THE DISPLAY OF COMPUTER GENERATED SHADED COLOR IMAGES FROM POLYGON DATA AND ITS APPLICATION IN THE MECHANICAL ENGINEERING COMPUTER-AIDED DESIGN ENVIRONMENT

by

Jan Carl Silverman

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Computing Science

Lehigh University

1982

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

Sept 8, 1982
(date)

Samuel L. Guld
Professor in Charge

Gerhard Rayna
Head of Division

ACKNOWLEDGMENTS

I am particularly thankful to Professor Samuel L. Gulden for his constant help and confidence. His guidance and support allowed me to achieve the goals I had set for myself. Also his patience in reading the drafts of this document and the comments he made were deeply appreciated.

I would especially like to thank Professor John B. Ochs for allowing me to do my research in the Mechanical Engineering Computer-Aided Design Laboratory, and treating me like a student in his department.

I would also like to thank Professor Tulga Ozsoy for the many discussions we had on his Mandrall model, and Jim Swain who created the Tie Rod model.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure No. Page

ABSTRACT

The purpose of this work is to clarify some techniques in the generation of shaded color images on a color raster terminal. The implementation of this capability is integrated into the existing software at the Mechanical Engineering Computer-Aided Design facility at Lehigh University.

The MOVIE.BYU program from Brigham Young University, was used on a basis for the shading algorithms. In order to display the output from this program, device drivers were written for the Digital Equipment Corporation's VS11 color raster terminal. The VS11 terminal can only display 16 colors, while for realistic images the output of MOVIE.BYU provides 256 shades of a single color. Consequently, a blending of the possible VS11 colors in a patterning effect was introduced into the program. In addition, to make the complexities of MOVIE.BYU transparent to users, some of its more general and complex features were hidden through the interfacing of a simpler command structure.

In making these additions, major modifications were performed on MOVIE.BYU. These modifications and the device drivers are documented in this paper.

The Computer-Aided Design Laboratory had a need to display models generated from the McAuto Unigraphics Design System as a shaded color object. The data from these models could be presented

in finite element form; i.e. three- or four-noded polygons, or six-, eight-, fifteen-, or twenty-noded volume elements. An additional feature of this paper is an interface program which transforms the data in any of the above finite element forms into a polygonal representation for MOVIE.BYU input.

CHAPTER I

INTRODUCTION

The procedure that was developed by the author at Lehigh University for generating a shaded color image from a geometric model involves the following steps. A model is created on the McAuto Unigraphics design station and a finite element mesh is applied to it. An interface program UGFEMBYU is run to transfer the output data from Unigraphics to a form compatible with MOVIE.BYU.

MOVIE.BYU is then run on the VS11 color terminal. The output is first displayed in the line drawing mode, using no hidden line removal or shading, and oriented to its desired position. The object is then viewed as a shaded image following a modified set of MOVIE.BYU commands and prompts.

A description of the hardware and the theory of displaying shaded images is discussed. The device drivers, interface program and command structure modifications to MOVIE.BYU are presented. This paper concludes with a sample demonstration run.

CHAPTER II

DESCRIPTION OF HARDWARE

VAX 11/780

The computer at the Mechanical Engineering Computer-Aided Design Laboratory is Digital Equipment Corporation's VAX 11/780 (5). It has a word and addressing size of 32 bits. The VAX is currently equipped with two megabytes of core and two 300 megabyte disk drives. The VAX runs the virtual operating system VAX/VMS with a page size of 512 bytes. Six VS11 color raster terminals are interfaced to it and operate at channel speed. This architecture is compatible with MOVIE.BYU's required 32 bit word size. With virtual memory it is possible to create large models composed of many polygons. The system is currently set up to handle models composed of up to 3,000 polygons.

VS11 Color Raster System

The VS11 color raster terminal currently used at Lehigh University is a 19-inch RGB (red, green, blue) color CRT which can display 16 colors. It has a visible resolution at 512 x 480 pixels. The color values for each pixel is defined in 4 bits. It also has incorporated into it a VT100 alphanumeric terminal. This feature is functionally separate from the VS11 graphic capability and just uses the same monitor for output.

