. Although conceptual in nature, it behaves in the same manner that an ordinary camera would behave. The user can specify a camera position, rotations and zoom factor. The animator is given control over the light source the illuminates a scene. He can create variations in primitive shading by moving the light about the stage.

Throughout the data generation process, the user is always prompted for information, whether it be a command or some primitive attribute. If he becomes lost at any point, typing HELP will tell him where he is and what he is expected to enter. In addition, error checking on user input is done immediately after it is entered.

2.3.2 Display -

Display is the process of giving visual feedback of the animation to the user and to record the final animation sequence. In order to produce realistic images, CAST incorporates hidden surface elimination and surface shading. In addition, the output is in "device independent" form.

Hidden Surface Removal

The removal of the hidden parts from a scene is one of the most active areas in computer graphics. The solutions are many and varied [20]. These algorithms may be divided into two groups. Those that work in object space and those that work in image space.

Object space algorithms determine which parts of the objects are visible by investigating the geometrical relationships between them. However, an image space algorithm works with the final image and determines what is visible at each pixel on the display screen. Object space algorithms tend to concentrate on hidden line removal where

as image space algorithms concentrate on hidden surface removal.

The algorithm used by CAST to remove hidden surfaces works in image space and is known as the Z-Buffer or Depth Buffer algorithm. This is the simplest of the image space algorithms. It requires that two arrays be kept, one for color information and one for depth information. The arrays are indexed by pixel coordinates (x,y). The algorithm is as follows:

1. For all pixels on the screen, set depth(x,y) to the scene size, and the color(x,y) to the background color.

2. For each its boundaries when the object is projected onto the screen. For each of these pixels:

3. Calculate the z depth of the object at point (x,y).

4. If z < depth (x,y) then this pixel is closer to the observer than the previously recorded pixel. If this is true, set depth(x,y) = z and color(x,y)= a value corresponding to the color at that point. If z>depth(x,y),then the pixel is obscured by the already recorded pixel, hence no action is taken

After all the objects have been processed, the color array will contain the image with hidden surfaces removed.

The main reason for choosing the z-buffer algorithm is that of simplicity. It is an easy algorithm to understand and implement. The display process involves breaking primitives up into planes which are broken up into lines by a scan line algorithm. The lines are broken up into pixels which are ready to enter into the z-buffer. Pixel depth is determined by linear and planar equations. The disadvantage to the z-buffer algorithm is that of space. Each pixel on the display screen must have two variables in memory, one for color and one for depth. A 512 X 512 display screen requires 524,288 storage locations. The z-buffer may reside in memory on large virtual addressing machines, but smaller systems would require the z-buffer to reside on disk and thus have a great impact on display computation time.

There are many schemes for specifying color [15]. CAST uses the HSI system of color description for the z-buffer. HSI uses three numbers to specify a color's characteristics:

1. Hue - the 'color' of the color.

2. Saturation - the degree to which it is undiluted by white. For example, pink is an unsaturated red.

3. Intensity - the brightness.

Hue is usually described as an angular position from 1 to 360 degrees. Red serves as the origin at 0 degrees. This is followed by yellow at 60, green at 120, cyan at 180, blue at 240 and magenta at 300. The complement of a color is its angular value plus 180 degrees. Intensity and saturation are specified as percentages from 0 to 100%. The advantage of the HSI system is that shading can be accomplished by simply changing the intensity value.

Shading

CAST incorporates object shading in order to produce realistic images. An object's shade is determined by three factors:

1. The amount of light falling on the object.

2. The object's orientation with respect to the light.

3. The object's reflectance.

Planes are shaded according to the equation:

$$E = (R \cos A) * I$$

where:

- E - Energy transmitted from the plane.
- R - The coefficient of reflectance for the plane (0.0 - 1.0)
- A - The angle of incidence between the plane and light source.
- I - The energy arriving from the light source.

Although the distance from the light source to an object may vary, CAST maintains that the amount of light falling on the

object will be constant. CAST also assumes that the objects are 100% reflective and that the light source is reflected uniformly in all directions (diffuse reflection).

Device Independence

"Device independence" is a methodology whereby software is isolated from machine peculiarities. CAST interfaces to a particular device through a set of low level routines called a "device driver". The driver contains routines to go into and out of graphics mode, set color, and draw a point or line. Device independence permits interfacing CAST to virtually any type of graphic display by replacing simple low level routines rather than complicated high level routines.

2.3.3 Animation -

Animation is the process of "giving life" to the objects which were defined during data generation. This is done by linking motions to the objects in the scene. These motions imitate actions of the physical world. The motions can involve a change in an object's position, orientation or size.

Traditional methods of filming link motions to objects by the use of a script. The script states which objects will have which motions, at which times. Note that the script is "key frame" in nature. For example a script might state that an actor walk from point A to point B in one minute. The script has stated the object (actor), the motion (walk), the motion limits (A,B) and the duration (1 minute). It is up to the actor to construct the in-between motions i.e., the process of putting one foot ahead of the other. The concept of an animation script is a key component of CAST.

In order to be read and executed under computer control, the script should follow some type of standard syntax. The CAST script syntax is patterned after the script syntax used by the ANIMA II system [10]. The script is broken into commands with one command per line. Each command has the form:

transformation, object name, transformation limit, time period

CAST transformations include SET, MOVE, ROTATE and SCALE. The object name is that which was supplied during data generation. The transformation limit may be; a coordinate (SET, MOVE), an angle (ROTATE) or a factor (SCALE). The time period is specified by the number of frames during which the transformation will be in effect. Since the standard animation film speed is 24 frames per second, a transformation applied over 24 frames will be one second long. The script command:

```
SET box1 1000,1000,1000 FRAME 1
```

will position the object called box1 at location 1000,1000,1000 at frame number 1. The script command:

```
ROTATE box1 X,180 FRAME 1,10
```

will rotate the object called box1, 180 degrees about its X axis from frames 1 though 10. Remember that during data generation, objects may be part of other objects. In the above examples, if box2 and box3 were PART OF box1, they would be transformed by the same transform applied to box1. This is where CAST displays its power. Complex transformations need not be the result of applying transformations to every single object, but by applying simple transformations to the parents.

In order to be effective, the animation script must not only control a scene and/or the objects in a scene, but it must give full control over the camera that views the scene. Note: this is not the camera that actually films the animation, but it is the point at which the scene is viewed. In the CAST system, the camera, although conceptual, behaves in the same manner that an ordinary camera would behave. The animator can make use of commands such as SET, MOVE, TILT, PAN, SPIN and ZOOM. The system gives the animator the same camera control that he would have in the physical world. The one exception to this is focus and aperture. Since the depth of field is infinite, there is no need for a focus/aperture control.

Finally, the animator has control over the light source that illuminates the scene. He is provided with commands which can SET or MOVE the light source during the animation.

3.0 FUNCTIONAL SPECIFICATION

3.1 Functions Performed

As stated in 2.1, the object of this thesis is to produce a working "language driven" animation system. The functions needed to perform this are:

1. A function that will give the animator the ability to create and/or edit three dimensional scenes.

2. A function for animating those scenes.

3. A function for displaying the scene with hidden surfaces removed and surface shading.

Since CAST is language driven, the system must support an interactive dialogue.

3.2 Limitations And Restrictions

The user should realize that this system is a first attempt. The design and implementation of an animation system not only requires knowledge of computer graphics, but also the graphical implications of operating systems, programming languages, information structure, hardware design and human factors. Each one of these areas could be the topic of other theses. As the use of CAST continues and experience is gained, decisions can be made as to what areas require further enhancements.

Current limitations include:

1. Limited number of hues.

2. No provision for planar transparency i.e., a plane is either solid or wire frame.

3. No means for copying a primitive.

4. Script commands must be sorted on ascending end frame number.

5. No provision for using other types of graphical input such as joysticks or tablets.

6. No shadow casting.

7. Animation cannot be produced in real time.

Probably the most serious limitation is 7, the lack of real time capability. Currently, the animator records the animation on film (a slow process in itself) which then must be processed. The wait period between recording the animation and viewing it may be intolerable. This will be especially true in the initial stages when script bugs will unexpectedly pop up. The only way to improve the speed is with the addition of high performance (high cost) hardware.

3.3 System Inputs

CAST receives input from the following:

1. Keyboard

2. Scene file

3. Script file

4. Assembled script file

3.3.1 Keyboard -

Since CAST is an interactive system, the user is in constant communication with it. The system receives all of the user input from the keyboard. Through the keyboard, the user can enter commands or respond to system prompts.

3.3.2 Scene File -

This file is used as input to the EDIT and ANIMATE subsystem. It contains all of the information necessary to restore the scene. It also contains information concerning the state of the camera and light.

3.3.3 The Script File -

Since filming an animation is likely to be a time consuming process, the animation process takes place in a batch mode environment rather than an interactive one. This requires that the script reside in a file which can be read, assembled and executed under program control. The script file is produced by the host's text editor.

3.3.4 Assembled Script File -

This file, generated by the script assembler, contains the assembled script.

3.4 System Outputs

CAST produces output for the following:

1. Graphic display device
2. Alphanumeric display device
3. Scene file
4. Assembled script file
5. Log file
6. Camera control

3.4.1 Graphic Display -

Picture output is the end result of the animation. During picture output, the z-buffer is mapped onto the display device using the terminal's graphic capability for the mapping function. This mapping is done in a best fit fashion. For instance if the terminal has color capability but no shading, only the hue value is mapped from the z-buffer to the terminal.

3.4.2 Alphanumeric Display -

Besides echoing the keyboard input, this device displays all user prompts and error messages.

3.4.3 Scene File -

This file is produced by CAST if the user instructs it to save the current scene. This can be done while he is in the BUILD, EDIT or ANIMATE subsystems. The file contains all of the information necessary to restore the scene for subsequent animation and/or editing. In the current implementation, this file is saved as an ASCII text file. This permits the experienced user to simply list this file at his terminal in order to determine its contents.

3.4.4 Assembled Script File -

This file is produced by the script assembler. In it animation commands, command parameters and object names are replaced with numbers and pointers. Although "low level" in nature, it is saved as an ASCII text file and therefore can be listed and/or edited by the host.

3.4.5 The Log File -

In the event of a system crash while in the middle of filming and animation, CAST provides the user a means for recovering. The system maintains a log file which contains status information. The log file is a text file where each line contains a frame number and the date/time that the frame was recorded on film. Thus each line signifies the successful completion of one frame in the animation. By examining at the last line in the log file, the user will see the current state of the animation.

3.4.6 Camera Control Output -

In order to allow for unattended operation, some form of camera control is needed. When a frame has been output to the terminal, the host must signal the camera to fire. The software must wait for the shutter completion before erasing the picture and continuing.

4.0 ARCHITECTURAL DESIGN

Figure 4 shows the architectural diagram of CAST.

