SLAC-PUB-939
(MISC)
AUGUST 1971

AN ON-LINE SYSTEM FOR INTERACTIVE PROGRAMMING AND COMPUTER GENERATED ANIMATION *

Robert C, Beach, Mary Anne Fisherkeller and George A. Robinson
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305

ABSTRACT

This paper describes an experimental graphics system at the Stanford Linear Accelerator Center. This facility consists primarily of a Varian 620/i computer and an IDIIØM display console with the 620/i linked to an IBM SYSTEM/360. In addition, a stereoscopic viewer has been constructed, and a motion picture camera has been completely synchronized with the display. Programming systems which have been developed include (1) a Graphic Sub-Monitor with a complete text editing system and on-line access to most of the facilities of the SYSTEM/360, (2) a package of PL/I procedures which may be used to prepare highly interactive programs, and (3) an anima tion generation system utilizing the compile-time preprocessor of PL/I. Animation sequences may be viewed at the display console in either 2-D or 3-D, and may be recorded directly onto motion picture film.

Key words and phrases: Computer Graphics, Interactive Graphics, On-line System, Stereoscopic Viewers, Computer Animation

CR Categories: 3.80, 4.22, 6.36

(Submitted to Communications of the ACM.)

---

* This work was supported by the U. S. Atomic Energy Commission.

1. Introduction

The Computation Group of the Stanford Linear Accelerator Center is engaged in the investigation of new computer techniques and hardware. One area of interest to the group is interactive systems that might aid a physicist in his data analysis. In order to develop some experimental systems, the SLAC Graphic Interpretation Facility, an interactive display which could be connected to an IBM SYSTEM/360, was acquired in December 1968. It is described in Section 2. The initial effort was to produce an on-line programming system with which to develop other capabilities. This system, the Graphic Sub-Monitor, includes a page-oriented text editor and essentially all of the facilities provided by the operating system of the IBM SYSTEM/360. It will be described in Section 3.

Using the Graphic Sub-Monitor, a package of PL/I subroutines was developed to help a programmer prepare highly interactive programs. This package is described in Section 4. Finally, in Section 5, a computer animation system is described. The animation system is based on the subroutines described in Section 4, and provides a means of viewing the animation on the display or recording it on motion picture film.

2. The Graphic Interpretation Facility

The primary computing resource at SLAC is an IBM SYSTEM/360 Model 91 with 2048K bytes of storage, operating under ØS/MVT with HASP.

The two basic components of the Graphic Interpretation Facility are a Varian Data Machines 620/i computer [1] with 8K 16-bit words of storage, and an IDIIØM display console [2] with a 21 inch CRT made by Information Displays Inc. When the display is operating, the 620/i memory contains a program for the 620/i to execute and a program (display file) for the IDIIØM to execute. These two programs run concurrently with the IDIIØM stealing cycles from the 620/i.

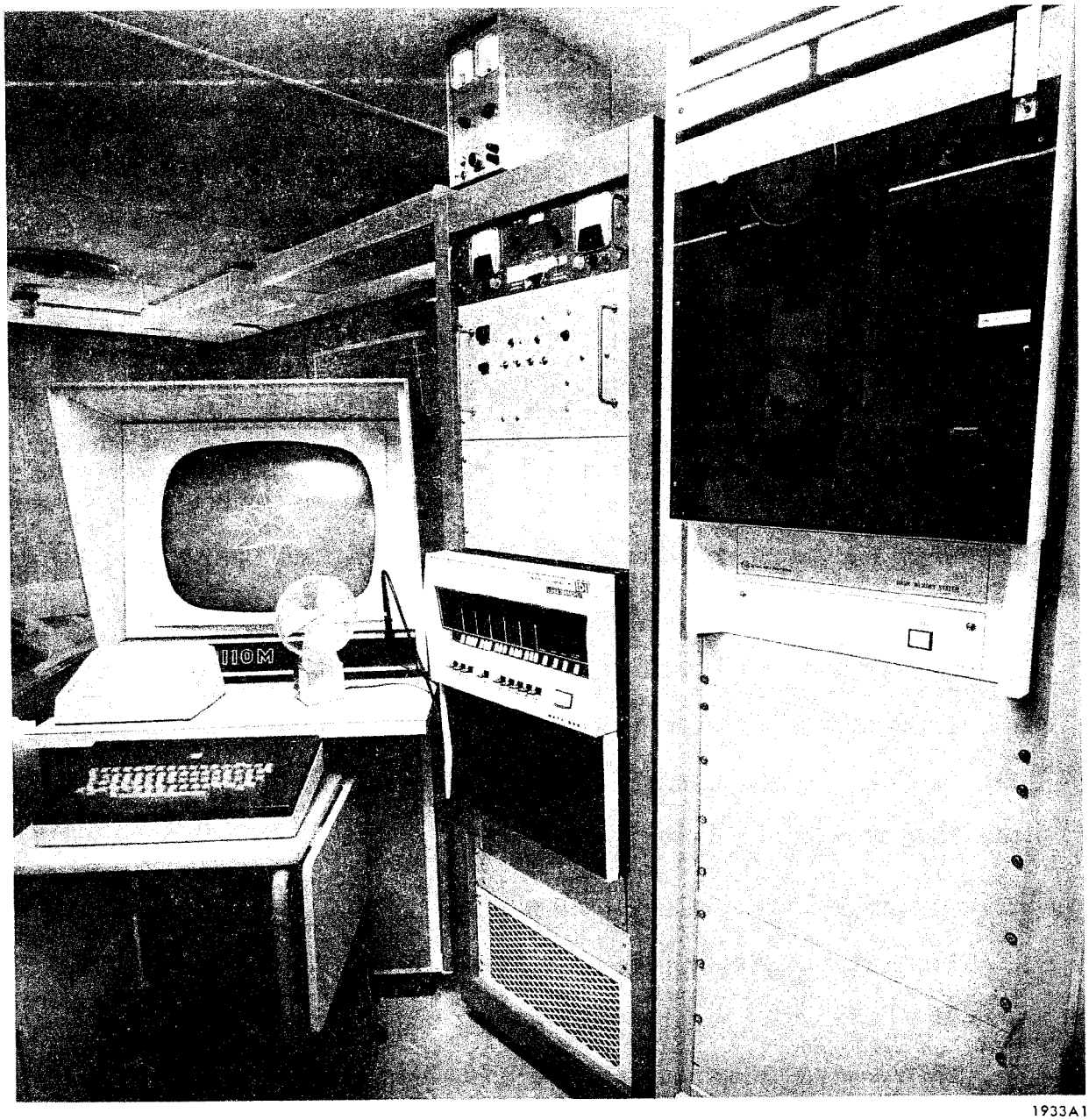

Figure 1: The Graphic Interpretation Facility

The instructions (orders) in the display file may display characters, points, or straight line segments, perform unconditional or subroutine jumps, count and index, or interrupt the 620/i. Characters, points, and lines are positioned on a 1024 by 1024 raster unit grid on the face of the CRT. Characters may be plotted in four sizes, either horizontally or vertically (rotated 90 degrees counter-clockwise). Lines may be drawn in any of four modes: solid, dashed, dot-dashed, or dots. Displayed information may be in any of four intensity levels and may be in either a steady or blinking mode. The 620/i instruction set includes, in addition to the usual set for a modern small computer, instructions to start and stop the IDIIØM in its execution of the display file, and instructions to read and reset registers associated with the display operation.