The VS11 color raster terminal is part of the VSV11/VS11 video graphics system from Digital Equipment Corporation. It consists of a display processor (high speed microprocessor), a graphic instruction set, image memory, joystick and sync generator. The VSV11/VS11 video supports two 32K "segments" of Display File Memory called Main and Auxiliary. The Main segment would normally contain the display file, and the Auxiliary segment would normally contain a library of subroutines or image data. The display processor will sequence through either of these segments with the Display Program Counter register containing the virtual address of the next memory location. The system operates as part of the VAX/VMS executive and is incorporated into it by SYSGEN procedures. For a complete specification and interface guide to the VAX refer to "VS/VSV11 VAX/VMS Version 2.0 S/W Driver's Users Guide" (7).

In order to display an image on the VS11, the data must be organized into a display file in the memory of the VAX. This file consists of a list of VS11 instructions (graphic, control and data), which define the image. These are sixteen bit instructions which tell the display processor what action to take. The Display File starting address is then moved to the Display Program Counter. The microprocessor then sequences through the Display File instructions and generates the desired image on the monitor. A complete technical user manual is available from Digital Equipment Corporation (6). A list of the common graphic instructions, their opcodes and bit patterns, and their implementation will be discussed in Chapter IV.

CHAPTER III

MOVIE.BYU DESCRIPTION AND THEORY

Functional Description

MOVIE.BYU (2) is a group of Fortran programs used for the display and manipulation of data in the form of N-sided polygons, solid elements or contour lines. This data could represent architectural, topological or mathematical models. MOVIE.BYU is used at Lehigh University's Mechanical Engineering Computer-Aided Design Laboratory for the shaded color display of Polygonal Element Data. This data is generated from a finite element description of a geometric model, created by the design capabilities of McAuto's Unigraphics Design System (10).

MOVIE.BYU requires a computer with at least a 32-bit word size, though a 16-bit version of MOVIE.BYU with fewer capabilities is available (4). A complete description and excellent training manual of its operation is available from Hank Christiansen at Brigham Young University (3).

The modules of MOVIE.BYU that are used in this paper are: COMMAND.FOR, HIDDEN.FOR, AND DEVICE.TK4., which are modified and renamed COMMANDLU.FOR, HIDDENLU.FOR, and DEVICEVS.FOR. They are compiled and linked together with the MACRO's VSA32768 and ASCII to form DISPLAY.EXE. These modules provide the capabilities for displaying an object as a line drawing, with or without hidden lines removed, or as a continuous shaded color image.

COMMAND.FOR is the interactive command processor. It provides a four letter key word command structure. Some of its capabilities include:

- Global or local rotations and translations
- Color selection of the background and individual parts
- Choice of shading parameters
  1. Uniform over the polygon surface (UNIFORM)
  2. Vary linearly over the polygon but not matching at the polygon boundaries (FLAT)
  3. Vary linearly over the polygon with shading matched at the boundaries (SMOOTH)
- Movable light source
- Adjustment of the intensity of the object's highlights and its functional variation with the angle between the reflected light and the observer.

COMMAND.FOR had to be modified to handle subroutine calls to the VS11. This involved rewriting some of the subroutines to take advantage of the VS11 capabilities. Additional changes were made to simplify the user interface and some new commands were introduced. Also, array dimensions were increased to allow the display of models with more elements.

HIDDEN.FOR includes the subroutines to provide the hidden line or surface removal functions. This module was only slightly modified. The changes that were made involved increasing array sizes so that models with more elements could be processed.

DEVICE.TK4 is the module that provides the Device Driver Routines. Since the drivers for DEVICE.TK4 were written for the Tektronix 4027 color terminal, the driver routines for the VS11 had to be created. The VS11 has a completely different functional description and addressing scheme than the 4027. Thus, the 4027 driver subroutines could only be used as a general guide as to what information was being passed to them and the manner in which they should respond.

Hidden Line - Surface Removal

MOVIE.BYU offers the computer-aided design user two important features, the hidden line algorithms and shading algorithms. These features are not normally available on traditional computer-aided design systems. Although newer solid modeling design systems are beginning to make use of them (1,11).