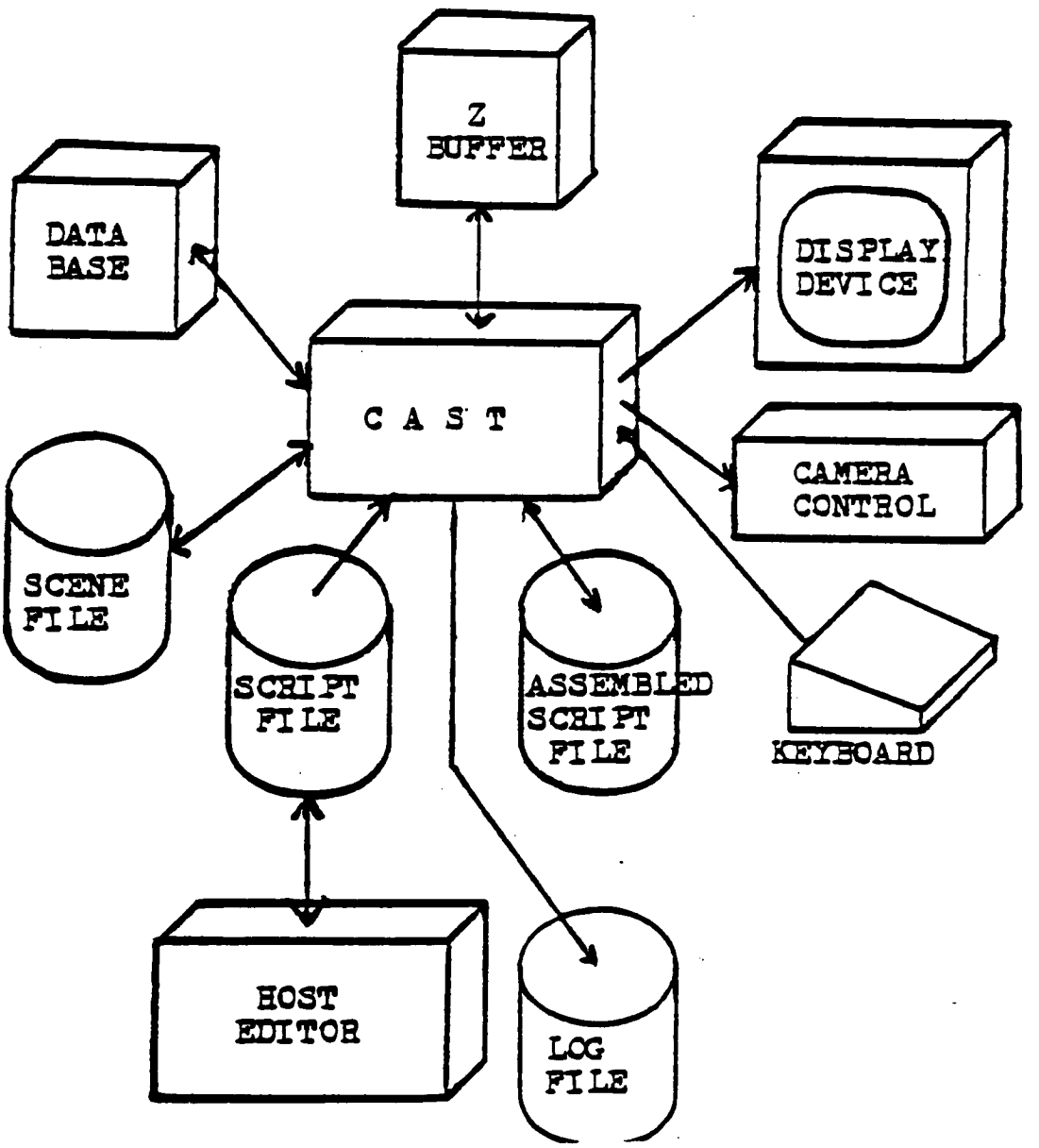


Figure 4

CAST reads input and writes output to the following areas:

1. Keyboard - All user input is received from this device.

2. Display - All user output and graphical output is displayed here.

3. Database - This is where the scene is held during BUILD/EDIT and ANIMATE. Also included are the camera and light parameters. During execution, the database resides in memory.

4. Scene File - The scene is copied from the data structures into this file as a result of the SAVE command while in BUILD/EDIT or the SAVE FRAME command while in ANIMATE.

5. Z-Buffer - Created by the BUILD/EDIT and ANIMATE subsystems. It is a three dimensional array that holds graphical output. It resides in memory during CAST execution.

6. Script File - When entering the ANIMATE subsystem, CAST gets the script from the script file. This file is generated by the user by running the host's text editor.

7. Assembled Script File - As the script is assembled, it is written into this file. This file is used for both input and output during animation.

8. Log File - Keeps a record of the frames produced during an animation.

9. Camera Control - On command from the host, this device will trigger the camera.

5.0 INTERFACE SPECIFICATION

5.1 External Interfaces

CAST communicates with the outside world with several different types of data:

1. User input
2. User output
3. Graphic output
4. Camera control
5. Scene file
6. Script file
7. Assembled script file
8. Log file

5.1.1 User Input -

All user input is received through the keyboard. Through it, the animator enters commands or parameters in response to CAST prompts.

The following is a look at the user input by subsystem.

Note: Bolding indicates the legal abbreviation.

Monitor Subsystem

1. animate - Run the ANIMATE subsystem.

2. build - Run the BUILD subsystem.

3. edit - Run the EDIT subsystem.

4. help - Display the monitor help file.

5. stop - Stop the CAST system and return to the host operating system.

BUILD / EDIT Subsystem

The BUILD/EDIT subsystem has two types of user input: command input and parameter input.

Command Input

1. add - Add a primitive to a scene.

2. delete - Delete a primitive from a scene.

3. display full - Display the scene using the full terminal capability i.e. color, shading, etc.

4. display line - Display a wire frame representation of the scene.

5. help - Display the BUILD/EDIT help file.

6. modify - Modify a primitive in a scene.

7. print extended - Print a detailed list of the primitives in a scene.

8. print short - Print a condensed list of the primitives in a scene.

9. print standard - Print a list of the primitives in a scene.

10. save - Save a scene.

11. setlight - Set the light position.

12. setview - Set the camera's parameters.

13. **stop** - Exit the BUILD/EDIT subsystem and return to the monitor.

Parameter Input

When working in the BUILD/EDIT subsystem, the animator will be prompted to enter various types of parameters. These include:

1. Object name - one to ten alphanumeric characters.

2. Object type - point, line, **t**riangle, **pl**ane, **a**rc, **c**ircle, **w**edge, **b**ox, p**y**ramid, **s**phere, **cy**linder and **co**ne.

3. Point location - three numbers specifying X, Y and Z coordinates. Each number can contain four digits to the left of the decimal point and three digits to the right.

4. Radius - one number specifying a radius.

5. Degrees - two numbers specifying the start and end angles for an arc.

6. Rotations - Numbers specifying X and Y axis rotation angles.

7. Color - bla**c**k, b**l**ue, green, **c**yan, **r**ed, **m**agenta, **y**ellow and **w**hite.

8. Solid - **y**es or **n**o.

9. Shade - **y**es or **n**o.

10. Zoom factor - A number between 0 and 100 that specifies the current lens magnification.

ANIMATE Subsystem

The ANIMATE subsystem receives its input directly from the user or from the script file. The following will be requested by the system prior to running the animation:

1. Script name - file name of the script. One to ten characters.

2. Display type - The type of picture output. Line or full.

5.1.2 User Output -

All user output is displayed on the alphanumeric screen. The system communicates information to the user with the following four types of information:

1. Prompts

2. Error messages

3. Help facility

4. Data structure dumps

The output of prompts and error messages is under the control of the system. The help facility and data structure dumps are only displayed by user request.

CAST Prompts

The type of prompt displayed will indicate in which subsystem the user is located.

1. <CAST> - Output by the system monitor.

2. <BUILD> - Output by the BUILD subsystem.

3. <EDIT> - Output by the EDIT subsystem.

Since the help and the ANIMATE subsystems do not have their own interactive language, no prompts (other than specific questions) are displayed.

Error Messages

The system communicates its error status to the user through error messages. These messages fall into three classes:

1. Warning

   A warning message is produced whenever user input might result in unexpected results such as leaving the editor without saving the modified scene. The warning message requires the user to specify his intent by answering a yes or no question e.g. Changes will not be saved, are you sure? A 'yes' response executes the command where as a 'no' response will abort it.

2. Error

   This type of message is displayed whenever the system detects an error in user input. A reply of "purple polka dots" to a prompt for a face color will produce such an error. The system will always recover from this type of error.

3. System Error

   If the system detects an error from which it can't (or shouldn't) recover from, a system error message is produced. This type of error indicates a problem with the software. It should not occur because of an error on the user's part. Included with the error message will be various parameters and a dump of the data structures. This will allow a programmer to track down the source of the problem.

CAST Help Facility

If the user responds with "HELP" while in the monitor, the BUILD subsystem or the EDIT subsystem, he will be provided with a list of allowable commands for that level. This allows the new user the possibility of running the system without the need of having a user manual at hand.

Data Structure Dump

Just as a memory dump is useful to the programmer, a data structure dump will be useful to the animator. There are three types of dump:

1. Short - List the names of the scene, primitives and the primitives parents.

```
  SCENE: galaxy
-------------------------------------------------------

  OBJECT: e.ursamaj   PART OF: galaxy
  OBJECT: dubhe       PART OF: galaxy
  OBJECT: n.ursamaj   PART OF: galaxy
```

2. Regular - Same as short but include a list of the primitive's attributes.

```
  SCENE: galaxy
-------------------------------------------------------

  OBJECT: e.ursamaj   PART OF: galaxy   TYPE: point
  POINT LOCATION: 3861.770, 4639.180, 18792.850
  COLOR: white

  OBJECT: dubhe       PART OF: galaxy   TYPE: point
                        :
                        :
                        :
```

3. Extended - Same as regular but include primitive addresses and a data structure dump.

```
  SCENE: galaxy   SIZE: 13106   BACKGROUND COLOR: black
-------------------------------------------------------

  OBJECT: e.ursamaj   PART OF: galaxy   TYPE: point
  NODE ADDRESS: 4   VECTOR ADDRESS: 4
  POINT LOCATION: 3861.770, 4639.180, 18792.850
  COLOR: white

  OBJECT: dubhe       PART OF: galaxy   TYPE: point
                        :
                        :
                        :
```

```
VIEW SET:
    4135.260 4701.810 18374.840 .000 .000 .000 1.000

    LOC    MA    MI    NODE          OVP      OV
    ---------------------------------------------
     0      3     3                    21      94
     1      0     0                 13106    .000
     2      0     0                     0    .000
     3      4    -1    galaxy          -1    .000
     4     -1     5    e.ursamaj        4    .000
     5                  :
                        :
                        :
```

Graphic Output

Picture output is the end result of the animation. A device dependent driver maps the device independent z-buffer to a particular device. Keeping the picture output in device independent form will allow the CAST system to drive virtually any type of graphics terminal. The system currently supports three graphic terminals:

1. Chromatics GC 1999 raster terminal

2. Ramtek 6211 raster terminal

3. DEC Gigi raster terminal

Camera Control Output

In order to allow for unattended operation, some form of camera control is needed. Currently the camera interface hardware (Haltek) sits on the RS-232 line between the host and the terminal. When a frame has been output to the terminal, a CONTROL G character sent by the host will be detected by the camera interface and the shutter will be tripped. The software must wait for the shutter completion before erasing the picture and continuing.

The Scene File

This file is produced as a result of the SAVE command. The file contains all of the information necessary to restore the scene for subsequent animation and/or editing. In the current implementation, this file is saved in sequential records.

The Script File

The script resides in a file which can be read, assembled and executed under program control. This file consists of records with one animation command per record. The record may be any length but only the 1st 80 characters are read. The script file is produced by the host's text editor.

Script Commands

Object commands:

1. SET object_name x,y,z FRAME n
2. MOVE object_name x,y,z,FRAME n,m
3. ROTATE object_name axis,degrees FRAME n,m
4. SCALE object_name factor FRAME n,m

Camera Commands:

1. SET CAMERA x,y,z FRAME n
2. MOVE CAMERA x,y,z FRAME n,m
3. PAN CAMERA degrees FRAME n,m
4. TILT CAMERA degrees FRAME n,m
5. SPIN CAMERA degrees FRAME n,m
6. ZOOM CAMERA factor FRAME n,m

Light Commands:

1. SET LIGHT x,y,z FRAME n
2. MOVE LIGHT x,y,z FRAME n,m

Other Commands

1. SAVE FRAME n
2. * Comment

Assembled Script File

This file contains the assembled version of the script. Commands are replaced by opcodes, primitive names are replaced by pointers and absolute parameters are replaced by incremental ones.

The Log File

The log file is produced by the ANIMATE subsystem. It is a text file where each line contains a frame number and the date/time that the frame was recorded on film.

