Ronald J. Hackathorn

COMPUTER GRAPHICS RESEARCH GROUP THE OHIO STATE UNIVERSITY

#### ABSTRACT

An animation software system has been developed at The Computer Graphics Research Group which allows a person with no computer background to develop an animation idea into a finished color video product which may be seen and recorded in real time. The animation may include complex polyhedra forming words, sentences, plants, animals and other creatures. The animation system, called Anima II, has as its three basic parts: a data generation routine used to make colored, three-dimensional objects, an animation language with a simple script-like syntax used to describe parallel motion and display transformations in a flexible, scheduled environment, the Myers algorithmused in the visible surface and raster scan calculations for the color display. This paper discusses the requirements, the problems, and the trade-offs of such a system. An overview of research in the area is given as well as the design and implementation highlights of the Anima II system.

## 1. Introduction

During the past several years, films from the University of Utah (16), General Electric Corp. (17) and by N. Max (18), illustrate that the ability to produce 3-D shaded object animation has been a significant addition to the field of computer animation. Max's comment about his film, "Sphere Eversion," describes the basic feeling towards this type of animation: "The film produces a visualization which could not have been achieved in any other medium, and could never have been animated by hand." (26)

A 3-D animation system which uses a visible surface algorithm to calculate the final displayed image must deal with severe time-space considerations resulting from the increased complexity of both the data and the data handling algorithms, through all phases of the system. Traditionally, shaded object animation while producing high quality has been a difficult, slow and expensive process as a result of implementationaltrade-offs among these various considerations.

An animation software system has been developed by the Computer Graphics Research Group as an

\*This research was supported by the National Science Foundation under Grant No. DCR 74-00768.

Permission to make digital or hard copies of part or all of this work or personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

Siggraph '77, July 20-22 San Jose, California

attempt to maximize the trade-offs involved in 3-D color animation. The goal has been to achieve the capability and image quality necessary for complex animation and, yet, maintain the total system efficiency necessary for a production animation environment.

Anima II is a computer animation system designed for the production of color, three-dimensional video tapes. It is aimed at the animator, educator and artist who requires anything from a high volume of short color sequences for teaching purposes, to realistic key frame animation involving complex color objects and precisely timed lifelike movements. The Anima II system provides an efficient environment for the creation, animation and real-time playback display of color-shaded polyhedra. The video output is directly connected to video recording equipment and a standard color television set.

2. Background

Before discussing previous research in the area of 3-D shaded animation systems it is important to briefly discuss the requirements, the problems and the trade-offs which accompany the design and implementation of such a system.

## 2.1 <u>Requirements</u>

System and user requirements for 3-D shaded animation can be classified into three factors determining overall system performance.

2.1.1 <u>Capabilities</u> - A shaded animation system can be viewed as having three separate capabilities. Each has a unique function within the system and each has different problems.

- Data Generation is the process of constructing a computer model representing the three-dimensional object or form that is to be animated. The type of data to be generated is determined by the type of visible surface algorithmused. Essentially, polygon-based algorithms need planar polygons while parametric surface algorithms need high-order patch equations. There have been many approaches to inputting of 3-D polyhedra. These include dual data tablet digitizing (32), single data tablet (23,27), and geometric modeling (4,8,13,29).
- <u>Animation</u> is the process of "giving life" to

54

the generated objects by specifying motions which imitate the actions of the physical world. These motions involve changes to an object's position, orientation (rotation), size and shape. Also the concepts of acceleration, deceleration and "path-following" are included as motion descriptions. The animator controls the motions of an object through a program or "script" written in the syntax of the system's language. Key frame techniques implemented in several 2-D/3-D line animation systems (3,6,19, 36), notably by the Film Board of Canada (34, 35), have proven a most effective means for specifying the motion dynamics (movement through time and space) of complex animation. The central notion of key frame animation is that an action of an object will change "from" some spatial state "to" a new state and that this action will range "from" some time frame within the sequence "to" a later frame. In this manner the user need only specify the spatial-temporal extremes and the in-between frames are calculated by the language. As well as other responsibilities, the animation lanquage must also control a visible surface algorithm in one manner or another.

Display and record techniques give the animation system the capability of viewing the animation during development and documenting the final animation sequence. The most common method of dealing with the output from a visible surface algorithm is to photograph it frame by frame. The image output may be a sweeping horizontal scan line on a refresh CRT, or may be buffered in a 2-D matrix memory with raster video display. Another method is to encode the visible surface algorithm's results and store this information on an analogue (30), or digital (5, 25) disk. The video sequence may then be read from the disk, decoded through a scan-line decoder, and displayed in real time on a video monitor. A refresh CRT needs filters to produce color while a raster-scan display will typically have color output.

2.1.2 Image Quality - The type of visible surface algorithm used by the system determines to a large degree the single (still) frame image quality produced. There has been much work in the area of visible surface algorithms, and no attempt is made here to present all the factors involved. Basically, however, there are two types of algorithms, the first type of which calculate the intensities of a curved visible surface to the resolution of a single picture element. This type includes the "reflected radiation" algorithm of Magi (20), the recursive bivariate surface patch algorithm by Catmull (9), and (10,28). The second type, polygon-based, simply colors or shades in the faces of 3-D polyhedra. Here the Watkins (31, 33) and Myers (24) algorithms serve for examples. The first method inherently produces a smoother surface than the other and lends itself to the calculation of texture, patterns and reflections (7). Thus, for still frame images one typically insists on a visible surface algorithmusing the first method. However, for multiple frame (moving) images, motion contributes significantly to quality. Indeed it may be argued that the quality of motion is the most significant contributor to the quality of the animation. In any case, for

animation a visible surface algorithm using the second method is capable of sufficient quality.