The hidden line or hidden surface algorithms provide a means for removal of hidden parts from images of solid objects. Figure 1a shows a cube and all the lines that define it. Figure 1b shows the same object with the hidden lines removed; only a part of the external surface is visible.

MOVIE.BYU uses an algorithm developed by Watkins (14) for its hidden line capability. The algorithm operates in image space, i.e. it performs no more calculations than are required for accuracy relative to the resolution of the display. Thus, the algorithm calculates the intensity for each of the 512 x 480 points on the screen. The Watkins algorithm is classified as a Scan Line Algorithm. It solves the hidden line problem one scan line at a time, starting at the top and proceeding down the 480 scan lines to the bottom.

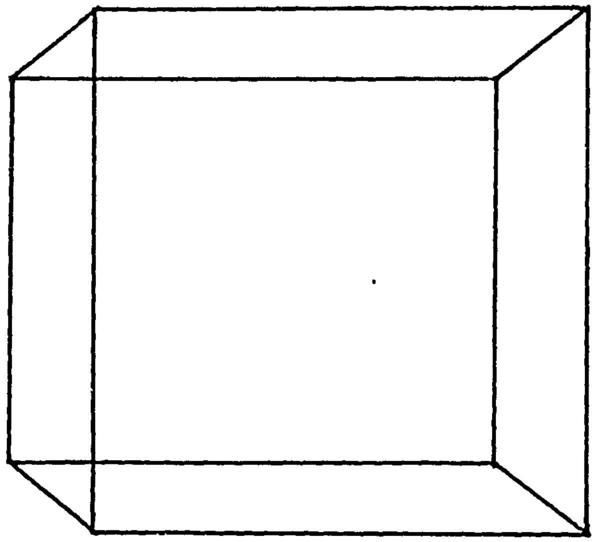


Figure 1a

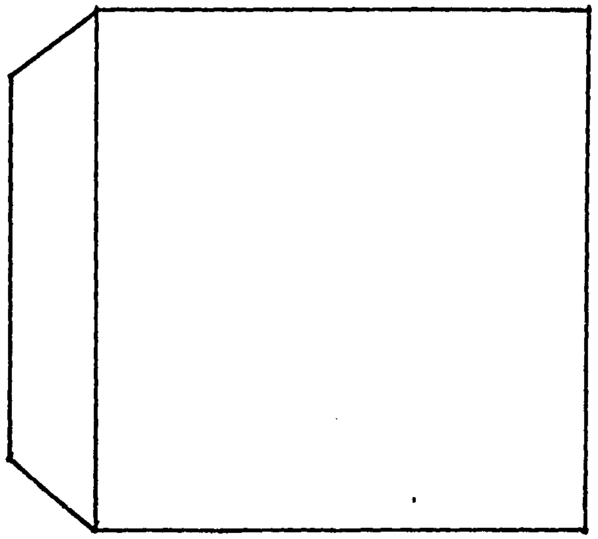


Figure 1b

Figure 1

Hidden Line Removal

The algorithm operates by first doing a y sort of the edges of the polygons, i.e. by sorting the line edges of all the polygons from maximum y to minimum y values. As the scan proceeds from maximum y to minimum y down the screen, the y sorted structure is examined to find any new polygon edges that are on the scan line. These are added to those already there and the edges that terminate on the scan line are deleted. This is an example of scan line coherence. The edges that intersect one scan line are likely to intersect the next. Thus a list of the "working" edges is maintained and the algorithm makes incremental additions and subtractions to the list.