Interaction at the console is by means of a solid state alphameric keyboard, a light pen, or a function keyboard with 32 buttons. Under program control, portions of the display file may be designated as light pen sensitive or insensitive. When the light pen is pointed at a sensitive item on the CRT, the 620/i will be interrupted. The coordinates of the lightpenned item, as well as its location within the display file, are available to the 620/i interrupt routine for processing. An interrupt is also generated by closing or opening a switch on the light pen or at fixed time intervals (1/60 seconds) by a built-in timer. The latter feature guarantees an interrupt within this time interval regardless of the size of the display file and may be used for urgent tasks such as light pen tracking. The function keys generate interrupts on both the closing and the opening of the switch. Plastic overlays may be placed on the function keyboard to identify the purpose of the buttons and the 620/i may sense a code punched into the overlay.

The 620/i and SYSTEM/360 are connected through an IBM 2701 Parallel Data Adapter Unit. Data may be transmitted in either direction through this link.

The 620/i can interrupt the SYSTEM/360, and determine whether the SYSTEM/360 is trying to read or write; however, the SYSTEM/360 is not able to interrupt the 620/i. The back-up store for the Facility is provided by a Vermont Research fixed-head drum with 480K 16-bit words of storage, and a magnetic tape manufactured by Peripheral Equipment Corporation.

In addition, the Graphic Interpretation Facility has two other unusual devices; a 16 millimeter motion picture camera which can be synchronized with the display, and a 3-D viewer. The 3-D viewer was constructed at SLAC by Charles Hoard, and the interface between the 620/i and these devices was designed and built by Michael Hu and Russell Matheson.

The movie camera is a Model 16M ARRIFLEX camera with both a standard motor and an animation motor. The movie camera may be operated in either of two basic modes. In the first, or "over the shoulder", mode the camera uses the standard motor and runs at its usual speed of 24 frames per second. As the camera runs, the 620/i continually senses the shutter to see if it is open or closed. When the shutter initially opens, the 620/i will start the display. After executing the display file once, the display is turned off until the next time the shutter opens. Thus movies can be made of a person using the display console and the CRT will not exhibit the flicker that is usually present in unsynchronized movies. The flicker in unsynchronized movies occurs when the camera shutter and the drawing of the image by the electron beam are out of phase. In the second, or animation, mode the 620/i sends signals to the animation motor telling it to advance the film and open the shutter. When the shutter has opened, the 620/i will flash the picture on the CRT. The shutter will then close, either automatically or on a signal from the 620/i. The advantages of equipment of this nature are low cost and versatility. The usual film recorders cost substantially more than the movie

camera and do not have the option of "over the shoulder" filming. Film recorders, however, usually generate higher quality images than CRT's designed for direct viewing.

The 3-D viewer is a device which contains a motor-driven rotating disk with clear and opaque areas. When the user looks through this device, the left and right eyes will alternately be blocked by the opaque areas on the disk. The 620/i can sense when the left or right eye has a clear view through the rotating disk. By presenting each eye with the appropriate display, it is possible to give the user a stereoscopic picture.

Although the Facility can operate in a stand-alone fashion, most of our work has been done using it in conjunction with the SYSTEM/360. The SYSTEM/360 provides fast computation and mass storage while the 620/i maintains the display.

3. The Graphic Sub-Monitor (GSM)

The Graphic Sub-Monitor [3] is a second-level operating system designed to work in problem state on the SYSTEM/360. The purpose of the GSM is to provide an environment in which a user can run any of a variety of jobs in an interactive situation without the annoyances inherent in writing ever-changing job control statements and with maximal flexibility in the sequence of operations he may choose. Among the hundred or so commands in GSM's repertoire, one finds commands for (1) text editing on a page-oriented device such as the IDIIOM (or alternatively, the IBM 2250 or IBM 2260); (2) invoking the language processors (with special emphasis on assembly language, FORTRAN, and PL/I), linkage editors, utilities, and other OS/360 processors; (3) executing user programs for debugging or for interactive production runs; (4) submitting other jobs to be placed in the system's input queues; and (5) interrogating HASP or OS to obtain and display information of interest to the interactive user.

The format of a GSM command is a key word, followed by arguments separated by commas, and terminated by a dollar sign. Data sets that a user wishes to edit are referenced by names he has assigned at the time they were created. Within a data set, each line is assigned a sequential line number as it is entered. When a data set is displayed, the lines will appear in their logical order and not in the sequence in which they were entered. For example, let us suppose that in Fig. 2 lines 133-134 were not present and the user wishes to insert them. He might type in EDIT, EXAMPLE$, which would prepare GSM to perform editing operations on the data set named EXAMPLE and would display the first page (usually 40 lines) of that set. Then he could type ACCEPT,23$ to indicate that the new text is to be accepted just before line 23. Suppose he then types (not quite correctly):

```
CALL IDPOS('ABSL',120,1000,ELEMENT,I);
KALL IDTEXT('VLRG', 'AREA-ARC LENGTH CALCULATOR',ELEMENT,I);
```

As each line is being typed in, it appears at the bottom of the display page. Then, immediately after the text terminator is entered, the line appears on the page display in the appropriate place (in this case between lines 22 and 23). After the last text line is typed, the user escapes from text mode by typing E$ or an escape character. A typing error such as the one appearing on the second line of text just entered will usually be detected by the user as it appears on the bottom of the screen as he is typing it in, or as he reviews the entire insertion after it is displayed in the appropriate place. Suppose, however, that this error escapes such means of detection and that the user proceeds to compile the program by typing CONVSQ$ (to put the data set in a form acceptable to the SYSTEM/360 language processors) and PL/1$ (to invoke the PL/I compiler). When the PL/I compiler has finished processing the program, the completion code returned by the compiler and displayed by GSM would indicate a severity code unacceptable to

```
0001   EXAMPLE: PROCEDURE OPTIONS(MAIN);
0002
0003   DECLARE XGRID(24) FIXED BINARY STATIC INITIAL(262,762,762,262,262,
0004         762,762,262,262,762,762,262,262,262,362,362,462,462,562,562,
0005         662,662,762,762);
0006   DECLARE YGRID(24) FIXED BINARY STATIC INITIAL(200,200,300,300,400,
0007         400,500,500,600,600,700,700,700,200,200,700,700,200,200,700,
0008         700,200,200,700);
0009   DECLARE 1 ATTR,
0010             2 STRING CHARACTER(4),
0011             2 ARRAY(5) FIXED BINARY;
0012   DECLARE ELEMENT CHARACTER(1000) VARYING;
0013   DECLARE (XARRAY,YARRAY)(500) FIXED BINARY;
0014   DECLARE (IARRAY,MARRAY)(10) FIXED BINARY;
0015   DECLARE STRS CHARACTER(35);
0016   DECLARE (AREA,ARCL,X1,Y1,X2,Y2) FLOAT BINARY;
0017   DECLARE (NCURV,I,J,K) FIXED BINARY;
0018
0019         CALL IBOPEN(10,1,0);  /* INITIALIZE THE DISPLAY.  */
0020         CALL IBT(DEL(15);  /*  SET PEN TRACKING PARAMETERS.  */
0021         CALL IBT(BN)('DRAW');
0022         ELEMENT='';  /*  GENERATE TITLES AND GRID.  */
0133         CALL IBPOS('ABSL',120,1000,ELEMENT,I);
0134         CALL IBTEXT('VLRG','AREA-ARC LENGTH CALCULATOR',ELEMENT,I);
0023         CALL IBPOS('ABSL',200,875,ELEMENT,I);
0024         CALL IBTEXT('NEBN','SELECT ONE OF THE FOLLOWING',ELEMENT,I);
0025         CALL IBPOS('ABSL',XGRID(1),YGRID(1),ELEMENT,I);
0026         CALL IBPTLN('LINE',XGRID,YGRID,24,'10'B,ELEMENT,I);
0027         CALL IBEPUT(10,'NORM',ELEMENT);
0028         ELEMENT='';  /*  GENERATE ELEMENT 01.  */
0029         CALL IBPOS('ABSL',236,840,ELEMENT,I);
0030         CALL IBTEXT('NEBN','TERMINATE PROGRAM',ELEMENT,I);
0031         CALL IBEPUT(101,'NORMBETC',ELEMENT);
0032         ELEMENT='';  /*  GENERATE ELEMENT 02.  */
0033         CALL IBPOS('ABSL',236,805,ELEMENT,I);
0034         CALL IBTEXT('NEBN','CLEAR SCREEN',ELEMENT,I);
0035         CALL IBEPUT(102,'NORMBETC',ELEMENT);
0036         ELEMENT='';  /*  GENERATE ELEMENT 03.  */
0037         CALL IBPOS('ABSL',236,770,ELEMENT,I);
0038         CALL IBTEXT('NEBN','COMPUTE AREA AND ARC LENGTH',ELEMENT,I);
                                                                        5
BLOCKS 03450 03456  LINES=00134 PTRS=005  CAR  .......                1933A2
```