2.1.3 Efficiency - A general definition of efficiency is "the ability to produce without waste." In a computer environment, the most valuable resource to prevent from wasting is computer time. In an animation environment the resource is people time. Efficiency in an animation system implies that all capabilities within the system are easy to use and produce their desired results quickly. For example, the system is inefficient if people who have been trained in animation have to be retrained in mathematics and/or computer programming just so they may apply their previous knowledge in areas of color, form, composition, rhythm, flow and motion. Further, it is inefficient if an animator is forced to stop repeatedly during the production process, because of a slow turn-around time to see the results.

To gauge efficiency at the system level (i.e. system responsiveness and system throughput) an animator must question all phases of the system: How long will it take to make the data? How hard is it to describe the animation? How long will it take to calculate (turn-around time)? How complicated is the final recording process and how long will it be to see the results? System efficiency in an animation environment can only be measured in terms of how long it takes and how hard it is for the animator to get an animation idea off of a storyboard and onto film or video tape. A system which provides direct interaction and fast feedback gives an animator the freedom to experiment with the system and get a feeling for what kind of animation can be done.

# 2.2 Problems and Tradeoffs

Fitting the algorithms used for producing, handling and displaying 3-D color data together into a unified animation system causes problems which effect the system's total performance. The problems are due to fixed limits within the system determined by how much time, money and memory was available. Trade-offs occur as some features must be lost in order for others to be implemented.

For example the amount of directly addressable memory available determines how much data memory and instruction memory can coexist. The size of the data space limits the complexity of the objects while instruction space can decide capability and faster response times since program overlay and task switch techniques can be avoided if all the programs are in main memory together.

Another example is image quality and its relationship to capability and efficiency. A high-order parametric surface equation realistically describes a smooth curved surface and has an increase in image quality over shaded polyhedra. While it may not be difficult to generate, the data for this type of algorithm without a control language, presents difficulties for the animator. The algorithm can also take a considerable amount of calculation time to generate the final pictures. For instance, there are some excellent results with Catmull's method that took 25 minutes on a PDP 11/45 for a single picture (7). Calculation time becomes important in an animation sequence where one minute takes 1440 or 1800 frames (depending on film/video recording). If polygon based shading algorithms are used, image quality drops but capability and efficiency increase (especially if the algorithm is efficient).

Another trade-off in an animation system is the means of displaying the data and recording the final sequence. Film offers higher quality (resolution and contrast ratio) compared to video, but must be chemically processed before the results can be seen. Video however, has the advantage that it can be immediately seen as it is being recorded and the video tape can be reused. Also color is a natural component in a video system whereas it must be added through filters for the film process.

It is often said that standard TV display of computer pictures is of low resolution because one sees the jaggies. This assumption is quite misleading and one should make a distinction between the inherent resolution of TV and computer generated pictures. For instance, if a color TV camera is recording a rotating 3-D color cube (a realworld object) and it is displayed on the monitor viewed at a distance 5 times the height of the screen, then there will be no apparent jaggies. On the other hand a computed animation sequence of a similar colored cube rotating on the monitor also viewed from the same distance will usually have jaggies. The visible surface algorithm must compute the 3-D position, intensity, hue and saturation for each point generating the scan lines to display the picture. Typically there is a certain percentage of error in these calculations and the computational time required to overcome these errors can be lengthy. What one must consider are the trade-offs. While high picture quality is important and desirable, what does it mean in the context of moving images and the bandwidth limitations of an NTSC signal? Vision research suggests that less picture resolution is necessary for moving images than static images.

# 2.3 Other Systems

Based on the literature to date, there have been many computer graphics facilities which have implemented either a technique for generating 3-D objects, an animation language, or a visible surface algorithm. Two examples would be the University of Utah which produced the Watkins Algorithm (31,33), and Archuleta's work at Lawrence Livemore Laboratory (2) where he implemented a fast version of the Watkins Algorithm on a CDC 7600. However, only a few facilities have attempted to integrate these fundamental capabilities into one complete system for the expressed purpose of animation.

2.3.1 An experimental 3-D animation system was developed at the IBM Watson Research Center by Appel et al. (1). This system produced output to a high resolution microfilm recorder in the form of hidden line or shaded objects. A special "movie specification language" was used to control motion, changing viewpoints of perspective and a remote light source capable of casting shadows. Efficiency in the system was increased by sharing program tasks among an IBM 360/67, a 360/91, and a 1130. 3-D data was entered into the system either by interactively picking points with an IBM 2250 or by

"encoding manually when additional artistic freedom is required."