The algorithm then examines this list of "working" edges to determine which faces of the polygons are visible, and the location of the corresponding edges on the scan line. To accomplish this, the edges of the polygons that fall on the scan line are sorted in the x direction and sample segments are created. In Figure 2, the segments AC, CB, and BD are normally created from polygon A and B. In contradistinction to traditional scan-line algorithms, the Watkins algorithm only creates segments AC and CD. It tests to see which line segment is in front of the other by doing a binary subdivision search in the z direction. If the closest edge in the x direction of one segment is deeper in z than the furtherest edge of another segment, then clearly the first segment is behind the second and no further calculations are necessary. In cases where this is not true, line CD is divided at its midpoint (Figure 3). Its left

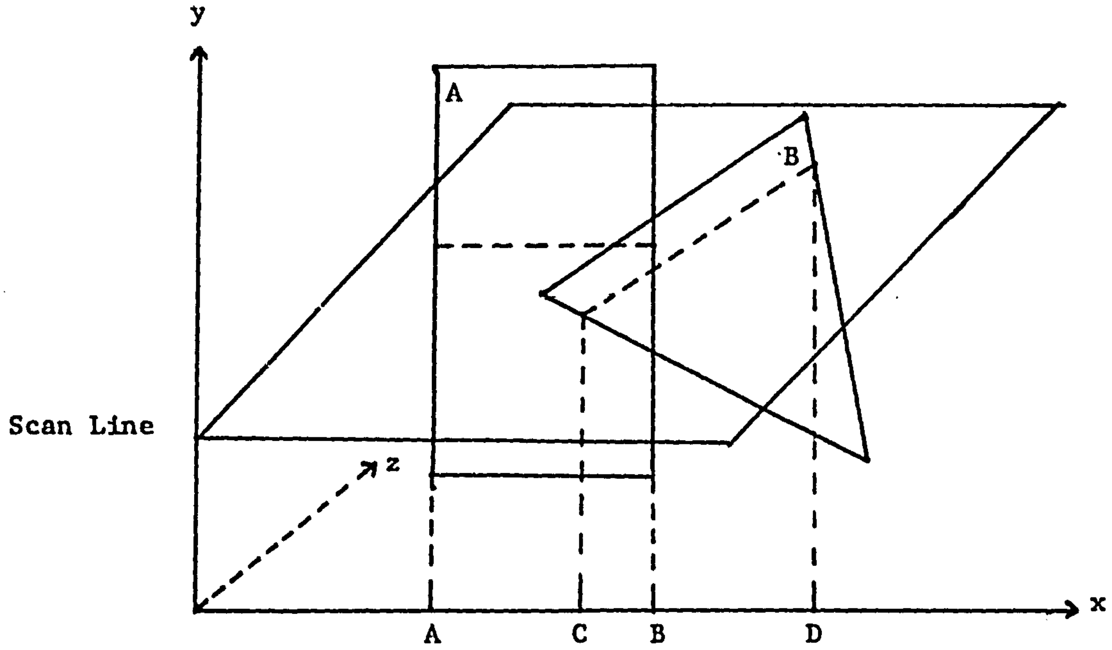


Figure 2

Scan Line Intersecting Polygons A and B

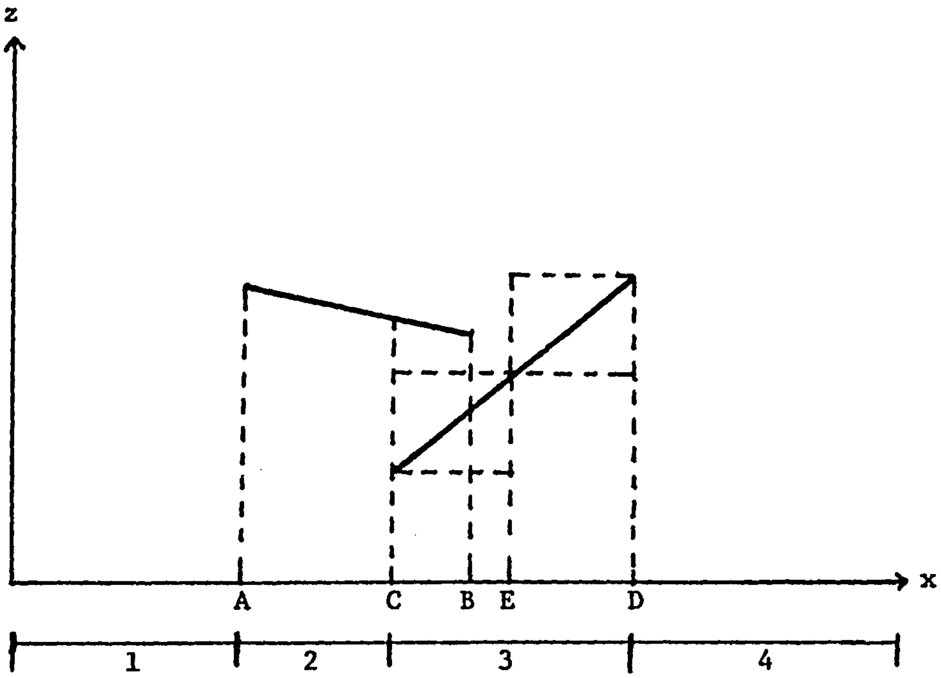


Figure 3

Depth Representation of Segments AB and CD half can be seen to obscure segment CB since point E is closer in the z direction than any point on AB and the depth at C and B need not be computed.

For a general discussion of other hidden line algorithms and an excellent book on computer graphics in general, refer to the book Principles of Interactive Computer Graphics (12). A detailed study of hidden line algorithms can be found in an article entitled "A Characterization of Ten Hidden-Surface Algorithms" (13). A new hidden line algorithm just released by Hedgley (9) which was developed at NASA, claims to be the most robust approach and complete general solution to the hidden line problem.

Shading

The realism of an image generated by computers depends not on just the hidden lines and surfaces being removed but on shading effects as well. Once the hidden surfaces are removed, the colors and their intensities are computed for the remaining surfaces and displayed. In Figure 3, region 1 and 4 would be the background color. Region 2 would be shaded according to the parameters for polygon A. Region 3 would be shaded according to the parameters for polygon B.