Figure 2: A Page of Text Ready for Editing

the user. Then he could visually page through the compiler output listings on the display using the VUELIST command, displaying each new page with a single keystroke on the IDIIOM (or with slightly more effort if a less suitable terminal such as the IBM 2250 model 2 or 2260 is used). If a printed copy is desired, it may be obtained by using the PRINT command. Having found the error on line 134 (the line number assigned to the second line of the insert by GSM) he could immediately type EDIT,EXAMPLE$ and CHANGE,134$ and the line in error would be presented for change. After he has changed the erroneous line, he could again type CONVSQ$ and PL1$.

When the user receives an acceptable completion code from the compiler, he can then type PLINK$ to linkage edit the compiled program using the PL/I subroutine library. If an unacceptably large severity code is returned by the linkage editor, he can again use the VUELIST command to examine the linkage edit map and diagnostics. Finally, he can execute the load module (in this case also named 'EXAMPLE') produced by the linkage editor by typing EXECUTE, EXAMPLE$ (or more briefly, G,EXAMPLE$).

For frequently used sequences of GSM commands, instead of typing in the commands one-by-one as they are to be executed, one may use a "stored program" approach by invoking a GSM procedure. For example, the sequence

```
EDIT,:1$
CONVSQ$
PL1$
PLINK$
EXECUTE,:1$
RETURN$
```

could have been generated using the same methods for editing text described above. It could have been assigned the name PL1GO; then to invoke it, the user would

merely type INVOKE, PL1GO, EXAMPLE$. In this case, ':1' is a formal parameter in the procedure and would be replaced by 'EXAMPLE' as the procedure was carried out. Since the GSM repertoire includes instructions for conditional branching within or between procedures and for invoking procedures from procedures, considerable flexibility exists in writing and using GSM procedures. For example, in PL1GO, instructions could have been included to test the completion code returned by the compiler or the linkage editor and inhibit the further execution of the procedure if an unacceptably high value was returned.

The ability to execute any load module provides a convenient user initiated extension to the GSM system. For example, one user load module named ASMERR scans assembly output listings for error indications, extracts the offending lines and the corresponding diagnostics, and prepares the information for display. Thus it is unnecessary to actually page through the listing output as described in the PL1 example above.

It is important to note that the page-oriented display features of a device like the IDIIOM or IBM 2250 make the system incomparably more powerful and efficient to use, from the user's point of view, than it would be if a line-at-a-time mechanical terminal such as a teletype or typewriter terminal were used. The principal economic factor limiting wider use of such systems has been the expense of such page-oriented terminals. It is hoped that the availability of lower priced drum/disk driven TV-scan type displays will soon provide a page-oriented capability at a more attractive price.

4. The IDIIØM Scope Package

A package of PL/I procedures, the IDIIØM Scope Package [4], has been provided for writing highly interactive programs without burdening the writer with all of the details of programming the 620/i and IDIIØM. The user of this

package can, by means of procedure calls, control the display console in addition to having all of the facilities of PL/I available to him. Similar packages are described in [5,6,7].

The basic items displayed by this package are called display elements. Procedures are supplied to construct elements which will display information on the CRT, and transmit elements between the SYSTEM/360 and 620/i. For example, the titles and grid shown in Fig. 3 were generated by the statements:

```
DECLARE ELEMENT CHARACTER(1000) VARYING;
ELEMENT=' ';
CALL IDPØS('ABSL',120,1000,ELEMENT,I);
CALL IDTEXT('VLRG','AREA-ARC LENGTH CALCULATOR',ELEMENT,I);
CALL IDPØS('ABSL',208,875,ELEMENT,I);
CALL IDTEXT('MEDM','SELECT ØNE ØF THE FØLLØWING',ELEMENT,I);
CALL IDPØS('ABSL',XGRID(1),YGRID(1),ELEMENT,I);
CALL IDPTLN('LINE',XGRID,YGRID,24,'10'B,ELEMENT,I);
```

The first line defines ELEMENT as a varying character string and the second line assures that it is empty. The calls to IDPØS, IDTEXT, and IDPTLN add positioning, text, and line display orders, respectively, to ELEMENT. The first argument in these procedure calls selects a specific type of order from the many possibilities; for example, the 'ABSL' parameter specifies that IDPØS is to generate absolute (and not relative) beam positioning orders. The variable I in these calls is an error flag which can be checked if there is a possibility that ELEMENT has overflowed. The arrays XGRID and YGRID define the twelve line segments in the grid.