2.3.2 Case Western Reserve University has a computer system which can generate shaded perspective pictures in real time. This "Shaded Graphic System" was developed for Case by Evans and Sutherland Corporation at a cost \$400,000. It consists of a graphics processor driving a pipeline of special purpose hardware for matrix multiplication and shading. Sharing memory with the graphics processor is a PDP-11 with a 10 megabyte disk and an assortment of I/O devices. 3-D data is processed on a scan line by scan line basis by a hardware implementation of Watkin's hidden surface algorithm and sent to a shader which uses the Gouraud shading technique (21). The resulting image is displayed on a raster scan CRT for realtime display. A camera unit with color filters under computer control is used to produce computer animated films.

Jones (22) describes a high level programming language he implemented for the Case system. It consists of a complete implementation, for the PDP 11, of Algol-60 with the addition of string variables, I/O facilities, and extensions for handling graphic shaded images. The primary purpose of this work was to facilitate the use of a custombuilt system which can produce shaded images in real time. According to Jones one important advantage of Algol was its block structure which Jones decided would lend itself quite nicely to the description of graphical structures. The consequence of this approach is that just as Algol itself is a way of talking about algorithms, the graphic-extendedAlgol is a way of talking about graphical data structures.

Currently, the system requires 3-D data to be entered through a dual data tablet arrangement which means the animator must provide detailed drawings from several viewpoints (something most animators with their "sketchy" storyboards don't have readily available). But besides this and the lack of color in the system, the combination of Jones' extended Algol-60 language and the powerful graphic display processor presents a good example of a general purpose 3-D real-time animation system. Most of the film "Sphere Eversion" was calculated with this system.

2.3.3 Credit should be given to Goldstein (20), Nagel, et al. (13) and Elin (14) for their pioneering work in the area of 3-D shaded animation with the Magi-Synthavision system. The unique visible surface algorithm uses curved patched surfaces, but its approach is fundamentally different from others. "Rays" are fired from some point in space and traced to the first visible point on a 3-D object. The advantage of this technique is that since the rays are stopped at the first surface encountered, no time is spent examining the parts of the model which would be normally hidden. The system is capable of generating data with a sophisticated "combinatorial geometry" technique (thus preventing the decrease in data generation capability, typically associated with parametric surface algorithms). Here, "the user specifies the geometry by establishing two tables. The first table contains the type and location of the bodies used in the geometric description (there are nine

basic shapes). The second describes the physical region in terms of the bodies in table 1 and the three Boolean operators, '+', '-', and 'or'. Each region has a unique region number and the bodies are numbered in the order of their occurence. The model is completely described in terms of its region number."

The input to the system also includes "the location and characteristics of a camera (focal length and size of image plane), the direction from which the light is coming and a set of instructions called "Director's Language," which tells the computer how to treat the objects (animate the actors) in the film."

The calculated visible surface output is stored on magnetic tape. Using this tape as input a second pass through the computer is made to convert the region-intensitydata into color-intensity. The film process (based on color addition) requires that the output tape be made with three weighted red, green and blue frames for every one frame from the input tape. The tapes, in this form, are fed through a Data General Minicomputer to a precision CRT. The images are filmed through a computer controlled color wheel (triple exposure-once for red, green and blue).

The Magi-Synthavision system has taken an excellent approach to 3-D shaded animation with the use of "Combinatorial Geometry" and a "Directors Language" to control their calculated visible surface output. Unfortunately the system suffers from a lack of interaction, because to use these powerful facilities the animator must keypunch in the commands to control both the data generation and animation process. Also, calculation time is slow, ranging from 30 seconds a frame for extremely simple data, up to around 20 minutes a frame (15).

# 3. <u>Anima II</u>

The animation software system has been implemented in a standard minicomputer environment (Diagram 1) with a PDP 11/45 as the central processing unit. The CPU has 64 K of core memory (32 K of which contains the RSX 11-D operating system) and 32 K of MOS memory. In addition, the peripherals include: a 4096 x 4096 Vector General refresh CRT with joystick, buttons and dials; a 44 mega-word (16 bit), "3330 type" moving head disk used as the system disk; a special purpose color, raster-scan decoder which serves as our real-time video interface. The software in the animation system was written in assembly language to increase efficiency (Diagram 2).

The system, Anima II, supports an environment in which a user trained in areas other than computer programming, is capable of:

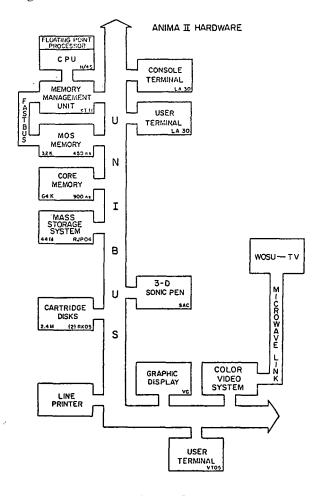
- <u>Creating</u> complex color polyhedra with a realtime interactive geometric "modeling" routine.
- Writing an animation script describing parallel, keyframe motion dynamics controlling multiple objects.
- <u>Animating</u> the script using a specially written animation language processor in which the Myers visible surface algorithm is the kernal

component in the calculations for the final display output.

 <u>Displaying</u> and directly <u>recording</u> in real time, the color video sequence that was calculated and stored (in binary) on the system disk by the animation language.

While each of these areas have noteworthy theoretical and implementational features in and of themselves, what is significant about the Anima II system is the integration of these separate, complex processes into a complete system, which is both easy to use and efficient.

Currently the Anima II system is supporting animation projects in the areas of education, telecommunication and art as well as research projects for astronomy, statistics and computer-aided design.