The shading effect for MOVIE.BYU is adjusted by the following parameters:

1. Regular light intensity - represents the color of the object and its general brightness.
2. Highlights - provides bright areas on the model caused by portions of the reflected light being indirect or nearly direct in line with the eye.

3. Single light source - can be set at any arbitrary distant position.

4. Diffused value - if a lighted side is such that it can not be seen (i.e., it's shade would match the background), one can increase its intensity by setting the amount of diffused light emitted by such a part.

MOVIE.BYU can generate three distinct types of shading. Figure 4 shows a curved surface approximated by 4 quadrilateral elements. The single head vectors at the corners are the normals to the surface at these corners. Flat shading calculates the corner intensity values using these normals. This produces the effect of distinct polygons, but does allow the variation of color over the polygon.

The single headed vectors at the center of each element are the average of the corner normal vectors. Uniform shading uses these vectors and thus color and intensity would be uniform over the element.

Smooth shading uses the double headed vectors at the nodes which represent the average of the element normals at the node. This allows variation of light intensity and color over the element, but guarantees continuity at the element boundaries.

MOVIE.BYU uses a shading technique developed by Goûraud (8). This algorithm assumes that light intensity varies linearly between any two points on the element boundaries. The flat and smooth shading feature of MOVIE.BYU uses this technique.

Advanced shading techniques such as transparency, shadow casting, surface texture and surface features are available in other