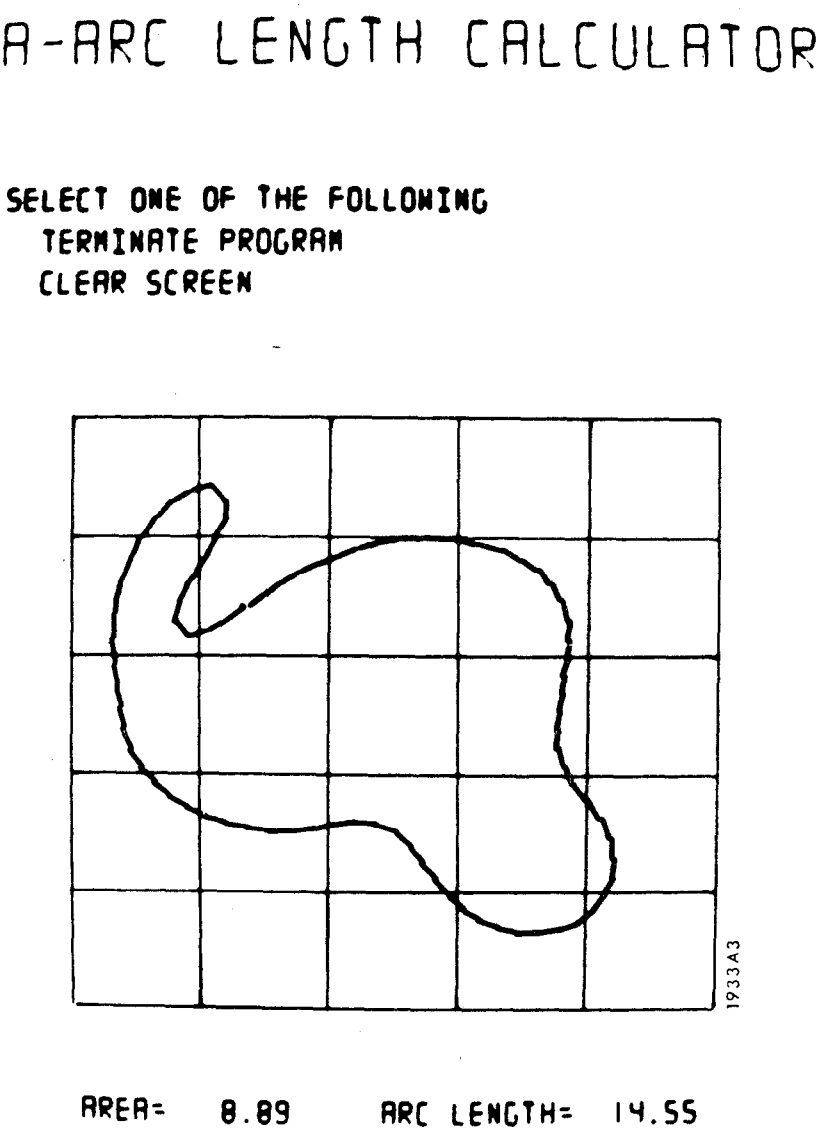

Figure 3: A Simple Interactive Display

When an element has been created, it may be transmitted to the 620/i where it will be added to the current display file. The previously defined element may be transmitted to the 620/i by writing:

```
CALL IDEPUT(10,'NØRM',ELEMENT);
```

This statement defines the element as a "normal" element with an identification number of ten. The identification number is used to manipulate the element in other sections of the program. For example:

```
CALL IDMØDX(10,'WINK');
```

will put element ten into the winking mode. Procedure IDMØDX may also change the intensity, line structure, or light pen sensitivity of an element, or delete it. For the convenience of the programmer, elements may be grouped together into sets. Sets may contain elements and other sets. A set may be manipulated in the same manner as an individual element.

A programmer may create four kinds of elements. A normal element consists of a sub-picture with which there is no interaction. A second type of element is a sub-picture positioned relative to the tracking cross. The light pen may then be used to move such an element around on the screen. A third type of element is a keyboard input buffer. When the console operator types on the keyboard, the characters will appear on the screen and be put into the keyboard input buffer. The final type of element is a light pen drawing buffer. When such an element is present, the console operator will be able to draw free-hand curves on the screen by moving the tracking cross with the light pen while depressing the switch on the light pen. The display orders for the free-hand curves are put into the light pen drawing buffer. The light pen may also be used to erase a curve, or a segment of a curve. This filling of keyboard input buffers and light pen drawing buffers is local to the 620/i, and this information will be transmitted

to the SYSTEM/360 only in response to an element read request. For instance, a statement such as:

```
CALL IDEGET(50,ELEMENT,I,J);
```

will transmit element fifty from the 620/i to the SYSTEM/360. Procedures are available to help the programmer interpret the contents of these buffer elements.

In addition, an element may act as a graphic subroutine. It will not be displayed on its own but will be displayed only when it is invoked by another element. Such an element will be in the display file once, but the picture it describes may appear in several places on the CRT. The procedure IDEXELT is provided to invoke an element as a subroutine. For instance the statements:

```
CALL IDPØS('ABSL',100,700,ELEMENT,I);
CALL IDEXELT(40,ELEMENT,I);
```

will cause orders to be inserted into ELEMENT. These orders position the beam and invoke the element whose identification is forty. The beam could then be re-positioned and element forty again invoked to place a second copy of element forty on the screen. The program in the 620/i is responsible for the final resolution of the call to element forty.

Procedures are also available to enable or disable interrupts from the 620/i and IDIIØM. For instance the statement:

```
CALL IDEATTN('PFKM LPDT','STØP');
```

will enable interrupts from the function keyboard (PFKM) and the light pen (LPDT). The second argument indicates that the display is to be turned off when the interrupt is reported. After this statement is executed, the use of either of these devices will cause an interrupt to be sent to the SYSTEM/360. In the SYSTEM/360, the record of the interrupt may either be queued, or cause a pre-selected PL/I procedure to be executed asynchronously. A procedure is available to check the interrupt queue.

Some interrupts, such as those from the function keyboard, are generated by hardware on the 620/i and, if enabled, are passed on to the SYSTEM/360. Other interrupts are generated by the program in the 620/i. For example, an interrupt may be generated each time the display has refreshed a specified number of times. This interrupt has proven to be very useful in adjusting the exposure of the motion picture film in animation mode.

Another important interrupt may be generated by the light pen in a special mode. In this mode a light pen sensitive element will brighten when the light pen is pointing at it. An interrupt is sent to the SYSTEM/360 only when the light pen switch is depressed. The information made available to the PL/I program, in addition to interrupt type, is: (1) the identification number of the element, (2) the index of the display order within the element that generated the specific character, point, or line at which the light pen was pointing, and (3) the X and Y coordinates of that item.

Fig. 3 illustrates a display produced by a simple program (the first part of the program is shown in Fig. 2). The initial display included a tracking cross and a third message "CØMPUTE AREA AND ARC LENGTH" under the "CLEAR SCREEN" message. The console operator used the light pen to draw a closed curve on the CRT, and then selected the third message. The program in the SYSTEM/360 responded by reading the curve, computing its area and arc length, deleting the third message and the tracking cross, and adding the message at the bottom of the screen. The console operator may then either select "CLEAR SCREEN" to return the display to its original state or he may select "TERMINATE PRØGRAM". The program which performed these actions consists of 122 PL/I statements.

5. The Animation Generation System

As experience was gained with the procedures of the previous section, it became clear that the task of writing programs to generate animation in 2 or 3 dimensions could be greatly simplified. In particular, a programmer should be able to define a sequence of 3-dimensional scenes and subsequently view it on the CRT in the usual manner or with the 3-D viewer. He also should be able to record the sequence on film with the movie camera for projection as a 2-D or 3-D movie. 3-D movies are generated by putting left and right eye images on alternate frames and then merging the film through an optical printer to obtain film which may be projected for an audience wearing Polaroid viewing glasses. Multi-color animated films may be made by recording each color on a separate black and white film strip and then printing these film strips on color film using the appropriate color filters.

This animation system [8] continues the spirit of the system described by Knowlton [9] in that an animation sequence is described by writing a program. Other systems [10,11,12] give the console operator the ability to define animation sequences by typing commands and drawing on the CRT. There are two reasons why we have not adopted this latter scheme. (1) These other systems tend to be written with the professional film maker in mind. Our system was written for scientifically oriented people who wish to visualize a mathematical theory of a phenomenon, or study a large collection of data which measures a phenomenon. (2) The Animation Generation System, in conjunction with the Graphic Sub-Monitor, permits new programs to be written, and existing programs to be changed, in a very easy and straightforward manner.