5.2 Internal Interfaces

Internally, most data is passed from routine to routine through the data base. See 6.2 for detailed discussion. In addition to the data base, most routines require additional data in order to perform their function. This data is temporary in nature and each subsystem uses a particular type of data.

5.2.1 BUILD/EDIT Subsystem -

Since this subsystem modifies scenes and/or objects in the database, the following data types are needed:

1. Names - scene, object, parent

2. Pointers - object, parent

3. Object parameters - type, coordinates, face information.

In addition, object deletion will require a stack where pointers to descendants may be stored. In order to determine whether the data base has been modified since the last SAVE operation, a boolean status flag will be maintained.

5.2.2 ANIMATE Subsystem -

In order to assemble and interpret scripts, variables will be needed to store:

1. Strings

2. Pointers into strings

3. Line numbers

4. Object pointers

5. Frame numbers

6. Command parameters

Since an operation performed on a parent object will also be applied to its descendants, the stack mechanism developed for the BUILD/EDIT subsystem will also be used in the ANIMATE subsystem to stack pointers to descendants. Routines which transform objects must be able to create and access a transformation matrix.

5.2.3 Scene Display -

The process of displaying a scene involves coordinate manipulation by transformation matrices and updating the z-buffer. The z-buffer is a three dimensional array and the current size is 512 X 512 X 2. At each point, the first value is the depth at that point, and the second is the hsi value at that depth.

The hsi value is a seven digit number of the form:

HHHSSII where:

H = hue (0 - 360 degrees)
S = saturation (0 - 99)
I = intensity (0 - 99)

In order to map the z-buffer onto the display screen, device parameters must be known. This includes:

1. Width of the screen in millimeters

2. Viewing distance in millimeters

3. Screen address boundaries

Since the user has the option of displaying the scene in either wire frame or shaded hidden plane form, a flag is kept to denote which option is in effect.

And finally, since the cords that approximate an arc are computed at display time, an array is provided to hold the coordinates for transformation.

6.0 MODULE DESIGNS

6.1 Processor Modules

CAST is currently implemented using 169 RATFOR routines. There are far too many to give a detailed design here. Since the implementation is built on previous work, only an overview will be presented. Specific questions should be directed to the RATFOR listings. See 10.4.

6.1.1 CAST Monitor -

The Monitor gives the user the prompt and will wait for input. When correct input is given, it will call the appropriate subsystem.

6.1.2 BUILD/EDIT Subsystem -

These routines will add, delete and modify objects in the data base. In addition, a provision for saving and restoring scenes must be provided. As explained in 6.2, the database is built around a tree structure. Algorithms which work on the data base may be traced to [12].

6.1.3 ANIMATE Subsystem -

This subsystem can be broken up into three sections:

1. Script assembly
2. Script interpretation
3. Scene display

Script Assembly

The script assembler is a simple one pass assembler which processes commands and their immediate operands. In the assembly process, commands are translated into numbers, object names into pointers and parameters into relative operands. Assembler algorithms can be found in [8].

Script Interpretation

Script interpretation is a simple process. For each frame in the animation, each command line will be read and the start and end frame numbers checked. If the current frame is between these two, the object and its descendants pointed to by the pointer operand are modified by the relative operands. This requires placing the object and its descendants into a stack and setting up the appropriate transformation matrix. For each object in the stack, the coordinates are copied into an array from the database, modified by the transformation matrix, and replaced into the database. When the last command has been processed, the data structures are displayed and the process repeats. After the last frame has been displayed, control returns to the monitor.

Scene Display

Scene display can be broken up into two areas:

1. Z-buffer construction
2. Z-buffer output

Z-Buffer Construction

Specific algorithms concerning z-buffer construction can be found in [14]. As described in 2.3.2, the z-buffer is a three dimensional array where each point contains a hue, saturation, intensity value and the depth of the value. Since the hsi and depth value of each pixel must be determined, the objects must be broken up into their constituent pixels. For example, a sphere is broken into planes, each plane is broken into scan lines, each scan line is broken into points which are then entered into the

z-buffer.

In order to allow for device independence, the z-buffer coordinates are device independent. The mapping from device independent to device dependent is done during z-buffer output.

Using camera and viewport parameters, two transformation matrices are constructed. One contains X, Y, Z translations and the Y axis rotation, and the other contains the X and Z axis rotations. When these matrices are applied to an object's coordinates, they will be transformed from scene coordinates to device independent coordinates. It is the objects which are specified in these coordinates which are then broken up into points and entered into the z-buffer.

Z-Buffer Output

Once every object has been transformed and entered into the z-buffer, the output to the graphic display can begin. Going from left to right and bottom to top, adjacent points with equal hsi values are concatenated together to form lines. These lines are then output to the device driver. It is up to the device driver to map the line into device dependent coordinates and the hsi values into like values based upon the device's capability.

6.2 **Data Base**

The CAST data base contains all the information about the scene currently being built, edited or animated. The data base resides in memory during execution. The most important constituent of the data base is the scene tree (see 2.3.1). CAST is written in RATFOR which doesn't permit structured data types or dynamic allocation. Since the most advanced data type is the array, some means of converting a multi-level multi-degree tree to a finite array must be found.

If each primitive in a scene could be represented by a single row within a N X M array, the problem would be solved (as long as the number of primitives <= N). Since each primitive has a unique name, one element can be reserved for this. The primitive's parameters pose a problem. Since different primitives require a different number of parameters, they cannot be kept in a finite array very

efficiently. This problem is solved by putting all the parameters in their own vector (called the parameter vector), one after the other. Then a word in the array can be reserved for a pointer into the parameter vector. The next problem concerns the descendant pointers. Since the number of descendants of a primitive can vary, it is not feasible to include them in the array. This could be solved using the same method that was developed for the parameters (e.g., a pointer into a list of pointers). This can be carried one level farther so that we have a pointer to a list of primitives. In effect, we would have a tree of lists. With this tree of lists, it becomes apparent that we need only two types of pointers:

1. A pointer to the list of descendants.

2. A pointer from one descendant in the list to another descendant in the list (a sibling pointer).

Our multi-degree tree described in 2.3.1 can be implemented as a binary tree.

The following scene:

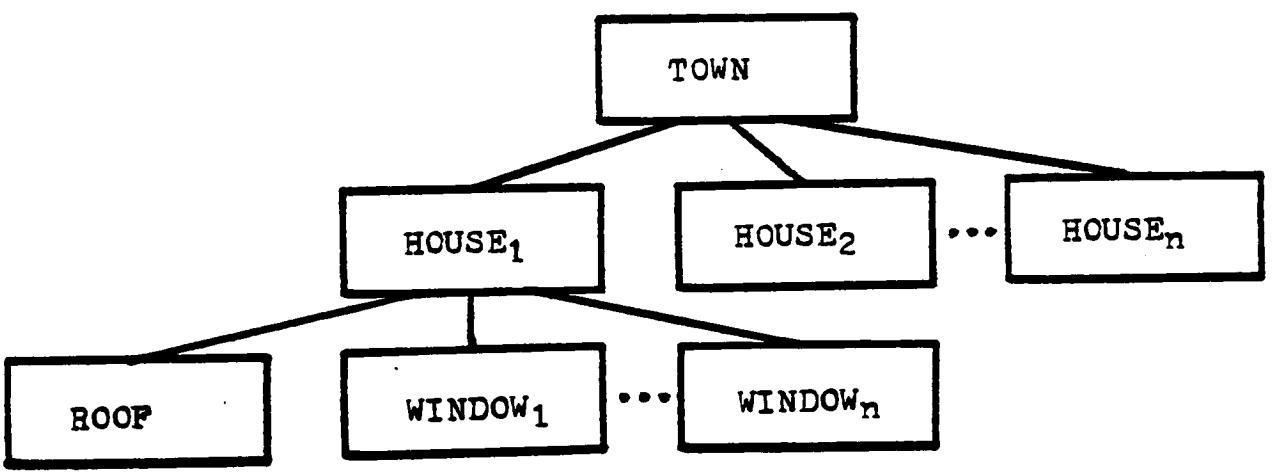


Figure 5

can also be represented as:

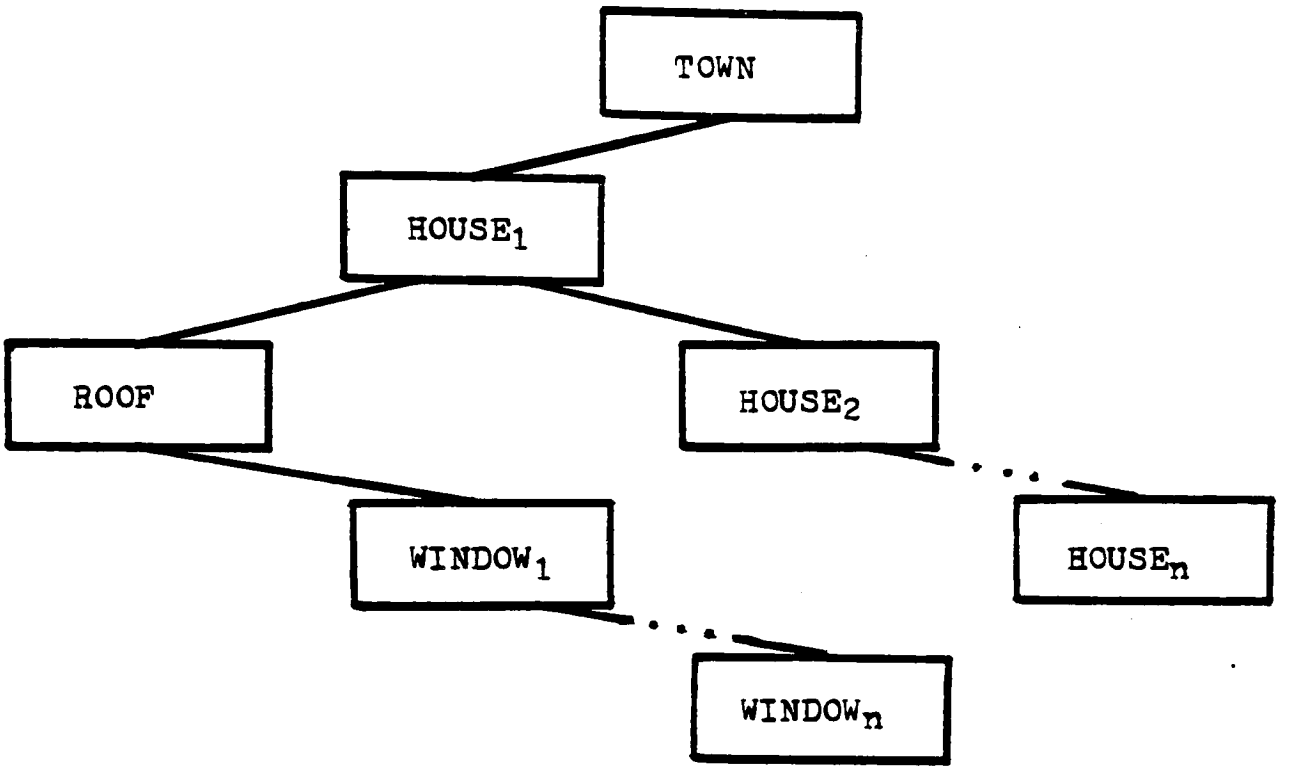
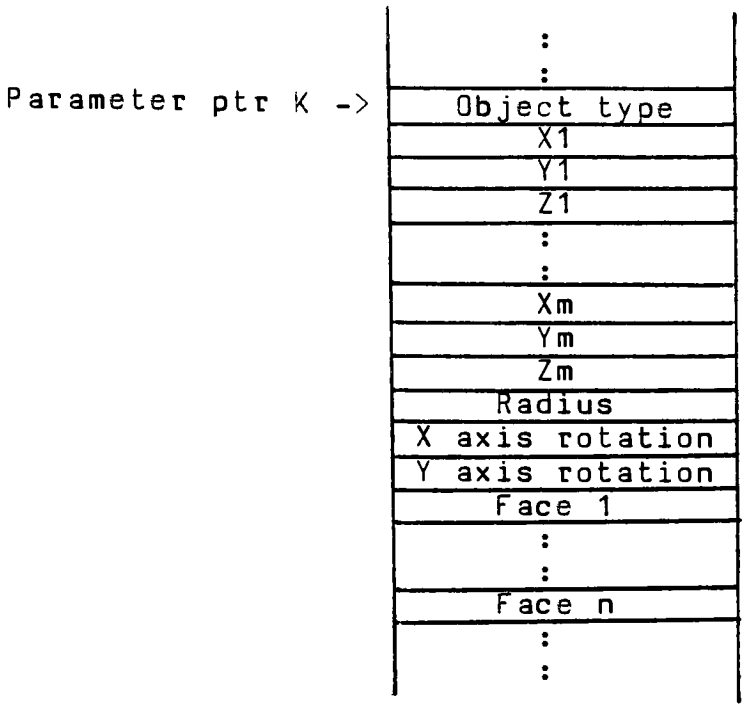