#### Diagram 1

# 3.1 Data Generation

The objects in the animation sequence are created with Parent's (29) interactive data generation program. The user views and interacts with the objects in real time on a random scan CRT. Concave polyhedra are joined and intersected to form complex shapes. The object can be bent or warped into

# ANIMA II SOFTWARE

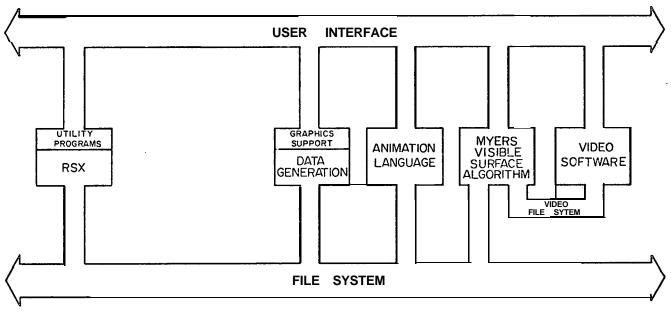
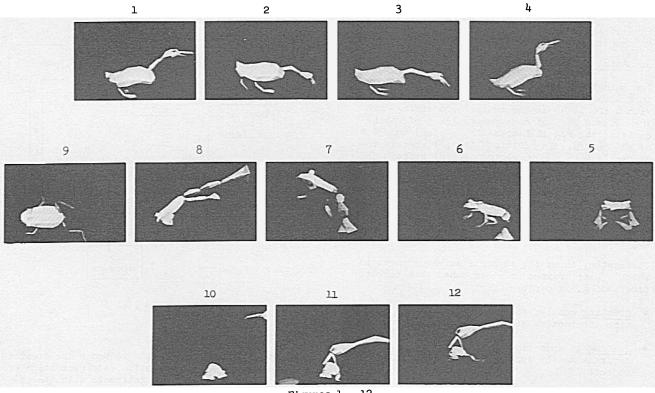


Diagram 2



Figures 1 - 12

multiple shapes for animating later. Transparent to the user is the data structure of the objects which consists of closed polygons forming closed convex or concave surfaces. The user is only aware of positioning two objects in some relation to each other, pushing a button, and either joining the two together or cutting one into the other. The process is accumulative and can be repeated as often as necessary to build the final object. The color of the objects can be specified when the user chooses his primitive objects (a green ball can cut a green hollow in a red cube) or individual faces may be selected and "painted."

The data generation routine uses 32 K of MOS memory for instruction and data space and uses 20 K of core memory for a device handler which buffers the display lists and refreshes the Vector General. The routine can handle up to 2500 unique edges. A user accustomed to the "sculpting and building" approach of the routine can make an object in a very short time. This can range from 5-15 minutes for a simple shape such as a block letter, 2-3 hours for the frog and duck in Figures 1-12, and up to five hours for complex data like the "Jack-in-the-Box" shown in the video tape accompanying this presentation. These times also include the bending and warping process to make multiple shapes for interpolation (blending) in the language. A detailed presentation of the intersection algorithm used in the data generation routine is being given at this conference by its author.

# 3.2 Script

Once the 3-D objects have been created, the user controls the rest of the animation process through his script. The script is a story-boarded animation idea, transcribed into a list of instructions written in the special descriptive syntax of the language. The language of the Anima II system offers a means of imitating the complex motions of "real" world objects by breaking each motion into simple, but precisely controlled changes through space and time. Language instructions are individually scheduled to be active over a range of time during the animation sequence. When the instruction has reached its time limit, it can be rescheduled to be active later in the sequence, or it can be removed from further consideration. An instruction specifies key frame time parameters and it also describes key frame spatial transform parameters. However only the extreme parameter which the instruction is changing to need be given, because the language keeps track of where each object currently is. This saves animators from having to keep records on their own of what they have done so far in the script. They specify where they want to go "to," and the language calculates what the vector should be to get there. This applies whether the transform affects position, rotation, size, shape, or path. Using this format, a combination of a "set" and a "change" scene directive can completely control one simple motion as in the example:

# SET POSITION name X, Y, Z, AT frame 1

CHANGE POSITION name TO, X, Y, Z, FROM frame 1, TO frame 100

These two scene directives read as "set the

position of some object to the point specified by X, Y, Z on the first frame of the sequence (coordinate and frame values can be given as numbers or symbolic variables)." "At the same time, <u>change</u> the position from the point where it was set, to a new point given by different X, Y, Z values and be there by frame number 100."

The ability to schedule the language instructions allows the user to animate multiple objects without being concerned with looping or programmatic flow control. This notion of parallel commands is quite different from the typical approach found in other graphics languages in which the animation is controlled by guiding an internal program address counter or pointer, into, through and out of a series of transformation control loops.