The system provides a programmer with a simple means of defining a geometric figure in space. The geometric figure may be anything that can be

constructed from lines and points. For added realism, surfaces may be defined and the hidden lines removed from the figure. To specify the animation sequence, a conceptual camera is positioned in space pointing at the geometric figure, and a picture is taken. Between successive pictures in the animation sequence, the camera position may be changed, the geometric figure changed, or both may be changed.

We have chosen to implement this system utilizing the compile-time preprocessor of PL/I. This facility of PL/I is similar in concept to the macro language supplied with most assemblers. They both may be used by the programmer to extend the power of the base language within the confines of a fairly rigid syntax.

An example of a simple program written in this extended PL/I is shown in Fig. 4. The two statements on lines 13, 14, and 15 are similar to the declaration statements of PL/I. The %ØPTIØNS= statement is used by the programmer to optimize compilation and execution speed, provide for the loading of the proper hidden line procedure, and other miscellaneous tasks. The %GEØMETRY= statement is used to declare the geometric variables to be used in the program. The geometric variables are points, lines, surfaces, objects, and views. An object is a collection of points, lines, and surfaces, while a view defines a camera position. In general, the declaration of a geometric variable consists of a keyword followed by all of the variables of that type. For instance the phrase:

PØINTS=PT(32),Q,R(3,4)

is a valid declaration of geometric points. The phrases for the different types of geometric variables are separated by ampersands.

The animation system provides the programmer with a simple means of asking the console operator to enter information with the keyboard or light pen. The keyboard input request is made by putting keyboard input buffers on the CRT

```
  /*****  STELLATED DODECAHEDRON GENERATOR  *****/
RCBMLT2: PROCEDURE OPTIONS(MAIN);

/*  THIS PROGRAM PRODUCES AN ANIMATION SEQUENCE
    SHOWING A STELLATED DODECAHEDRON ROTATING IN
    SPACE.  HIDDEN LINES IN THE FIGURE ARE
    REMOVED.  A STELLATED DODECAHEDRON IS THE
    SOLID FIGURE WHICH IS GENERATED BY EXTENDING
    THE FACES OF A REGULAR DODECAHEDRON.          */

%INCLUDE HEAD1;
%OPTIONS=' MAIN & RHL=SOLID ';
%GEOMETRY=' POINTS=PT(32) & LINES=LN(90) &
  SURFACES=SU(60) & OBJECTS=OB & VIEWS=VW ';
DECLARE PX(32) FLOAT BINARY STATIC INITIAL(
   0.2764,-0.1056,-0.3416,-0.1056, 0.2764, 0.5528,
   0.1708,-0.4472,-0.4472, 0.1708, 0.4472,-0.1708,
  -0.5528,-0.1708, 0.4472, 0.3416, 0.1056,-0.2764,
  -0.2764, 0.1056, 0.0000, 0.8944, 0.2764,-0.7236,
  -0.7236, 0.2764, 0.7236,-0.2764,-0.8944,-0.2764,
   0.7236, 0.0000);
DECLARE PY(32) FLOAT BINARY STATIC INITIAL(
   0.2008, 0.3249, 0.0000,-0.3249,-0.2008, 0.0000,
   0.5257, 0.3249,-0.3249,-0.5257, 0.3249, 0.5257,
   0.0000,-0.5257,-0.3249, 0.0000, 0.3249, 0.2008,
  -0.2008,-0.3249, 0.0000, 0.0000, 0.8507, 0.5257,
  -0.5257,-0.8507, 0.5257, 0.8507, 0.0000,-0.8507,
  -0.5257, 0.0000);
DECLARE PZ(32) FLOAT BINARY STATIC INITIAL(
  -0.4472,-0.4472,-0.4472,-0.4472,-0.4472, 0.1056,
   0.1056, 0.1056, 0.1056, 0.1056,-0.1056,-0.1056,
  -0.1056,-0.1056,-0.1056, 0.4472, 0.4472, 0.4472,
   0.4472, 0.4472,-1.0000,-0.4472,-0.4472,-0.4472,
  -0.4472,-0.4472, 0.4472, 0.4472, 0.4472, 0.4472,
   0.4472, 1.0000);
DECLARE L1(90) FIXED BINARY STATIC INITIAL(
    5, 1, 2, 3, 4, 1, 2, 3, 4, 5, 6,11, 7,12, 8,13,
    9,14,10,15, 6, 7, 8, 9,10,16,17,18,19,20,21,21,
   21,21,21,22,22,22,22,22,23,23,23,23,23,24,24,24,
   24,24,25,25,25,25,25,26,26,26,26,26,27,27,27,27,
   27,28,28,28,28,28,29,29,29,29,29,30,30,30,30,30,
   31,31,31,31,31,32,32,32,32,32);
DECLARE L2(90) FIXED BINARY STATIC INITIAL(
    1, 2, 3, 4, 5,11,12,13,14,15,11, 7,12, 8,13, 9,
   14,10,15, 6,16,17,18,19,20,17,18,19,20,16, 1, 2,
    3, 4, 5, 5, 1,11, 6,15, 1, 2,12, 7,11, 2, 3,13,
    8,12, 3, 4,14, 9,13, 4, 5,15,10,14,16,17, 7,11,
    6,17,18, 8,12, 7,18,19, 9,13, 8,19,20,10,14, 9,
   20,16, 6,15,10,16,17,18,19,20);
DECLARE S1(60) FIXED BINARY STATIC INITIAL(
    1, 2, 3, 4, 5, 1, 6,11,20,10, 2, 7,13,12, 6, 3,
    8,15,14, 7, 4, 9,17,16, 8, 5,10,19,18, 9,11,12,
   22,26,21,13,14,23,27,22,15,16,24,28,23,17,18,25,
   29,24,19,20,21,30,25,26,27,28,29,30);
DECLARE S2(60) FIXED BINARY STATIC INITIAL(
   35,31,32,33,34,37,38,39,40,36,42,43,44,45,41,47,
   48,49,50,46,52,53,54,55,51,57,58,59,60,56,64,63,
   62,61,65,69,68,67,66,70,74,73,72,71,75,79,78,77,
   76,80,84,83,82,81,85,87,88,89,90,86);
DECLARE S3(60) FIXED BINARY STATIC INITIAL(
   31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,
   47,48,49,50,51,52,53,54,55,56,57,58,59,60,65,64,
   63,62,61,70,69,68,67,66,75,74,73,72,71,80,79,78,
   77,76,85,84,83,82,81,86,87,88,89,90);
DECLARE DOBC CHARACTER(6) INITIAL('  9.00');
DECLARE DSCR CHARACTER(6) INITIAL(' 10.00');
DECLARE EYES CHARACTER(6) INITIAL('  0.70');
DECLARE SCRZ CHARACTER(6) INITIAL('  2.50');
DECLARE FPIC CHARACTER(6) INITIAL('    24');
DECLARE MPIC CHARACTER(6) INITIAL('   240');
DECLARE RSTP CHARACTER(6) INITIAL('  1.00');
DECLARE (XDOBC,XDSCR,XEYES,XSCRZ,XRSTP)
  FLOAT BINARY, (XFPIC,XMPIC) FIXED BINARY;
DECLARE (V1,V2,V3) (3) FLOAT BINARY;
DECLARE (ARG,S,C) FLOAT BINARY;
DECLARE (I,J) FIXED BINARY;
%INCLUDE HEAD2;

/*  TWO INTERACTIVE DISPLAYS ARE CREATED TO
    ALLOW THE CONSOLE OPERATOR TO ENTER
    PARAMETERS THROUGH THE KEYBOARD.  THE FIRST
    DISPLAY GIVES HIM THE OPTION OF CHANGING
    PARAMETERS ASSOCIATED WITH CAMERA POSITIONING
    WHILE THE SECOND DISPLAY LETS HIM CHANGE
    PARAMETERS WHICH CONTROL THE NUMBER OF FRAMES
    PRODUCED.                                     */
DSP1: DSPINIT;
      DSPTEXT(190,700,CPARM=VLRG,
        'ENTER CAMERA PARAMETERS');
      DSPTEXT(260,600,VSPAC=-35,
        'DISTANCE TO CENTER OF OBJECT',
        'DISTANCE TO THE SCREEN',
        'EYE SEPARATION (STEREO ONLY)',
        'SCREEN SIZE');
      DSPKBRD(680,600,,DOBC,DSCR,EYES,SCRZ);
      ON CONVERSION GO TO DSP1;
      GET STRING(DOBC) LIST(XDOBC);
      GET STRING(DSCR) LIST(XDSCR);
      GET STRING(EYES) LIST(XEYES);
      GET STRING(SCRZ) LIST(XSCRZ);
DSP2: DSPINIT;
      DSPTEXT(190,700,CPARM=VLRG,
        'ENTER MOTION PARAMETERS');
      DSPTEXT(260,600,VSPAC=-35,
        'START/END FIXED PICTURES',
        'NUMBER OF MOVING PICTURES',
        'ROTATION (DEGREES/PICTURE)');
      DSPKBRD(680,600,,FPIC,MPIC,RSTP);
      ON CONVERSION GO TO DSP2;
      GET STRING(FPIC) LIST(XFPIC);
      GET STRING(MPIC) LIST(XMPIC);
      GET STRING(RSTP) LIST(XRSTP);

/*  THE GEOMETRIC FIGURE IS NOW DEFINED.          */
      DO I=1 TO HBOUND(PT,1);
        V1(1)=PX(I); V1(2)=PY(I); V1(3)=PZ(I);
        POINT(PT(I),,V1);
      END;
      DO I=1 TO HBOUND(LN,1);
        LINE(LN(I),,PT(L1(I)),PT(L2(I)));
      END;
      DO I=1 TO HBOUND(SU,1);
        SURFACE(SU(I),,LN(S1(I)),LN(S2(I)),
          LN(S3(I)));
        OBJECT(OB,EXPAND,SU(I));
      END;

/*  THE ANIMATED SEQUENCE IS NOW PRODUCED.  THE
    FIRST AND LAST FRAMES ARE REPEATED XFPIC
    TIMES.                                        */
      LEADER;
      DO I=1 TO XMPIC;
        ARG=(I-1)*XRSTP; S=SIND(ARG); C=COSD(ARG);
        V2(1)=-C*C;          V3(1)=S*C;
        V2(2)=S*C*(1+S);     V3(2)=C*C-S*S*S;
        V2(3)=S*(C*C-S);     V3(3)=-S*C*(1+S);
        V1=-XDOBC*V2;
        VIEW(VW,,V1,V2,V3,XEYES,XDSCR,XSCRZ);
        PICTURE(OB,VW,NGC&RHL=SOLID);
        IF (I=1)|(I=XMPIC) THEN REPEAT(XFPIC);
      END;
      LEADER;

%INCLUDE TAIL;

END RCBMLT2;
```