Figure 6

Each left link (major pointer) denotes a descendant and each right link (minor pointer) denotes a sibling. Since the primitive's relationships can be modeled by a binary tree, the primitive array becomes a N X 4 array of the form:

1	Scene name	Major ptr	-	-
2	Primitive name 1	Major ptr 1	Minor ptr 1	Parameter ptr 1
3	Primitive name 2	Major ptr 2	Minor ptr 2	Parameter ptr 2
:	:	:	:	:
:	:	:	:	:
n	Primitive name n	Major ptr n	Minor ptr n	Parameter ptr n

The parameter entries in the parameter vector follow the general form:



Object type is a number from 1 - 12. The number of coordinates, the presence of a radius, x/y axis rotation, are determined by the primitive. Face is a three digit number of the form:

CHO

where:

C - color 0-7
H - shade 1-yes, 0-no
O - solid 1-yes, 0-no

In addition to the primitives array and the parameters vector, information about the camera and light must be stored. The camera's parameters are stored in the VIEWSET vector. This contains:

1. Camera position X,Y,Z.

2. Camera rotations X,Y,Z.

3. Zoom factor

The light's parameters are stored in the LIGHTSET vector and contain:

1. Light position X,Y,Z.

The primitives array, parameters vector, viewset and lightset are the groups of information contained within the data base.

6.3 **Communications Among Modules**

Information concerning the communication among modules can be found in 5.2.

7.0 VERIFICATION AND VALIDATION.

7.1 Test Plan

Since the CAST implementation was developed using top down design and top down coding, is was only natural to use top down testing to debug it. The idea behind top down testing is rather simple. As modules are coded in a top down manner (i.e. the main program is written first, followed by the routines that it calls, followed by the routines that they call, etc.), they are tested in a likewise manner. This requires that when testing a module at level n, the modules at the level n+1 are dummies. In his book on structured programming, Yourdon [22] presents advantages to top down testing. Some of these are:

1. System testing is virtually eliminated.
2. Major interfaces of the program are tested first.
3. It is often much easier to find bugs with a top down testing approach.
4. Top down testing provides a natural test harness for the testing of lower level modules

In addition to the top down testing, I decided to film a short animation. Since the script would include every script command it would not only test the ANIMATE subsystem but it would also give a demonstration of the capabilities of CAST.

7.2 Test Procedures

CAST was tested as it was coded in a top down manner. The scene used for the animation test is described as follows:

SCENE: box-scene SIZE: 32767 BACKGROUND COLOR: black

OBJECT: block1 PART OF: box-scene TYPE: box

```
  POINT #1 LOCATION: 1000.000, 1000.000, 1000.000
  POINT #2 LOCATION: 1010.000, 1000.000, 1000.000
  POINT #3 LOCATION: 1010.000, 1010.000, 1000.000
  POINT #4 LOCATION: 1010.000, 1000.000, 1010.000
  FACE #1 COLOR: red        SOLID: yes   SHADE: no
  FACE #2 COLOR: yellow     SOLID: yes   SHADE: no
  FACE #3 COLOR: blue       SOLID: yes   SHADE: no
  FACE #4 COLOR: green      SOLID: yes   SHADE: no
  FACE #5 COLOR: cyan       SOLID: yes   SHADE: no
  FACE #6 COLOR: magenta    SOLID: yes   SHADE: no

OBJECT: x-axis      PART OF: box-scene      TYPE: line
  POINT #1 LOCATION:  990.000, 997.000, 997.000
  POINT #2 LOCATION: 1000.000, 997.000, 997.000
  COLOR: white

OBJECT y-axis       PART OF: box-scene      TYPE: line
  POINT #1 LOCATION:  990.000,  997.000,  997.000
  POINT #2 LOCATION:  990.000, 1007.000,  997.000
  COLOR: white

OBJECT z-axis       PART OF: box-scene      TYPE: line
  POINT #1 LOCATION:  990.000,  997.000,  997.000
  POINT #2 LOCATION:  990.000,  997.000, 1007.000
  COLOR: white
```

The following is the script used for the test film.

```
  SET CAMERA 1005,1015,950 FRAME 1
  TILT CAMERA -10 FRAME 1,1
  MOVE block1 1010,1000,1000 FRAME 2,121
  MOVE block1 1010,1010,1000 FRAME 122,241
  MOVE block1 1010,1010,1010 FRAME 242,361
  MOVE block1 1000,1000,1000 FRAME 362,481
  ROTATE block1 X,180 FRAME 482,601
  ROTATE block1 Y,180 FRAME 602,721
  ROTATE block1 Z,180 FRAME 722,841
  SCALE block1 2.0 FRAME 842,901
  SCALE block1 0.5 FRAME 902,961
  MOVE CAMERA 995,1015,950 FRAME 962,1081
  MOVE CAMERA 995,995,950 FRAME 1082,1201
  MOVE CAMERA 995,995,940 FRAME 1201,1321
  MOVE CAMERA 1005,1015,950 FRAME 1322,1441
  PAN CAMERA 20 FRAME 1442,1501
  PAN CAMERA -20 FRAME 1502,1561
  TILT CAMERA 20 FRAME 1562,1621
  TILT CAMERA -20 FRAME 1622,1681
  SPIN CAMERA 20 FRAME 1682,1741
  SPIN CAMERA -20 FRAME 1742,1801
  ZOOM CAMERA 3.0 FRAME 1802,1861
```

ZOOM CAMERA 1.0 FRAME 1862,1921

7.3 Test Results

The filmed animation demonstrates the capabilities of the CAST system. When making statements about the robustness of a system, one cannot help but remember Dijkstra's statement "Testing shows the presence, not the absence of bugs." However, the application of top down design and top down testing insures that the CAST system is virtually bug free.

8.0 CONCLUSIONS

8.1 Problems Encountered And Solved

1. Input of circular primitives.

   Quite a bit of time was spent on the method for circular primitive input (e.g., arc, circle, cylinder, cone). The problem lies not with specifying location and size, but with specifying orientation. After investigating several methods (e.g., normal vector, three points on circumference, etc.) I settled on specifying the X and Y axis rotations.

2. Format of the parameters vector.

   I originally intended for the parameters vector to be of integer format. This would restrict the user to positioning primitives on integer coordinates. When animations were run, it became apparent that this format would not work. With integer coordinates, transformations applied to small primitives would produce a jumping motion. In order to provide more resolution, the parameters vector was changed from integer to real.

8.2 Discrepancies And Shortcomings Of The System

1. Speed

   Of the shortcommings mentioned in 3.2, the most noticeable is that of speed. A reduction in turnaround time could be realized by incorporating any of the following:

   1. Convert the device interface from serial RS-232 to parallel (preferably connected to the host's system bus).

2. Record animation output directly on video tape rather than photographic film.

3. Use a more efficient hidden surface algorithm (ideally implemented in hardware).

A change in any one of these areas will have a great impact on turnaround time.

8.3 Lessons Learned

8.3.1 Alternative Approaches For An Improved System -

1. Human interface

The basic idea of a "language driven" animation system is that the user communicates with the system by using a language. CAST receives the language through the keyboard. The speed at which the animator can work will be directly proportional to his typing speed. This could by improved by the use of menus. Rather than type the word DELETE, he would use a locating device to pick the word DELETE from a menu.

2. Database

Much of the written code involves setting up and manipulating a hierarchical data base. A reduction in program size could be realized by making use of the host's database management system (if it has one).

3. Hidden line

The z-buffer algorithm, although simple to implement is very inefficient in both space and time. If one object totally obscures another, the algorithm still requires that the obscured object be broken up into pixels which are then checked for obscurity. A reduction in display time could be obtained by using another algorithm.

8.3.2 Suggestions For Future Extensions -

1. Shadow casting

   Additional realism in the animation could be gained if the primitives could cast shadows on other primitives.

2. Titling

   Additional script commands could be added which would allow the user to create titles and allow him to control title font, color and scrolling.

3. New primitives

   Additional primitives such as a torus and an ellipsoid would provide the animator with greater flexibility when designing his scenes. Adding a new primitive requires specifying its format in the parameters vector and adding a routine to break it up into planes.

4. A simple editor for creating scripts

   Currently, CAST requires the animator to use the host's editor for constructing scripts. Perhaps a simpler editor based on a menu system would be easier to learn and use.

5. Interface output to video tape

   An increase in turnaround time and a reduction in cost could be obtained if one could replace the photographic film with video tape.

6. Camera acceleration/deacceleration

   When the camera or primitive is transformed, the applied transform is held constant. In the real world, one has to contend with object mass and therefore transformations include acceleration when started and deacceleration when stopped.

8.3.3 Related Thesis Topics For The Future -

1. Development of a graphical data base
2. Development of a procedural animation language
3. Methods for human/graphical interaction

9.0 REFERENCES

1. Blasgen, M. W., Gracer, F. (1970). KARMA: A System for Storyboard Animation. Proceedings of the Ninth Annual UAIDE Meeting.

2. Burtnyk, N., Wein, M. (1975). Computer Animation of Free Form Images. Proceedings of the Second Annual Conference on Computer Graphics and Interactive Techniques -- SIGGRAPH '75.

3. Cevam, C. W. (1965). Archaeology of the Cinema. Harcort, New York.

4. Computer Science and Technology Graduate Student Handbook (1977). Rochester Institute of Technology, Rochester, New York.

5. Crow, Franklin C. (1979). Shaded Computer Graphics in the Entertainment Industry. IEEE Tutorial on Computer Graphics.

6. Csuri, Charles A. (1977). 3-D Computer Animation. Advances in Computers. Academic Press, Inc., New York.

7. DeFanti, T. A. (1973). The Graphics Symbiosis System -- An Interactive Mini-Computer Animation Graphics Language Designed for Habitability and Extensibility. Ph.D. Thesis, Department of Computer and Information Science, Ohio State University, Columbus Ohio.

8. Donovan, John J. (1972). Systems Programming. McGraw-Hill, Inc., New York.

9. Guide to the CST Thesis (1982). Rochester Institute of Technology, Rochester, New York.

10. Hackathorn, Ronald J. (1977). ANIMA II: A 3-D Color Animation System. Proceedings of the Fourth Annual Conference on Computer Graphics and Interactive Techniques -- SIGGRAPH '77.

11. Knowlton, Kenneth C. (1968). Computer-Animated Movies. Emerging Concepts in Computer Graphics. W. A. Benjamin, Inc., New York.

12. Knuth, Donald E. (1975). Fundamental Algorithms. Addison-Wesley Publishing Co., Reading, Massachusetts.

13. Levoy, Marc (1977). A Color Animation System Based on the Multiplane Technique. Proceedings fo the Fourth Annual Conference on Computer Graphics and Interactive Techniques -- SIGGRAPH '77.

14. Newman, William M., Sproull, Robert F. (1979). Principles of Interactive Computer Graphics. McGraw-Hill, Inc., New York.