The photographs in Figures 1 through 12 are several of the extreme positions taken from an animation sequence involving a duck, a frog and the meeting of the two. The first four stills show the duck in a head-down, head-up position as it takes a drink of water. What the photos can't show is the duck wadling, wagging his tail, flapping his beak, as well as changing his orientation (turning to one side then the other) and moving through space--all at the same time. The scheduled commands in the script can be given quite directly to control the transformations needed for this type of animation. The animator works on motions independently, component by component. In the case of Figures 5 to 9, the animator created the frog and then intuitively "bent" the legs and arms into the extreme shapes that make up jumping and swimming. Then, in the script, the animator decides what the timing will be to get the frog to change realistically from one shape to another. When this is settled, the animator may decide on when to turn the frog during the sequence. After that, how should the frog be moved to give the effect it is swimming. Here, acceleration and deceleration can be controlled by the animator to improve the quality of motion. The introduction of the duck, as seen in Figures 10 to 12 presents no difficulty to the user. Commands animating the duck and frog are given directly, in parallel and with no regard to mutual interference.

When an animator is satisfied with the actions of the objects, he has the option of controlling the whole scene. Commands are used similar to conventional animation terms such as pan, tilt, zoom and field, which change the relationship of the observer to the objects. Other features of the animation language include color, brightness, fades, **lighting control** in the form of independent position and rotation of multiple light sources and the ability to calculate a single frame or short animation segment within the script.

#### 3.3 Animation Language

When the objects have been made and the motion described, the animator need only evoke the language to calculate the final video sequence. The language processor, designed and implemented by Hackathorn (12), follows the user written script. It compiles an animation file which contains all the object and color parameters needed by the visible surface routines next in the production process. If, for example, the script describes a sequence animating thirty multi-colored block letters and lasting for twenty seconds (600 frames), then the compiled animation file will look like a sequential list of six hundred dynamically changing data structures, each defining the spatial and display parameters of a collection of colored surfaces for one frame.

The program tasks for the animation language is divided into four routines: preprocessor, scheduler, interpreter, and compiler.

# 3.3.1 Preprocessor

The preprocessing routines are concerned primarily with building the data structure, but also with the keyword parsing of the script syntax. This routine is controlled first by the prescene directives then by the scene directives of the user written script. The prescene directives instruct the preprocessor in the building of the data structure for the entire animation sequence. The data structure includes:

- Face and vertice information describing the three-dimensionalpolygonal surface of each object in the sequence.
- The different possible shapes that any object can change into.
- Group pointers for each object.
- Sub-group pointers within each object (object parts).
- Multiple "floating" light sources.
- Multiple, three-dimensionalpaths through space, sharable by all objects.
- A separate color for the inside and outside of every face for every object.

The animation language currently can control up to 128 objects, groups of objects, or possible object shapes, however, the real limiting factor in the maximum complexity of the system is the 32 K address space of the PDP-11/45 CPU. The animation language uses 32 K of MOS memory and 32 K of core memory. This allows 20 K of object data space (4000 to 5000 unique edges) and elsewhere a 16 K section for buffering shape vertices and path vertices (about 5300 points at 3 words each). If there is room in memory for the data, the language can control 128 objects, 128 shapes, 64 groups, 32 paths and over 500 command instructions.

The preprocessor routine parses, interprets, and executes each of the prescene directives until it comes to a SCENE START directive in the script. For the remainder of the script the data structure is fixed, no new objects can be added, and the routine parses nothing, but scene directives. Each scene directive gets parsed and converted into a "command block," kept as part of a data list in memory. A command block has all the parametric and key frame (schedule) information in it that was given in the directive line. It also contains pointers into the data structure, plus a workspace area big enough to hold the unique motion values which will change from frame to frame.

# 3.3.2 Scheduler

The scheduler is the first of three routines which are evoked for each frame. The scheduling routine is event driven by the start of each new frame. Every frame it:

- Sequences through each command block in the list, compiled by the parser.
- Judges whether the command block is flagged active or inactive after a comparison of the key frame information in the command block and the current system frame counter.
- Updates the motion parameters in the workspace area if the command is active this frame.

The scheduler routine works double duty by both scheduling the command blocks and updating the unique motion information that each block carries. It is at this state that the concepts of "set" and "change" become important. A "set" command block holds its initial parameters through its time range within the animation process. However, a "change" block has a direction initially calculated as specified by the animator with a "change to" directive. From the first frame of activity, the direction (an increment in X, Y, ' and Z) of the "change" block will be added to the block's own internal workspace memory. These incrementing (positive or negative) parameters get interpreted and executed as if they belonged to a "set" commandblock. This information is used, with no further modifications, by the interpreter routines in doing the actual transformations to the data structure.

# 3.3.3 Interpreter

After the command blocks have been scheduled for the current frame, the interpreter finds each active command, determines the parameter type (rotation, position, size, color, shape, path, etc.), and performs the necessary motion or display transformations to the data structures. The key to the interpreter is that for each new frame, all command blocks scheduled active will start their transformation on the original data. In this manner, both the order of the commands and the range of their schedule determines what transformations will be done to the data on any given frame.

# 3.3.4 Compiler

The compiler routine compiles a data file as opposed to executable code. The routine calculates the color of each face, does perspective transformations, clips all faces not seen by the observer and builds an animation file containing a complete scene description of every frame in the script.

The color of a face is a product of the relationship between the current positions of the light sources and the plane of the face. The system has three light sources which it keeps as X, Y, Z points in space and allows them to be translated and rotated just like objects. The distance each of the light sources is from the face, decides a weighted brightness. From this relationship a value between 1 and 224 is determined. This value corresponds to a color palette made up of 224 entries, each entry describing a fifteen bit red, green, blue hue combination. The color palette is logically organized into eight intensity-chroma sections with 28 entries (the first entry is the darkest color and the last entry is the brightest). When the object is created in the data generation stage, it is "colored" by assigning one of the eight intensity-chroma sections to each face. With the information supplied by the light source calculations the final offset into the color palette is produced.