1933A4

Figure 4: A Complete Animation Generation Program

and waiting for the console operator to make any necessary changes. When the console operator signals that he has finished, by generating a keyboard interrupt, the modified character strings are transmitted to the SYSTEM/360 where it is the program's responsibility to process them. In figure 4, lines 88 through 96 generate a display containing descriptive text and the four character strings in lines 66 through 69. The console operator may change these four strings and then generate a keyboard interrupt. These four character strings are then read from the 620/i into the SYSTEM/360 where statements 98 through 101 convert the strings to floating binary numbers.

A point is defined by specifying its 3-dimensional cartesian coordinates. In lines 117 and 118 of Fig. 4, the X, Y, and Z coordinates of a point are selected out of the arrays PX, PY, and PZ; and a point is defined. Line 121 shows a line being defined by specifying its end points. The arrays L1 and L2 contain the indices of the end points of each of the 90 lines in the figure. Thus, the first line is the one between PT(5) and PT(1). The missing second argument of the POINT and LINE statement may be used to specify a color code for color animated movies.

A surface is defined by giving a maximum of ten boundary lines which must form a planar convex polygon. The more efficient hidden line removal algorithms in this system, those that work on solid objects, require that the boundary lines be listed in a counter-clockwise direction as seen from the outside of the figure. Lines 124 and 125 show a triangular surface being defined using the arrays S1, S2, and S3 to supply the indices of its boundary lines. The first such surface is the one bounded by LN(1), LN(35), and LN(31). The statement for defining a geometric object may be used to re-define or expand (line 126) the definition of an object.

The definition of a view (line 139) is not as simple as the other geometric items. The form of the VIEW statement is:

VIEW(GVIEW,,R,V,H,e,d,f) ;

where GVIEW is the geometric view being defined. The parameters which define the view (see Fig. 5) are:

R: A reference point which is, conceptually, a point midway between the eyes of the viewer.

V: A vector in the direction that the viewer is looking.

H: A vector, perpendicular to V, defining the horizontal direction of the projection screen. This vector is optional; if it is not given, the system will generate a vector parallel to the X-Y plane.

d: The distance from R to the projection screen.

e: The distance between the eyes of the viewer.

f: The size of the projection screen.

Other items of interest in Fig. 5 are the points P and Q. P represents a point in the object being viewed and Q is the projection of P (from E) onto the screen. For a stereo picture, the point P is projected twice, once from E and once from E'. For a two dimensional picture P is projected from R. If P is between R and the screen, then it will appear to float in space in front of the CRT or motion picture screen when viewed in a 3-D mode. The mathematical details of how this information is used are described in the Appendix.

The last step in an animation generation program is that of producing the animated sequence. To do this, the statements LEADER, PICTURE, and REPEAT are provided. In Fig. 4, the LEADER statement (lines 132 and 143) initializes the system and produces a few frames of leader when the program is in a film