15. Raster Graphics Handbook (1980). Conrac Division, Conrac Corporation, Covina, California.

16. Schachter, Bruce J. (1981). Computer Image Generation for Flight Simulation. IEEE Computer Graphics and Applications, Vol. 1, No. 4.

17. Schebor, Frederick S. (1982). CAST: A System for Color Animation and Scene Transformation. Report to the Thesis Committee, Rochester Institute of Technology, Rochester, New York.

18. Scott, John T. (1979). Computer Films for Research. Physics Today, American Institute of Physics.

19. Shoup, Richand G. (1979). Color Table Animation. Proceedings of the Sixth Annual Conference on Computer Graphics and Interactive Techniques -- SIGGRAPH '79.

20. Sutherland, I. E., Sproull, R. F., Schumacker, R. A. (1974). A Characterization of Ten Hidden-Surface Algorithms. ACM Computing Surveys, Vol. 6, No. 1.

21. Yarbrough, L. D. (1969). A Nonprocedural Language for Computer Animation. Pertinent Concepts in Computer Graphics. University of Illinois Press, Urbana, Illinois.

22. Yourdon, Edward (1975). Techniques of Program Structure and Design. Prentice-Hall, Inc., englewook Cliffs, New Jersey.

10.0 APPENDICES

10.1 User Manual

10.1.1 Introduction -

CAST is an acronym for Color Animation and Scene Transformation. The objective of this system is to give non-computing personnel the ability to create still and animated pictures with the aid of a digital computer.

Data generation and animation language are two key components that comprise an animation system. The CAST system incorporates an interactive dialogue to assist the user in generating and/or editing three dimensional scenes, and an animation language which uses a simple script like syntax to describe parallel motion.

The third key component of an animation system is the display. The CAST system incorporates hidden surface elimination and surface shading necessary to produce realistic images. The output is in "device independent" form so that it may be adapted to virtually any type of display.

10.1.2 **System Overview -**

The CAST system is built on three key components:

1. Data generation
2. Display
3. Animation

Data Generation

Data generation is the process which enables the animator to input the objects required for the animation. In order to be effective, data generation must be built around a method that will permit the user to create three dimensional objects with relative ease. Many systems require the operator to input a large number of point locations that lie on the surface of the object they describe. Although flexible and precise, it is a very time consuming process. The method used by CAST involves giving the user a standard set of geometric primitives. The process of constructing a three dimensional scene amounts to defining the location (in the Cartesian coordinate system), the dimensions and orientation of the primitives. These primitives are chosen from a library of twelve types and are the basic building blocks of the CAST system.

The primitives include:

1. Point
2. Line
3. Triangle
4. Plane
5. Arc

6. Circle

7. Box

8. Wedge

9. Pyramid

10. Sphere

11. Cylinder

12. Cone

For example, suppose the animator wished to create a scene which contained a house. A simple house could be described as a wedge sitting on top of a box. A lawn would be modeled as a green plane positioned under the box. Doors and windows would be rectangular planes attached to the sides of the box.

One problem is apparent with this type of structure. Suppose the user wished to apply a transformation (e.g., rotation) to the house during the animation. Normally this would require that he transform all of the objects (box, wedge and plane) separately but in step with one another. CAST solves this problem by giving the user a joining operator. With this operator, primitives may be logically joined together and treated as one object. The operator requires that one object in a group assumes the role of "parent" and the others are "descendants". Any operation performed on the parent will affect all of its descendants. Any operation performed on a descendant will not affect its parent or siblings.

All primitives are described by a list of attributes. These attributes consist of:

1. Object name - 1 to 10 characters supplied by the user.

2. Defining points - Points which define the size and orientation of the object.

3. Object color

4. Object presentation - Either solid or wire-frame

5. Object shade - A flag indication whether the object is to be displayed with shading.

If an object contains more than one face, each face can have its own color, presentation and shade. Object presentation and shade attributes do not apply to points and lines.

CAST incorporates an interactive BUILD/EDIT subsystem to aid the animator in creating scenes. The BUILD subsystem is used for creating new scenes and the EDIT subsystem for editing previously built scenes. When the BUILD subsystem is entered, the user is prompted to enter the scene name, the background color, and the stage size. Colors include red, yellow, green, cyan, blue, magenta, black and white. The stage size is the distance from the viewing location at which objects beyond this point will not be displayed. Providing the animator with a finite stage allows him to "hide" objects that are not required on stage at some point in the animation.

Once the initial scene information is entered, the user can then add objects to the scene. When adding objects, the user is prompted for the object name, the name of its parent, point and face information. The size and orientation of each primitive is described by a set of points. The operator is prompted to enter the coordinates of each point in the set. Once the points have been entered, he is prompted for the face information. After a primitive has been entered, any of these attributes may be changed through the use of the MODIFY command. If a primitive is not needed it may be removed from the scene by using the DELETE command. DELETE makes use of the parent-descendant relationship. If a parent is deleted, all of its descendants are also deleted. When the user is satisfied with the scene, he records it with the SAVE command. This command creates a disk file in which the scene is saved. The scene file is used as input for the EDIT or ANIMATE subsystem.

In addition to primitives, the user has control over a CAMERA and a LIGHT. It is through the camera that the user views his scene. Although conceptual in nature, it behaves in the same manner that an ordinary camera would behave. The user can specify a camera position, rotations and zoom factor. The animator is given control over the light source the illuminates a scene. He can create variations in primitive shading by moving the light about the stage.

Throughout the data generation process, the user is always prompted for information, whether it be a command or some primitive attribute. If he becomes lost at any point,

typing HELP will tell him where he is and what he is expected to enter. In addition, error checking on user input is done immediately after it is entered.

Display

Display is the process of giving visual feedback of the animation to the user and to record the final animation sequence. In order to produce realistic images, CAST incorporates hidden surface elimination and surface shading. In addition, the output is in "device independent" form.

Hidden Surface Removal

The removal of the hidden parts from a scene is one of the most active areas in computer graphics. The algorithm used by CAST to remove hidden surfaces works in image space and is known as the Z-Buffer or Depth Buffer algorithm. The display process involves breaking primitives up into planes which are broken up into lines by a scan line algorithm. The lines are broken up into pixels which are ready to enter into the z-buffer. Pixel depth is determined by linear and planar equations.

Shading

CAST incorporates object shading in order to produce realistic images. An object's shade is determined by three factors:

1. The amount of light falling on the object.
2. The object's orientation with respect to the light.
3. The objects reflectance.

Planes are shaded according to the equation:

$$E = (R \cos A) * I$$

where:

E - Energy transmitted from the plane.
R - The coefficient of reflectance for the plane (0.0 - 1.0)

A - The angle of incidence between the plane and light source.
I - The energy arriving from the light source.

Although the distance from the light source to an object may vary, CAST maintains that the amount of light falling on the object will be constant. CAST also assumes that the objects are 100% reflective and that the light source is reflected uniformly in all directions (diffuse reflection).

Device Independence

"Device independence" is a methodology whereby software is isolated from machine peculiarities. CAST interfaces to a particular device through a set of low level routines called a "device driver". The driver contains routines to go into and out of graphics mode, set color, and draw a point or line. Device independence permits interfacing CAST to virtually any type of graphic display by replacing simple low level routines rather than complicated high level routines.

Animation

Animation is the process of "giving life" to the objects which were defined during data generation. This is done by linking motions to the objects in the scene. These motions imitate actions of the physical world. The motions can involve a change in an objects position, orientation or size.

Traditional methods of filming link motions to objects by the use of a script. The script states which objects will have which motions, at which times. Note that the script is "key frame" in nature. For example a script might state that an actor walk from point A to point B in one minute. The script has stated the object (actor), the motion (walk), the motion limits (A,B) and the duration (1 minute). It is up to the actor to construct the in-between motions i.e., the process of putting one foot ahead of the other. The concept of an animation script is a key component of CAST.

In order to be read and executed under computer control, the script should follow some type of standard syntax. The script is broken into commands with one command

per line. Each command has the form:

transformation, object name, transformation limit, time period

CAST transformations include SET, MOVE, ROTATE and SCALE. The object name is that which was supplied during data generation. The transformation limit may be; a coordinate (SET, MOVE), an angle (ROTATE) or a factor (SCALE). The time period is specified by the number of frames during which the transformation will be in effect. Since the standard animation film speed is 24 frames per second, a transformation applied over 24 frames will be one second long. The script command:

```
SET box1 1000,1000,1000 FRAME 1
```

will position the object called box1 at location 1000,1000,1000 at frame number 1. The script command:

```
ROTATE box1 X,180 FRAME 1,10
```

will rotate the object called box1, 180 degrees about its X axis from frames 1 though 10. Remember that during data generation, objects may be part of other objects. In the above examples, if box2 and box3 were PART OF box1, they would be transformed by the same transform applied to box1. This is where CAST displays its power. Complex transformations need not be the result of applying transformations to every single object, but by applying simple transformations to the parents.

In order to be effective, the animation script must not only control a scene and/or the objects in a scene, but it must give full control over the camera that views the scene. Note: this is not the camera that actually films the animation, but it is the point at which the scene is viewed. In the CAST system, the camera although conceptual, behaves in the same manner that an ordinary camera would behave. The animator can make use of commands such as SET, MOVE, TILT, PAN, SPIN and ZOOM. The system gives the animator the same camera control that he would have in the physical world. The one exception to this is focus and aperture. Since the depth of field is infinite, there is no need for a focus/aperture control.

Finally the animator has control over the light source that illuminates the scene. He is provided with commands which can SET or MOVE the light source during the animation.

10.1.3 Architectural Design -

Figure 7 shows the architectural diagram of CAST.

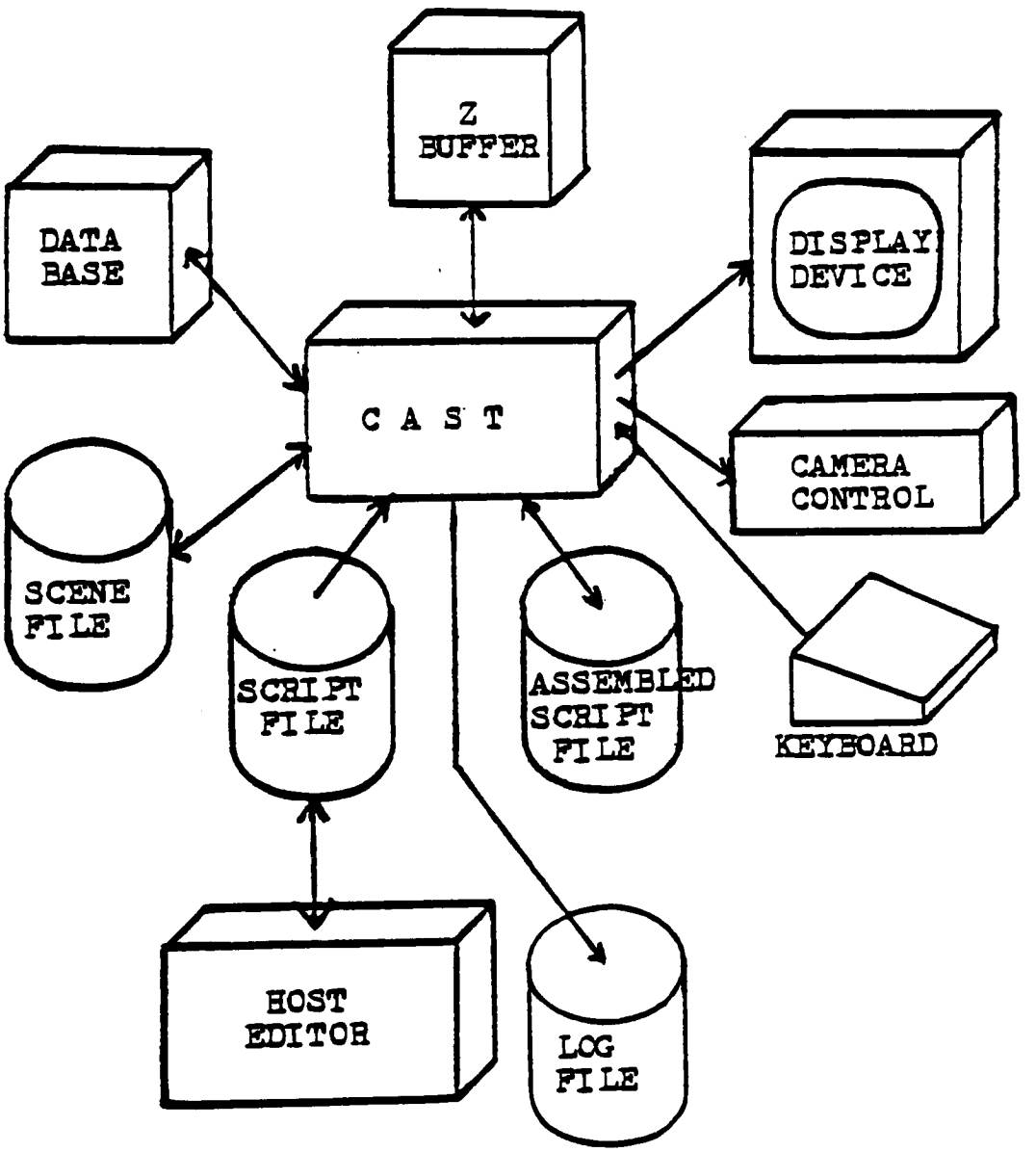


Figure 7

CAST reads input and writes output to the following areas:

1. Keyboard - All user input is received from this device.

2. Display - All user output and graphical output is displayed here.

3. Database - This is where the scene is held during BUILD/EDIT and ANIMATE. Also included are the camera and light parameters. During execution, the database resides in memory.