The scene has a user-specified observer position. Every object has its own "picture plane." With this information, the compiler routine calculates perspective. Each frame, the vertices after being transformed by the interpreter are projected onto a picture plane. The 'Z' axis coordinates are unaffected by the perspective so that depth comparisons may be done later by the objects in memory as one object. It checks which faces can still be seen and appends the animation file with:

- The faces in the object that are displayable.
- The colors of the displayable faces.
- The transformedvertices for the current frame.
- Miscellaneous display parameters i.e., z-clipping plane position, and background color.

When the last frame of the script has been compiled, what is left is a data file on the system disk ready to be turned into the final color video sequence by the visible surface algorithm and a raster-scan conversion routine. Up to this point the calculation time has been relatively short. The only major calculations in the language are the dot products and face normals needed for the light source equations. As a result, the language typically calculates a 300 frame (10 second) sequence in under 5 minutes.

Through the script the animator may request that the animation file on the disk be played back (in real-time) to the Vector General. Since the transformations of the objects are already completely defined for every frame in the sequence, the V.G. playback routine has no computation requirements. This makes for an excellent way of previewing the animation sequence to get an idea about the motions, but of course no color or lighting information can be displayed. If V.G. output is not specified in the script, the language automatically evokes the visible surface.

# 3.4 Visible Surface Algorithm

The visible surface routine of the Anima II system is a version of the Myers Algorithm. Full implementation details of the original algorithm may be found (24), but for completeness, a brief description of the algorithm, as it affects the animation process will be discussed.

The program uses 32 K of MOS memory, containing a data space of 20 K. At the beginning of each new sequence, the program reads a list of faces from the animation file left by the language. Here we note that the language has described all the objects in the animation sequence to appear as one to the visible surface routine, also that all polygons

created with the data generation routines have been reduced to triangles.

For each frame and starting with the first, the procedure is as follows. The face's information is read in. This contains faces clipped out of view, backfaces removed optionally by the animator, and color for each displayed face. Next the list of unique vertice is read in as well as miscellaneous information such as background color.

The program checks each face against the face file for this frame and if it is to be displayed (not clipped or "back faces" removed) the face is added to a list of faces whose highest 'Y' value is identical to that of the current face. When all faces have been checked for displayability, the algorithm begins producing the visible surface output. As is typical of linear to raster conversion and visible surface algorithms, a scan line at a time is processed. Starting at the highest of the 512 scan lines, lines are processed one line at a time until all lines are processed. Each line is processed as follows. If the line contains no active faces (i.e., no face starts, crosses or ends on the line) it is ignored. If the list of faces starting on the line is not null then all of the faces on the list undergo a format conversion and are added to a list of active faces. If the list of active faces is not null then each face on the list is processed, one face at a time, in whatever order the faces on the list are in, until each active face has been processed.

Processing a face means processing a segment of a face, since one scan line at a time is processed. Thus the list of active faces can be thought of as a list of segments to process on a scan line. The first segment of the list is scanned (i.e., converted to points). The 'Z' (distance from the observer) and intensity values for each point are stored in the appropriate places in the ZVSLS (Z values scan line structure) and IVSLS (intensity values scan line structure) respectively. Both the ZVSLS and IVSLS consist of 512 locations, each location of which corresponds to a horizontal position on the output raster. At each horizontal position the 'Z' value of the new point is compared with the 'Z' value of the point in the ZVSLS. If the new 'Z' is closer to the observer then both the ZVSLS and IVSLS values at the current horizontal position are updated with the values from the new segment. If the new 'Z' is farther or equal then no updating occurs.

After processing a segment the corresponding active face is updated for the next scan line. If the lowest point of the face has been passed then the face is removed from the list of active faces. After processing all faces on the list of active faces for a scan line the scan line is converted into run length encoded binary data and stored a scan line at a time on the system disk.

Given a typical animation sequence which contains

polyhedra of around 1000 edges and covering an area of about one quarter of our TV monitor, the visible surface and raster scan conversion calculation of a 300 frame (10 seconds) sequence takes between 5 to 10 minutes. If the complexity doubles, but the area remains the same then the same sequence will take 7 to 12 minutes. However, if the area doubles and the complexity remains the same, the sequence will take 10 to 20 minutes.

# 3.5 Display and Record

Currently we are using a standard broadcast television as the viewing mechanism, a large capacity digital disk for image storage and broadcast video for the raster-scan format image representation. The broadcast video is not stored in composite NTSC format, but rather is stored as run-lengths of particular intensity-chromacombinations which are converted (in real-time) to composite NTSC format for display. The use of run-length encoding is our response to the insufficiency of current computer technology to easily handle the large quantities of information implied by raster-format representation of dynamic images. For example, a raster-format dynamic image of 512 by 512 resolution, 8 bits per resolvable element information content, 30 frames per second display rate and 30 seconds duration requires over 235 million bytes of storage. The implied data transfer rate (8 million bytes per second) is prohibitive within our general purpose design strategy. This is due to the fact that although disks of over 200 million byte capacity are available, the transfer rate available is less than 2 million bytes per second.