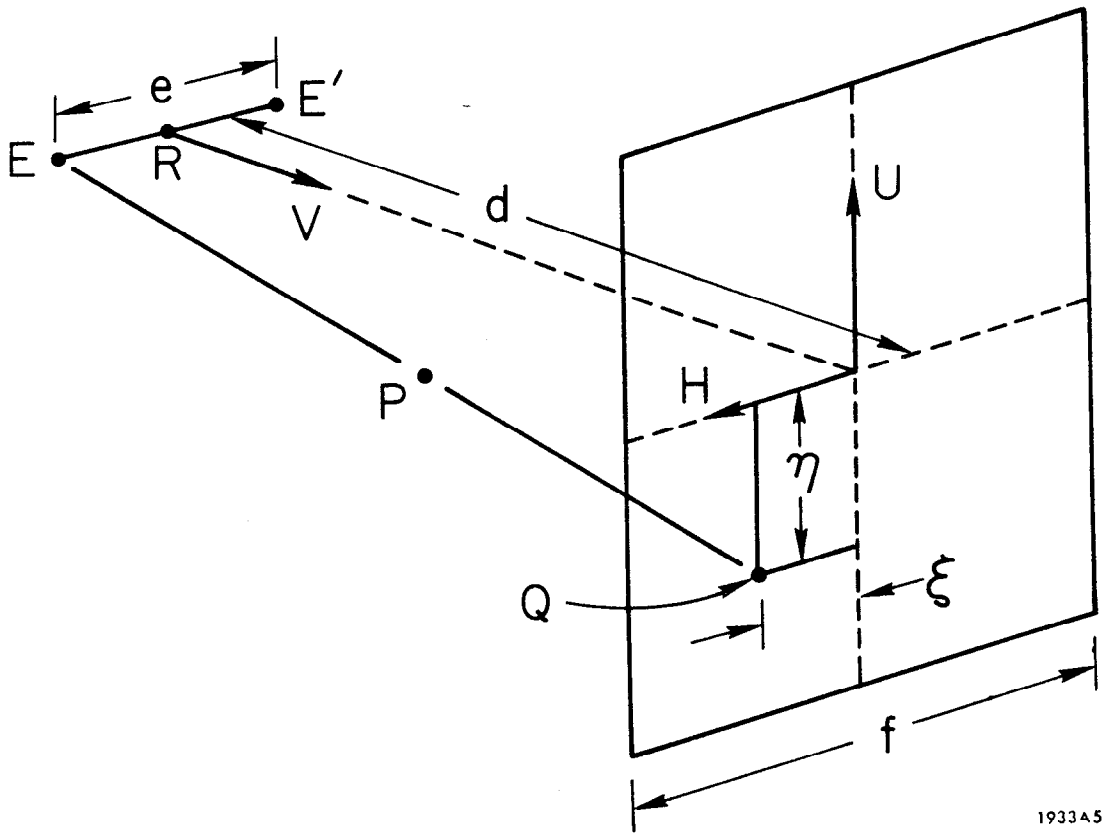

Figure 5: The Geometry of the VIEW Statement

producing mode. The PICTURE statement (line 140) accepts as input an object, a view, and a list of options; up to three such triples may be specified in a single PICTURE statement. This statement causes the geometric data to be processed and a picture to be transmitted to the 620/i. The options list is used to optimize the computation and select a hidden line algorithm if hidden lines are to be removed. In an animation sequence it is often desirable to stop the action for short periods. The REPEAT statement (line 141) is provided for this purpose. It simply repeats the last display generated by the PICTURE statement a specified number of times.

Three hidden line algorithms are currently available in this programming system. They are selected in the PICTURE statement by coding RHL=CØNVEX, RHL=SØLID, or RHL=GENERAL. The first algorithm is very fast and efficient but assumes that the object is a single convex body. It is based on the fact that an edge of a convex body is completely invisible if it is a boundary of two back faces, and is completely visible otherwise. A back face is one whose surface normal points away from the viewer, otherwise the surface is a front face. Unfortunately, most objects of interest are not convex. The second algorithm assumes that the object consists of a number of solid bodies; that is, the surfaces define bodies with an inside and an outside. The object may also contain miscellaneous points and lines which will be processed by the hidden line eliminators. Some efficiency is retained in this algorithm because only front faces need be considered in determining how much of a line is hidden. In addition, an edge is completely invisible if it is the boundary of two back faces, or the boundary of a front and back face with the back face closer to the viewer. Fig. 6 was produced from the program in Fig. 4 using this algorithm. The third hidden line algorithm makes no geometric assumptions and can process any collection of lines, points, and surfaces positioned arbitrarily in space. Since it compares each line or point with every surface, it can be very time consuming.

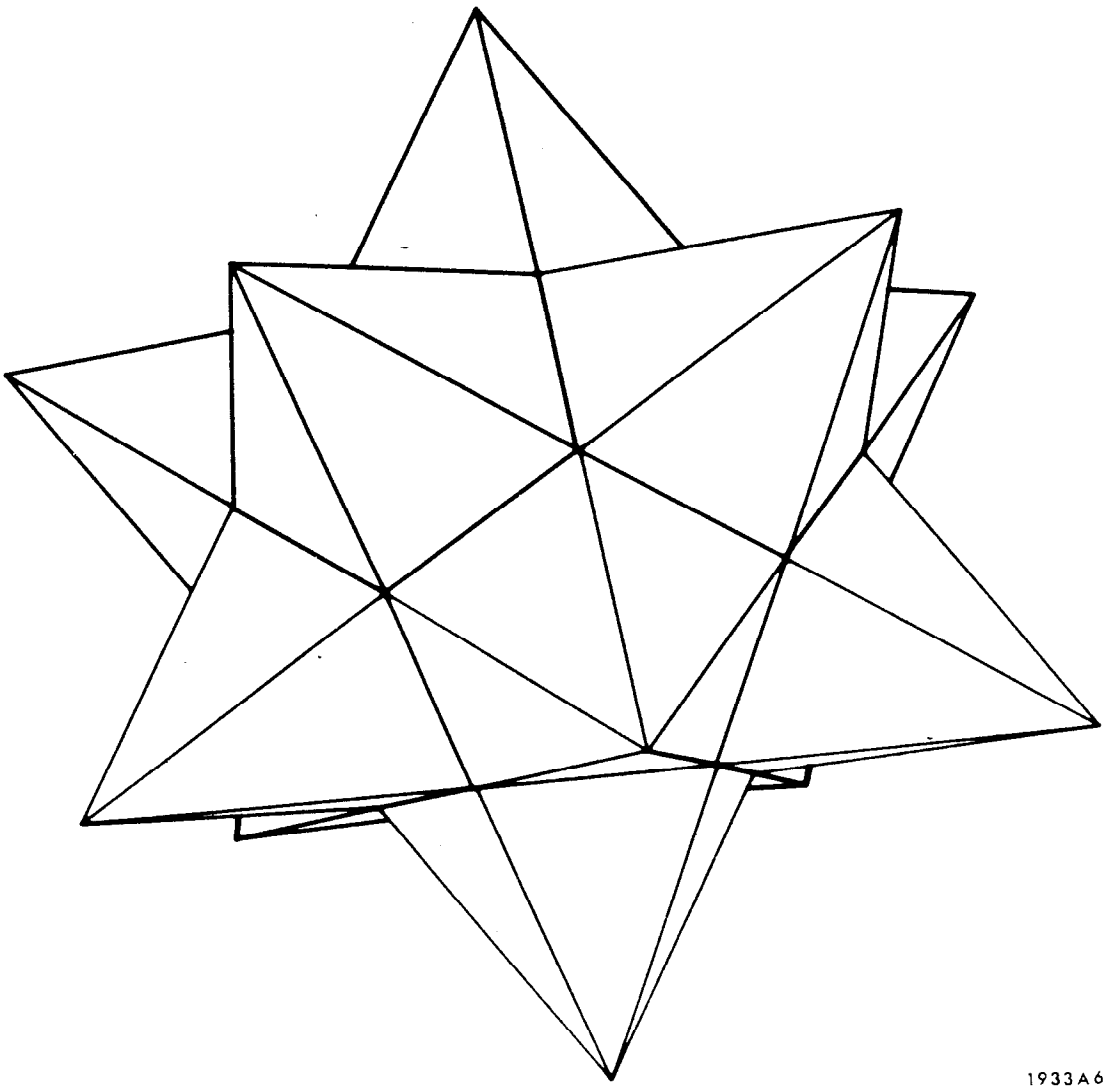

Figure 6: The Stellated Dodecahedron

Other algorithms could also be added to this system. One which is efficient requires that the programmer break the object up into smaller convex bodies. While such a scheme puts a greater burden on the programmer, it results in an algorithm with most of the good features of the RHL=SØLID algorithm in addition to the ability to recognize an edge as being fully visible very early in the algorithm.