4. Scene File - The scene is copied from the data structures into this file as a result of the SAVE command while in BUILD/EDIT or the SAVE FRAME command while in ANIMATE.

5. Z-Buffer - Created by the BUILD/EDIT and ANIMATE subsystems. It is a three dimensional array that holds graphical output. It resides in memory during CAST execution.

6. Script File - When entering the ANIMATE subsystem, CAST gets the script from the script file. This file is generated by the user by running the host's text editor.

7. Assembled Script File - As the script is assembled, it is written into this file. This file is used for both input and output during animation.

8. Log File - Keeps a record of the frames produced during an animation.

9. Camera Control - On command from the host, this device will trigger the camera.

10.1.4 System Operation -

Startup

CAST runs on a Digital Equipment Corporation VAX 11/780 computer. In addition, CAST requires a UNIX operating system. After obtaining a VAX account, log on the VAX using the account name and pass word. Currently there are three versions of CAST. Each version outputs images to a different graphic terminal. Use the following table to determine which program to run.

Terminal	Program Name
Chromatics GC 1999	castchro
Ramtek 6211	castram
DEC Gigi	castgigi

To run the program simply type the appropriate name. When CAST starts running you will see the message:

CAST Animation System Version x.x mmddyy

x.x is the current version number and mmddyy is the date of the last version change. At this point you are in the CAST monitor.

CAST Monitor

The monitor is the highest level of the CAST system. Through it, you call up the BUILD/EDIT and ANIMATE subsystems. The monitor is identified by the prompt <CAST>. The presence of this prompt indicates that CAST is waiting for you the type a monitor level command.

The monitor commands are:

1. animate - Run the animation subsystem.

2. **b**uild - Run the BUILD subsystem.

3. **e**dit - Run the EDIT subsystem.

4. **h**elp - Display the monitor help file.

5. **stop** - Stop the CAST system and return to the host operating system.

Note: B**olding** indicates the legal abbreviation. For example, to run the BUILD subsystem type build or b.

BUILD Subsystem

The BUILD subsystem allows you to create scenes. When the BUILD subsystem starts you are prompted to

Enter the scene name:

In the current configuration, the scene name contains one to 10 alphanumeric characters. The first character must be a letter. If there currently exists a scene with the same name, that scene will be erased and the current one will take its place.

Once the scene name has been entered, you are prompted to

Enter the stage size:

In the current configuration, the stage size is a number from 1 to 131067. If the distance from an object to the camera is greater than the stage size, it will not be displayed. Positioning objects beyond the stage allows you to hide them until they are needed in the animation.

After the stage size has been entered, you are prompted to

Enter the background color:

The background can be thought of as a colored curtain which is located at the stage perimeters. The permissible colors are red, green, blue, yellow, cyan, magenta, black and white. When this is done you are given the BUILD prompt <BUILD>. At this point you can enter any of the BUILD/EDIT commands. See: **Summary of BUILD/EDIT Commands**

EDIT Subsystem

The EDIT subsystem allows you to modify a previously built scene. When the EDIT subsystem starts, you are prompted to

Enter the scene name:

You respond by typing the name of a previously built scene. Once this is done you are given the EDIT prompt <EDIT>. You can now enter any of the BUILD/EDIT commands

Summary of BUILD/EDIT Commands

The following is a list of the commands and a brief description of their result.

1. add - Add a primitive to a scene.
2. delete - Delete a primitive from a scene.
3. display full - Display the scene using the full terminal capability i.e. color, shading, etc.
4. display line - Display a wire-frame representation of the scene.
5. help - Display the BUILD/EDIT help file.
6. modify - Modify a primitive in a scene.
7. print extended - Print a detailed list of the primitives in a scene.
8. print short - Print a condensed list of the primitives in a scene.
9. print standard - Print a list of the primitives in a scene.
10. save - Save a scene.
11. setlight - Set the light position.
12. setview - Set the camera's parameters.

13. **sto**p - Exit the BUILD/EDIT subsystem and return to the monitor.

Description of BUILD/EDIT Commands

add

Description: This command allows you to add primitives to a scene. When adding a primitive, you are prompted for the name, parent name, primitive type, location/orientation, color(s), solid/shade.

delete

Description: Delete allows you to remove a primitive or primitives from a scene. When deleting a primitive, you are prompted for the name. If there are any primitives which are part of the deleted primitive, they are also deleted.

display **f**ull

Description: When the display full command is executed, the current scene will be displayed with hidden surface processing and surface shading. See figure 9. After the scene is displayed, the terminal bell will sound. You continue by hitting RETURN.

display **l**ine

Description: This command displays the current scene using a wire-frame representation. See figure 9. Planes are represented by their outlines which are the same color as the plane. Hidden surfaces are not removed. Since less computation is needed, it is much faster to execute a display line than to execute a display full.

help

Description: Entering this command will give you a list and brief description of the BUILD/EDIT commands.

modify

Description: Modify allows you to change the attributes of a primitive. You are first prompted for the primitive name. After it is entered, it will be displayed again. If you type in a different name followed by hitting RETURN, that new name will replace the old name. If you just hit RETURN, the name is unchanged. This is repeated for the parent name, location/orientation, color(s), and solid/shade. Note that you cannot modify the primitive type.

print

Description: The print commands allow you to obtain a list of the primitives that are in the current scene. There are three versions of this command: **p**rint **s**hort, **p**rint **r**egular and **p**rint **e**xtended.

1. Short - List the names of the scene, primitives and the primitives parents.

2. Regular - Same as short but include a list of the primitive's attributes.

3. Extended - Same as regular but include primitive addresses and a data structure dump.

For example, suppose a scene called galaxy had three point primitives e.ursamaj, dubhe and n.ursamag in it. The output of print would be as follows:

print **s**hort

```
  SCENE: galaxy
-------------------------------------------------------

  OBJECT: e.ursamaj    PART OF: galaxy
  OBJECT: dubhe        PART OF: galaxy
  OBJECT: n.ursamaj    PART OF: galaxy
```

print regular

```
  SCENE: galaxy
-------------------------------------------------------

  OBJECT: e.ursamaj    PART OF: galaxy    TYPE: point
  POINT LOCATION: 3861.770, 4639.180, 18792.850
  COLOR: white

  OBJECT: dubhe        PART OF: galaxy    TYPE: point
                         :
                         :
                         :
```

print extended

```
  SCENE: galaxy    SIZE: 13106    BACKGROUND COLOR: black
-------------------------------------------------------

  OBJECT: e.ursamaj    PART OF: galaxy    TYPE: point
  NODE ADDRESS: 4    VECTOR ADDRESS: 4
  POINT LOCATION: 3861.770, 4639.180, 18792.850
  COLOR: white

  OBJECT: dubhe        PART OF: galaxy    TYPE: point
                         :
                         :
                         :


  VIEW SET:
      4135.260 4701.810 18374.840 .000 .000 .000 1.000

      LOC    MA    MI    NODE         OVP      OV
     ---------------------------------------------
       0      3     3                   21      94
       1      0     0                13106    .000
       2      0     0                    0    .000
       3      4    -1    galaxy         -1    .000
       4     -1     5    e.ursamaj       4    .000
       5                 :
                         :
                         :
```

save

Description: When executed, this command stores the current scene onto disk storage.

setlight

Description: This command allows you to position the light source within a scene. You are given the current location of the light. Entering a new location followed by hitting RETURN, will up date the light's position. A RETURN by itself will leave the position unchanged.

setview

Description: This command allows you to position the camera within a scene. You are given the current location and the current rotations. See figure 11. Entering numbers followed by hitting RETURN will update them. Hitting just RETURN, will leave them unchanged.

stop

Description: When this command is entered, the BUILD or EDIT subsystem stops and you are returned to the monitor.

Parameter Input

When working in the BUILD/EDIT subsystem, the animator will be prompted to enter various types of parameters. These include:

1. Object name - one to ten alphanumeric characters.
2. Object type - point, line, triangle, plane, arc, circle, wedge, box, pyramid, sphere, cylinder and cone. See **Object Types**.
3. Point location - three numbers specifying X, Y and Z coordinates. Each number can contain four digits to the left of the decimal point and three digits to the right. The first location entered is called the base point.
4. Radius - one number specifying a radius.
5. Degrees - two numbers specifying the start and end angles for an arc.
6. Rotations - Numbers specifying X and Y axis rotation angles. See figure 10.

7. Color - black, blue, green, cyan, red, magenta, yellow and white.

8. Solid - yes or no.

9. Shade - yes or no.

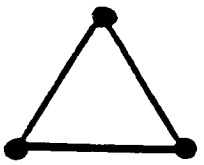
10. Zoom factor - A number between 0 and 100 that specifies the current lens magnification.

Object Types

The following diagrams represent the twelve primitives supported by CAST. The numbers indicate the order in which the locations are entered. The first location entered is called the base point.