The run-length decoding and analog systems were constructed by Dr. John Staudhammer and his associates (DIGITEC, Inc.; Box 5486; Raleigh, N. C. 27607). The analog system and rearend of the runlength decoding system are similar to an earlier system built under Staudhammer's direction. The decoding system converts our run-length format to that used in Staudhammer's earlier system. (30)

A dynamic sequence is transferred from the disk to the TV according to the following scheme. A 32 KB run-length buffer is divided into two 16 KB buffers for double buffering. A buffer is filled from the disk. While this buffer is being filled, information to/from the disk controller from/to the CPU must be multiplexed with the data from the disk. This multiplexing is automatically handled by the UNIBUS priority arbitration unit. Fortunately, the quantity of control information necessary to run the disk is a small percentage of the quantity of data being transferred. Also fortunate is the fact that the dual ported MOS main memory permits the instructions and associated data of the control program to be fetched simultaneously with the data being stored from the disk. Thus, there are virtually no memory cycles lost directly to the control program.

Information flowing into the run-length decoding system is buffered in an internal 32 KB MOS buffer before it is decoded. This is the reason that information may be transferred from the MOS main memory buffer into the decoding system with no concern for field or frame boundaries. More explicitly, since field and frame boundary information is contained in the data, putting off decoding the data until after information transfer permits the data to be treated as a uniform stream.

The calculations below are intended to give a quantitative indication of the capabilities of the system. It should be noted that in order to provide the clearest calculations, minor overheads such as start of field instructions are ignored.

The following calculations assume an average of one byte per run. This case is approached for images with (typically) fewer than 33 intensity-chroma combinations within a scan line and fewer than 25 within a field. The disk specifications are those of the manufacturer. Since the RJPO4 disk system is (relatively) the slowest part of the system, it determines the maximum performance level. For contiguously stored files (as video files have to be in this system) the disk can be read continuously at maximum possible speed with the exception that some time (7 milliseconds) is lost when changing cylinders. Since there are 19 tracks per cylinder and the disk requires 16.7 milliseconds for one revolution, one cylinder can be ready every 317 milliseconds. Since 214,016 bytes are stored per cylinder, the average data transfer rate is 675 bytes per millisecond. Allowing 10 milliseconds for change of cylinder, 207,266 (214,016 minus 6,750) bytes can be obtained for every cylinder read. Note that the storage space "passed over" for change of cylinder is best wasted as an extra revolution would be required to retrieve it. Thus, the average data transfer rate is 654 (207,266 divided by 317) bytes per millisecond. Since each TV frame lasts about 33 milliseconds, this is 21,582 bytes per frame. At one byte per run, this is 21,582 runs per frame. Since the disk has 411 cylinders and the system is retrieving 207,266 bytes per cylinder, there are 85,186,362 retrievable bytes. At a maximum of 21,582 runs per frame, this represents 3,947 frames. Since the data is contiguous, any reduction in runs per frame directly translates into more frames. Thus, at 2,158 runs per frame there are 39,470 frames. (25)

#### 4. Conclusion

The development of computer generated 'solid' object animation is changing the way an animator approaches the documentation of an idea. Conventional animation involves drawing and redrawing planar images on each frame throughout the entire sequence. Image creation and image animation are very often the same process. But in a 3-D computer animation environment, the user first builds a colored object then animates it and these processes are separate. The approach of 3-D color animation is similar to that found in other disciplines such as Cinematography, Theatre and Choreography. Here actors or dancers are chosen and given their roles by a director who is responsible for the whole show. The approach is closer still to that of puppet animation in which the work if Jiri Trinka, Willis O'Brien (King Kong) and Jim Hensen with his Muppets serves as excellent examples.

The implementation of such an animation system requires balancing system requirements against the available resources, while at the same time keeping some notion of efficiency in mind. Anima II, has been developed and implemented as one solution to the production of 3-D color animation. Each of the subsystems in Anima II have been especially designed to both interact freely with a user and integrate transparently into a unified system. While sitting at one work station, a user of Anima II can create, animate and display 3-D colored objects, then directly record the animation onto standard video cassette tape. The system has limitations in the areas of data complexity: 5000 unique edges per scene; and data transfer: limited mainly by the system disk which can transfer about 20,000 bytes (1-3 bytes per run-length) each video frame. Currently methods are being explored to improve these areas and the areas of image quality and total system throughput.

#### ACKNOWLEDGEMENTS

The Anima II system was designed and implemented by the Computer Graphics Research Group at The Ohio State University. Work on the project was performed by Charles Csuri (Director), Allan Myers, Richard Parent, Timothy VanHook, Diana Rainwater, and the author. Funding for this project was provided by NSF Grant DCR 74-00768.