Although the animation generation program in Fig. 4 appears to generate the animation sequence only once, this is not the case. A part of the expansion of the %INCLUDE TAIL statement (line 145) is a GØ TØ statement which transfers control into part of the %INCLUDE HEAD2 statement (line 78). The expansion of the %INCLUDE HEAD2 statement contains statements to put an initial display on the CRT. This display gives the console operator the ability to change certain parameters and select the mode in which the animated sequence is to be run. Thus, an animation sequence may be run through many times, perhaps in different modes, during one execution of an animation generation program.

Appendix: The Projection Matrix

One of the basic repetitive operations is the operation of transformation of a point (X, Y, Z) in the model space into screen coordinates (μ, ν). In this system the data supplied to the VIEW macro is first used to construct a 3x4 matrix M. This matrix has the property that the coordinate transformation may be carried out by:

$$\lambda \begin{pmatrix} \mu \\ \nu \\ 1 \end{pmatrix} = M \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \qquad (1)$$

If equation 1 is expanded into three scalar equations, these equations will define $\lambda\mu$, $\lambda\nu$, and λ. If the first two equations are divided by the third, the result is a pair of rational expressions for μ and ν. Alternatively the transformation can be carried out by a single matrix multiplication and two divisions. In Fig. 5, U is a vector which defines the upward direction of the screen. In the following, the vectors V, H, and U are assumed to be unit vectors. The variables ξ and η measure distance, along H and U respectively, of the projected point Q.

From Fig. 5, it is evident that E and Q may be given by:

$$E = R + \frac{e}{2} H \qquad (2)$$

$$Q = R + d\ V + \xi\ H + \eta\ U$$

where (ξ, η) is related to (μ, ν) by:

$$\xi = \frac{f}{1024} (\mu - 512) \qquad (3)$$

$$\eta = \frac{f}{1024} (\nu - 512)$$

When working with the left eye projection of a stereo view, it is sufficient to replace e by -e in equation (2) and all following equations. For a simple 2-D projective view set e to zero. Since E, P, and Q lie on a straight line, the vectors (P-E) and (Q-E) are parallel, and therefore are proportional.

$$\lambda\ (Q-E) = (P-E) \qquad (4)$$

Substituting equations 2 into the left hand side of equation 4 results in:

$$\lambda \left[\left(\xi - \frac{e}{2} \right) H + \eta\ U + d\ V \right] = (P-E) \qquad (5)$$

Equation 5 may be rewritten in the form:

$$\lambda M_1 \begin{pmatrix} \xi - \frac{e}{2} \\ \eta \\ d \end{pmatrix} = (P-E) \qquad (6)$$

$$M_1 = (H \mid U \mid V)$$

where the columns of M_1 are H, U, and V. Since the 3x3 matrix M_1 is orthogonal, its inverse is equal to its transpose. Thus, multiplication of equation 6 by the inverse of M_1 gives:

$$\lambda \begin{pmatrix} \xi - \frac{e}{2} \\ \eta \\ d \end{pmatrix} = M_2(P-E) \qquad M_2 = \begin{pmatrix} H^T \\ U^T \\ V^T \end{pmatrix} \qquad (7)$$

where the rows of M_2 are H^T, U^T, and V^T. Now substitute equations 3 into equations 7 to obtain:

$$\lambda \begin{pmatrix} \frac{f}{1024}\ \mu - \frac{f+e}{2} \\ \frac{f}{1024}\ \nu - \frac{f}{2} \\ d \end{pmatrix} = M_2(P-E) \qquad (8)$$

To simplify future computation, replace the indeterminate λ by $1024\ \lambda/f$. When this is done, equation 8 becomes:

$$\lambda \begin{pmatrix} \mu - \frac{512(f+e)}{f} \\ \nu - 512 \\ \frac{1024d}{f} \end{pmatrix} = M_2(P-E) \qquad (9)$$

The column vector on the left hand side of equation 9 may be written as:

$$\begin{pmatrix} 1 & 0 & -\frac{512(f+e)}{f} \\ 0 & 1 & -512 \\ 0 & 0 & \frac{1024d}{f} \end{pmatrix} \begin{pmatrix} \mu \\ \nu \\ 1 \end{pmatrix} \qquad (10)$$

Now multiply equation 9 on the left, by the inverse of the 3x3 matrix in equation 10. The result of this is:

$$\lambda \begin{pmatrix} \mu \\ \nu \\ 1 \end{pmatrix} = M_3 (P-E) \qquad (11)$$

$$M_3 = \begin{pmatrix} 1 & 0 & \frac{f+e}{2d} \\ 0 & 1 & \frac{f}{2d} \\ 0 & 0 & \frac{f}{1024d} \end{pmatrix} M_2$$

Finally, the first of equations 11 may be written as equation 1 where:

$$M = (M_3 \mid (-M_3 E)) \qquad (12)$$

Equation 12 states that the first three columns of M are the same as those of M_3 and the last column of M is the column vector $-M_3 E$.

REFERENCES

1. Varian Data 620/i Computer Manual. Bulletin No. 605-A, Varian Data Machines, 2722 Michelson Drive, Irvine, California 92664.
2. IDIIØM Technical Description. Information Displays, Inc., 333 North Bedford Road, Mount Kisco, New York 10549.
3. Robinson, G.A. Preliminary GSM-3 Command Manual - Computation Group Technical Memo #122. Stanford Linear Accelerator Center, Stanford, California 94305.
4. Beach, R.C. The SLAC Scope Package for the IDIIØM - Computation Group Technical Memo #80. Stanford Linear Accelerator Center, Stanford, California 94305.
5. Graphic Subroutine Package (GSP) for FØRTRAN IV, CØBØL, and PL/1. Form C27-6932, IBM Programming Systems Publications, Poughkeepsie, New York 12401.
6. The SLAC 2250 Scope Package - FØRTRAN Version, Library Program No. KO-59. Stanford University Computation Center, Stanford, California.
7. The SLAC 2250 Scope Package - PL/1 Version, Library Program No. KO-110. Stanford University Computation Center, Stanford, California.
8. Beach, R.C. A Programming System for Producing Computer Generated Motion Pictures on the Graphic Interpretation Facility - Computation Group Technical Memo. #123. Stanford Linear Accelerator Center, Stanford, California 94305.
9. Knowlton, K.C. A Computer Technique for Producing Animated Movies. Proc. AFIPS 1964 SJCC Vol. 25, Spartan Books, New York, pp 67-87.
10. Talbot, P.A. et. al. Animator: An On-Line Two-Dimensional Film Animation System. Comm. ACM, Vol. 14, April 1971, pp 251-259.

11. Baecker, R. M. Interactive Computer-Mediated Animation, Ph. D. Thesis, MIT, Dept. of Electrical Engineering, Cambridge, Mass., June 1969.

12. Gracer, F., Blasgen, M.W. Karma: A System for Storyboard Animation. Computer Graphics - A Quarterly Report of SIGGRAPH, Vol. 5, Winter-Spring 1971, pp 26-36.