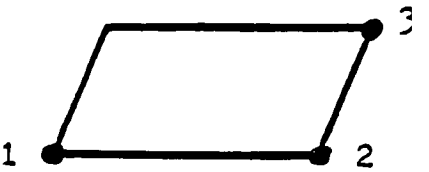
POINT

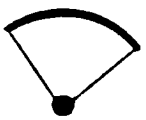
LINE



TRIANGLE

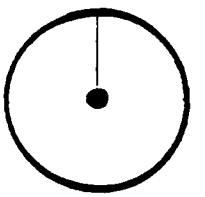
PLANE

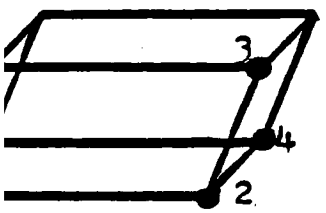




ARC

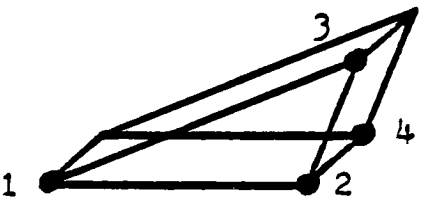
CIRCLE

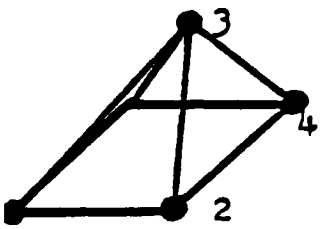




BOX

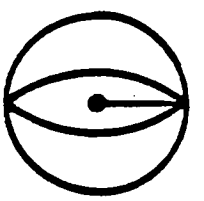
WEDGE

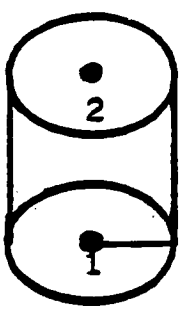




PYRAMID

SPHERE





CYLINDER

CONE

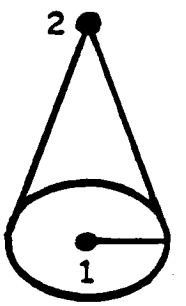


Figure 8

Animate Subsystem

Upon entering the animate subsystem, you are prompted to

Enter the scene name:

After entering the name of a previously built scene, you are then prompted to

Enter the script name:

This is the name of a file containing the animation commands. See **Script Commands** for more information. The script then goes through the script assembler. The assembler checks for correct command syntax, i.e. spelling, and checks that objects in the script are defined in the scene. As each command is checked, it is displayed on the terminal. Any errors in the command will be displayed underneath it. When the assembler has checked the last line in the script, you are given the message

Assembly Complete xx ERRORS

If xx is not zero, (i.e. there are errors) you are returned the the CAST monitor. If there are no errors you are prompted:

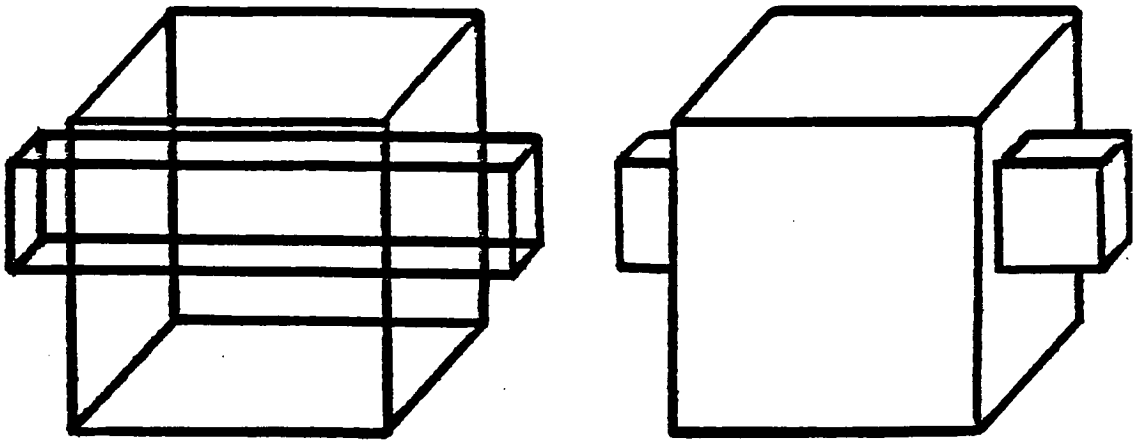
Run the animation?

Any response other than yes (or y) will return you to the CAST monitor.

If you answered yes (or y) then you are prompted to

Enter the display type:

The type refers to the presentation of the scene on the graphic terminal. The allowable responses to this prompt are 'line' or 'full'. A response of 'line' will draw the scenes using a wire-frame representation. With this type of presentation, only the outlines of the surfaces are drawn. The color of the outline will be the color of the surface. Hidden surfaces are not removed and there is no shading. See figure 9. A response of 'full' indicates that the scenes will be displayed with hidden surfaces removed, surfaces filled in and shaded (if stated during BUILD/EDIT).



Line Presentation Full Presentation

Figure 9

Since a wire-frame representation can be computed and drawn much faster than with the full option, the line mode is useful for debugging the scene and script.

When the presentation prompt has been answered, you are then given the message:

When the equipment is set up, hit RETURN

This allows you to make any additional connections (i.e. film recorder) necessary to record the animation. When the RETURN key is hit, the animation starts. After it finishes, you are returned to the CAST monitor.

Finally, when the animation starts, a file, named animat.log, is generated. As each frame is displayed, a line indicating the frame number, display time and date are written to this file. If, due to a malfunction, the animation should stop, this file may be listed to determine the last frame displayed before the malfunction.

Summary of Script Commands

Object commands:

1. SET object_name x,y,z FRAME n
2. MOVE object_name x,y,z,FRAME n,m

3. ROTATE object_name axis,degrees FRAME n,m

4. SCALE object_name factor FRAME n,m

Camera Commands:

1. SET CAMERA x,y,z FRAME n

2. MOVE CAMERA x,y,z FRAME n,m

3. PAN CAMERA degrees FRAME n,m

4. TILT CAMERA degrees FRAME n,m

5. SPIN CAMERA degrees FRAME n,m

6. ZOOM CAMERA factor FRAME n,m

Light Commands:

1. SET LIGHT x,y,z FRAME n

2. MOVE LIGHT x,y,z FRAME n,m

Other Commands

1. SAVE FRAME n

2. * Comment

Description of Script Commands

SET

Format: SET object_name x,y,z FRAME n

Description: Position the base point of object_name to location x,y,z at frame n. If the object_name = CAMERA, then the camera will be repositioned to x,y,z. If the object_name = LIGHT, then the light will be repositioned to x,y,z.

Examples: SET box_1 1000,1000,1000 FRAME 105
SET LIGHT 250,175,510 FRAME 21

MOVE

Format: MOVE object_name x,y,z FRAME n,m

Description: Move the base point of object_name from its current location to location x,y,z from frames n through m. If object_name = CAMERA, then the camera will be moved. If object_name = LIGHT, then the light will be moved.

Examples: MOVE box_1 3000.01,100,250.1 FRAME 37,47
MOVE CAMERA 10,10,10 FRAME 1325,1425

ROTATE

Format: ROTATE object_name axis,degrees FRAME n,m

Description: Rotate object_name about its axis by degrees from frame n through m. The rotation angle is measured clockwise about the origin when looking at the origin from a point on the + side of the axis. See figure 10.

Example: ROTATE box_1 X,-10.0 FRAME 64,78

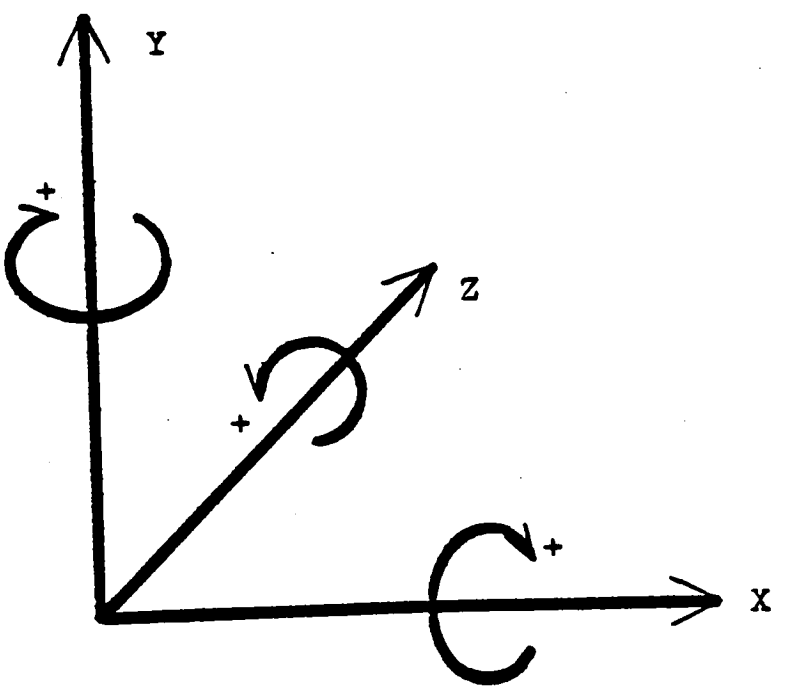


Figure 10 - Object rotation axis

SCALE

Format:	SCALE object_name factor FRAME n,m
Description:	Scale object_name by factor from frames n through m. If factor is > 1.0 then the object size will increase. If factor is < 1.0 then the object size will decrease.
Example:	SCALE box_1 2.25 FRAME 5,10

PAN

Format:	PAN CAMERA degrees FRAME n,m
Description:	Rotate the camera about its Y axis by degrees from frame n through m. See figure 11.
Example:	PAN CAMERA 10.55 FRAME 24,48

TILT

Format:	TILT CAMERA degrees FRAME n,m
Description:	Rotate the camera about its X axis by degrees from frame n through m. See figure 11.
Example:	TILT CAMERA -24.6 FRAME 64,128

SPIN

Format:	SPIN CAMERA degrees FRAME n,m
Description:	Rotate the camera about its Z axis by degrees from frame n through m. See figure 11.
Example:	SPIN CAMERA 5 FRAME 32,32

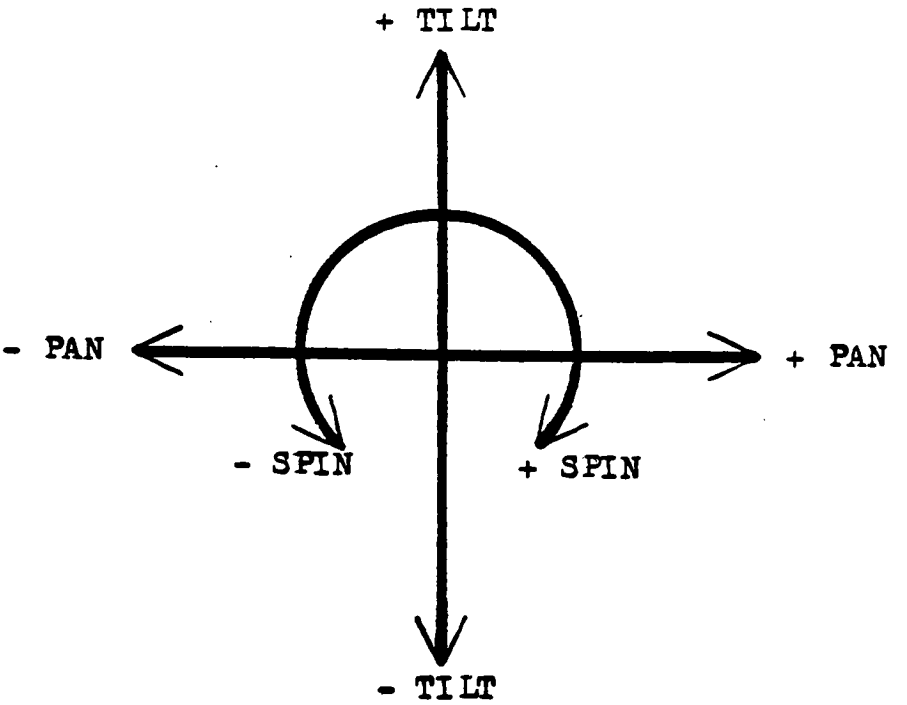


Figure 11 - Camera Rotations

ZOOM

Format:	ZOOM CAMERA factor FRAME n,m
Description:	Change the focal length of the camera's lens by factor from frames n through m. If factor > 1.0, then the focal length will increase. If factor < 1.0, then the focal length will decrease.
Example:	ZOOM CAMERA .05 FRAME 1,24

SAVE

Format:	SAVE FRAME n
Description:	When the nth frame has been generated, it will be saved in a file as a scene. This scene may te examined using the EDIT subsystem. The scene name is frameXXXXX where XXXXX is the frame number.
Example:	SAVE FRAME 105

COMMENT

Format:	* text
Description:	The * command allows comments to be placed in the script. The letters following the * are ignored.
Example:	* You should always comment * your script!