#### REFERENCES

- Appel, A., Stein, A., Landstein, J. (1970). The Interactive Design of Three-Dimensional Animation, <u>Proceedings of the Ninth Annual</u> <u>UAIDE Meeting</u>.
- 2. Archuleta, Personal Communication with CGRG.
- Baecker, R. M. (1969). Interactive Computer Mediated Animation. Dissertation, <u>Massachusetts Institute of Technology</u>.
- Baumgart, B. G. (1974). Geometric Modeling for Computer Vision. Dissertation, Stanford University. <u>NTIS Report Number AD/A-002261.</u>
- Belady, L. (1970). TV Plus Computer Equals Videographics. <u>Proceedings of the Ninth</u> <u>Annual UAIDE Meeting</u>.
- Blasgen, M. W., Gracer, F. (1970). KARMA: A System for Storyboard Animation. <u>Proceedings of the Ninth Annual UAIDE Meeting.</u>
- Blinn, J. F., Newell, M. E. (1976). Texture and Reflection in Computer Generated Images. <u>Communications of the ACM</u>, Vol. 19, No. 10.
- Braid, I. C. (1975). The Synthesis of Solids Bounded by Many Faces. <u>Communications of</u> <u>the ACM</u>, Vol. 18, No. 4.
- Catmull, E. (1974). A Subdivision Algorithm for Computer Display of Curved Surfaces. Tech. Report UTEC-CSC-74-133, University of Utah.
- Clark, J. (1976). Hierarchical Geometric Models for Visible Surface Algorithms. <u>Communications of the ACM</u>, Vol. 19, No. 10.
- 11. Csuri, Charles (1975). Computer Animation, <u>Proceedings of the Second Annual Conference</u> <u>on Computer Graphics and Interactive Tech</u> niques--SIGGRAPH '75.
- Csuri, Charles A. (1977). 3-D Computer Animation. <u>Advances in Computers</u>. Academic Press, Inc., New York.
- 13. Davis, J. R. (1968). A Model Making and Display Technique for 3-D Pictures, Proceedings of the Seventh Annual UAIDE Meeting.
- Elin, L. (1975). Synthevision: Serendipity from the Nuclear Age, <u>Artist and Computer</u>, edited by R. Leavitt, Harmony Press.

- Elin, L. (1977). Presented at National Conference and Workshop on Electronic Music and Art, University of Buffalo, Suny.
- 16. Film "Walking Man," University of Utah.
- 17. Film "NASA Space Shuttle" General Electric.
- 18. Film "Sphere Eversion" N. Max.
- 19. Gattis, W., Watson, (1971). An Input Translator for Animation and Its Relationship to Key Position Character Animation. <u>Proceedings of the Tenth Annual UAIDE Meeting.</u>
- 20. Goldstein, R. (1971). A System for Computer Animation of 3-D Objects, <u>Proceedings of</u> the Tenth Annual UAIDE Meeting.
- Gouraud, H. (1971). Computer Display of Curved Surfaces. <u>IEEE Transaction on</u> <u>Computers</u>.
- 22. Jones, B. (1976). An Extended ALGOL-60 for Shaded Computer Graphics. <u>Proceedings ACM</u> <u>Symposium on Graphics Languages.</u>
- Lafue, G. (1975). Computer Recognition of 3-Dimensional Objects from Orthogonal Views. <u>Research Report No. 56</u>, Institute of Physical Planning, Carnegie-Mellon University.
- Myers, A. J. (1975). An Efficient Visible Surface Program. Technical Report to the National Science Foundation, Grant Number DCR 74-00768A01.
- 25. Myers, A. J. (1976). A Digital Video Information Storage and Retrieval System. Proceedings of the Third Annual Conference on <u>Computer Graphics and Interactive Techniques</u>--SIGGRAH..
- 26. Max, N., (1975). Computer Animation of the "Sphere Eversion," <u>Proceedings of the Second Annual Conference on Computer Graphics</u> -SIGGRAPH.
- Negroponte, N. (1973). Recent Advances in Sketch Recognition. <u>Proceedings of the</u> <u>National Computer Conference</u>.
- Newell, M. (1975). The Utilization of Procedure Models in Digital Image Synthesis, Ph.D. Dissertation, University of Utah.
- 29. Parent, R. E., Chandrasekaran, B. (1976). Moulding Computer Clay. <u>Pattern Recogni-</u> tion and Artificial Intelligence.

(C. H. Chen, Ed.) Academic Press, Inc., New York.

- Staudhammer, J., Eastman, J. F. (1975). Computer Display on Colored Three-Dimensional Object Images. Proceedings of the Second Annual Symposium on Computer Architecture, pp. 23-27.
- 31. Sutherland, I. E., Sproull, R. F., Schumacker, R. A. "A Characterization of Ten Hidden-Surface Algorithms," ACM Computing Surveys, Vol. 6, No. 1, pp. 1-55, 1974.
- 32. Sutherland, I. E. (1974). Three-Dimensional Data Input by Tablet. Proceedings of the IEEE. Vol. 62, No. 4, pp. 453-462.

- 33. Watkins, G. S. (1970). A Real-Time Visible Surface Algorithm. <u>University of Utah</u> <u>Technical Report UTEC-CSC-70-101.</u>
- Wein, M., Burtnyk, N. (1971). A Computer Animation System for the Animator. Proceedings of the Tenth Annual UAIDE Meeting.
- 35. Wein, M., Burtynk, N. (1975). Computer Animation of Free Form Images. Proceedings of the Second Annual Conference on Computer Graphics and Interactive Techniques--SIGGRAPH '75.
- 36. Whitney, John, Citron, J. (1968). Camp-Computer Assisted Movie Production, Proceed-<u>ings of the AFIPS Fall Joint Computer</u> <u>Conference</u>.