Script File

The script resides in a file which must be read, assembled and executed under program control. This file consists of lines with one animation command per line. The line may be any length but only the 1st 80 characters are read. The script file is produced by the host's text editor. Within the file, script commands must be ordered such that the terminating frame numbers are equal or increasing. For example:

```
* This is legal since 64 >= 56
MOVE box_1 10,10,10 FRAMES 37,56
ZOOM CAMERA 2.0 FRAMES 24,64

* This is illegal since 56 < 64
MOVE box_1 10,10,10 FRAMES 24,64
ZOOM CAMERA 2.0 FRAMES 37,56
```

10.2 Film Exposure

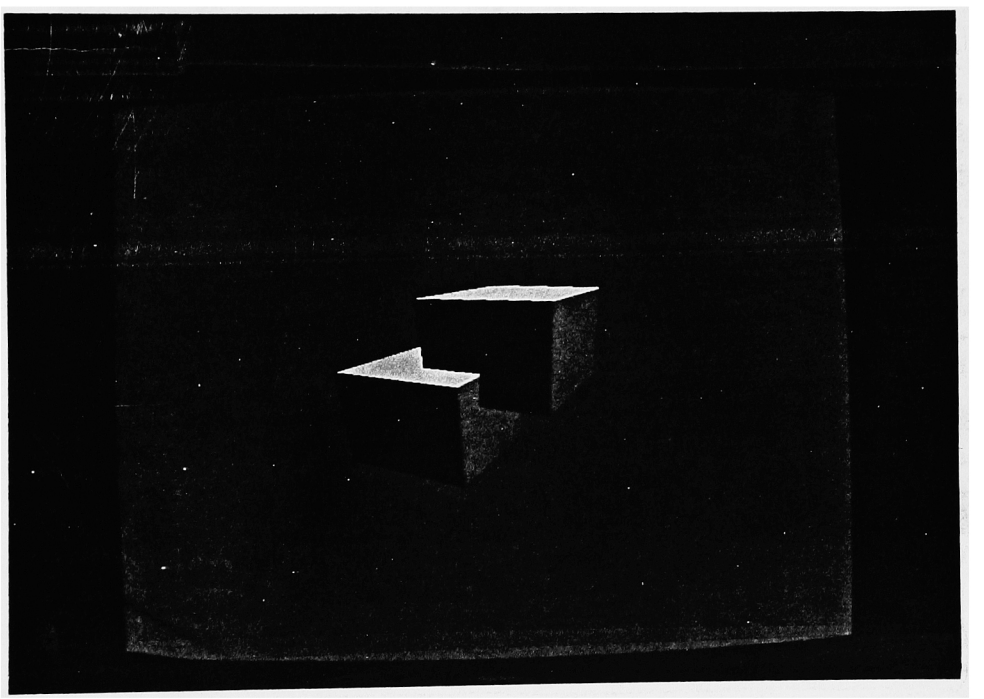
The following table can be used as a guide in determining proper film exposure.

ASA	Seconds	F-Stop
25	1	5.6
50	1	8
64	1/2	5.6
125	1/2	8
400	1/8	8

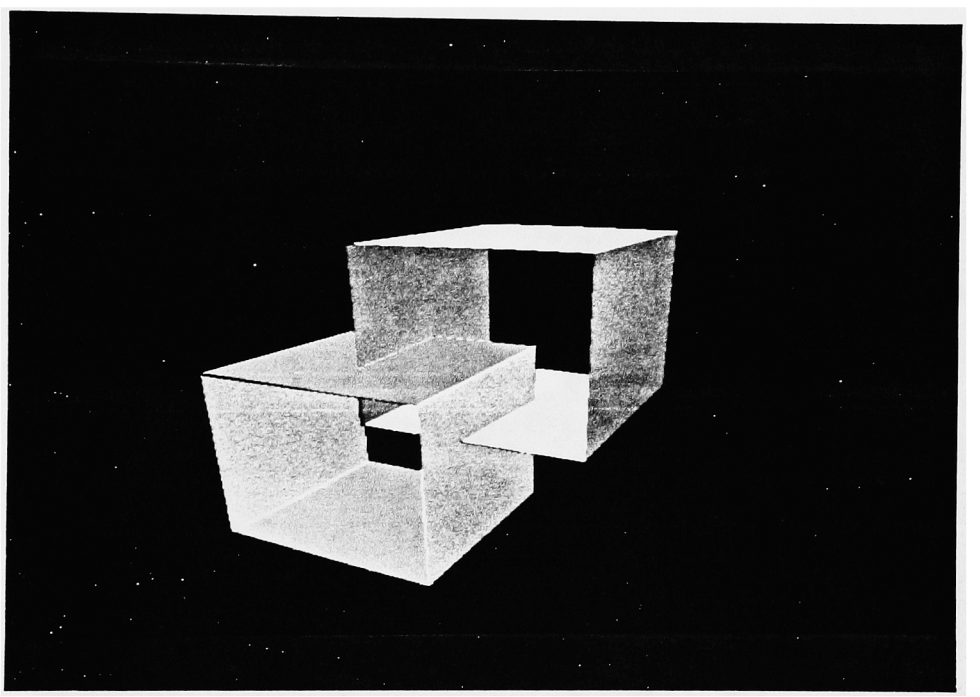
In all cases, when photographing a video monitor, the shutter speed should be slower than 1/30 sec.

10.3 Sample Images

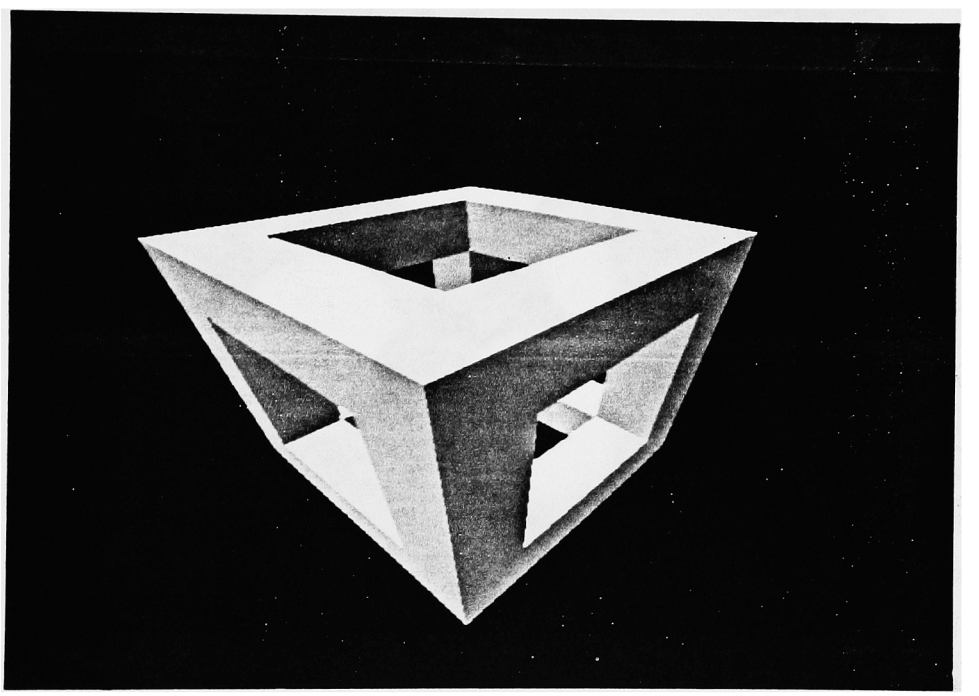
The following five photographs are samples of CAST image output.



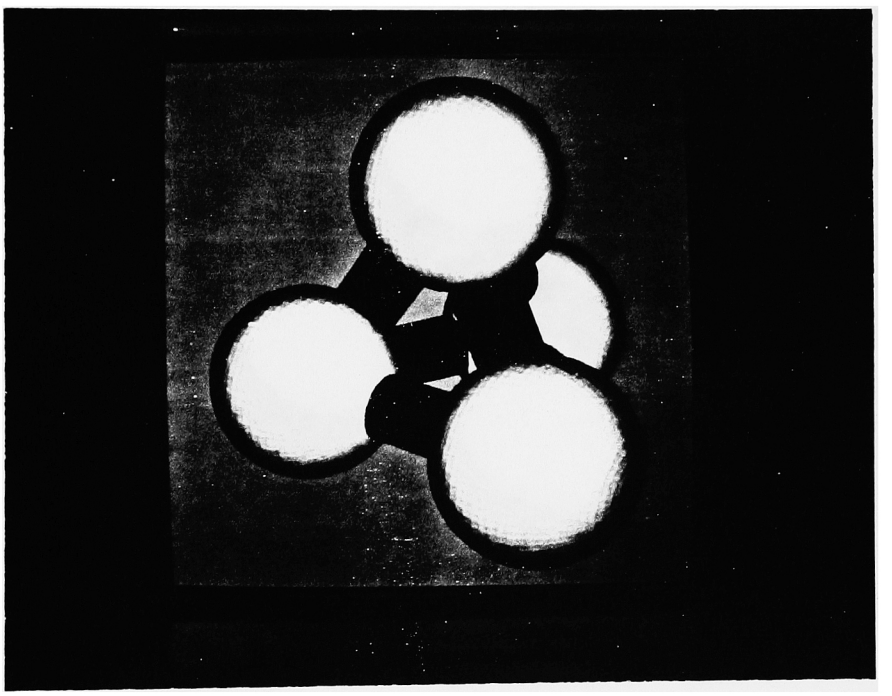
Two intersecting boxes.



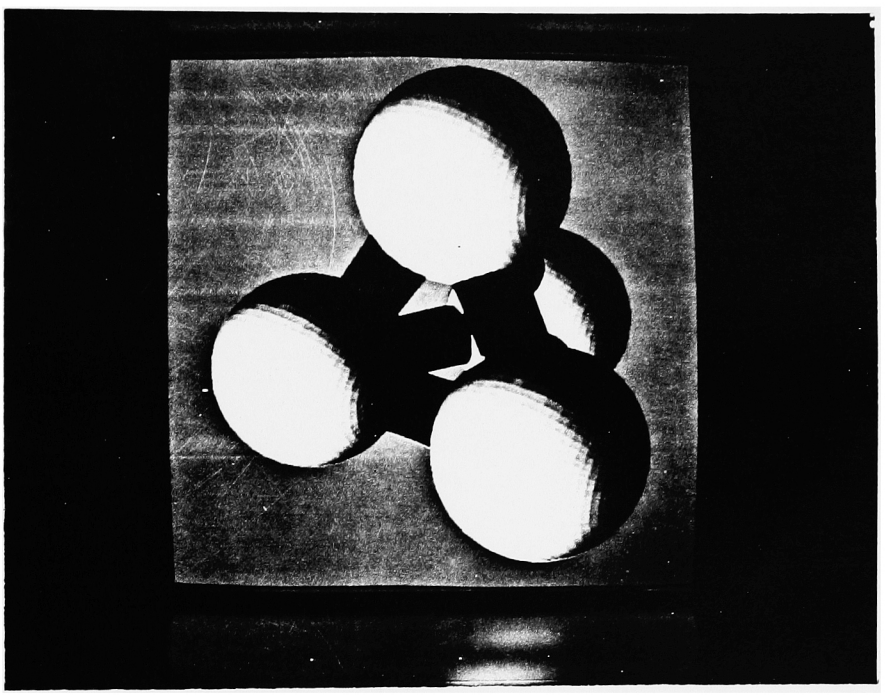
Two intersecting boxes. Faces 1 & 3 are not solid.
Zoom has been increased.



A wire-frame box constructed from 12 boxes.



A tetrahedron formed with cylinders and spheres.
The light source is located at the camera position.



A tetrahedron formed with cylinders and spheres.
Ther light source is located approximately 30 degrees
to the left of the camera position.

10.4 Program Listings

Due to the large amount of software needed to implement the CAST system (approximately 10,000 lines of both code and comments), listings could not be conveniently included in this thesis. The CAST system is in the public domain. Any questions pertaining to CAST should be directed to the School of Computer Science and Technology, Rochester Institute of Technology.