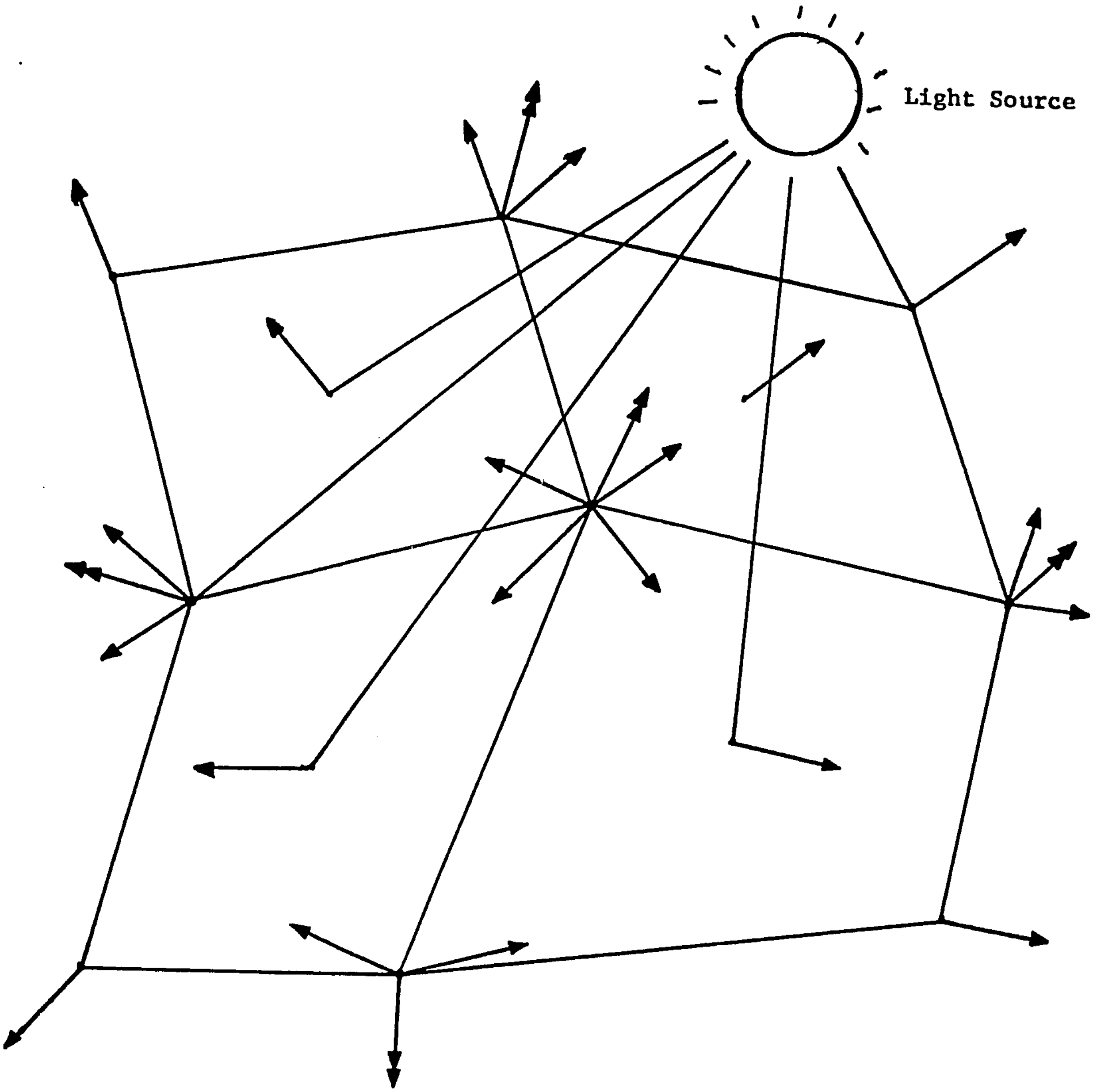


Figure 4

Normal Vectors on a Curved Surface

programs. Extremely realistic images that are difficult to distinguish from photographs can be generated with these techniques. An overview of shading techniques is discussed in the book Principles of Interactive Computer Graphics (12).

CHAPTER IV

DEVICE DRIVER DESCRIPTION

In order to write the device drivers for the VS11 color terminal two problem areas had to be mastered. First, an understanding of the VS11 and its command structure was required. This was accomplished by reading the programmer section of the VS11 users manual (6). This manual gives an opaque and highly technical description of the command structure. Fortunately, a VS11 demo program (SAMPLEF.FOR) supplied by Digital Equipment Corporation demonstrated how some of the commands operated. Between the two sources and much experimentation, an adequate understanding of the VS11 command structure was acquired.

The second problem area and an even more obscure one was the question of which graphic functions were needed and how to interface them with MOVIE.BYU. The two modules which required modification were COMMAND.FOR and DEVICE.TK4. COMMAND.FOR is a very large problem of approximately 7200 lines and DEVICE.TK4 is approximately 2700 lines of comments and Fortran code. DEVICE.TK4 had to contain the main initialization and device driver routines for the VS11 and was renamed DEVICEVS.FOR. Device driver subroutine calls were added to COMMAND.FOR. The latter also required some restructuring of its subroutines and was renamed COMMANDLU.FOR. Since these programs are so large, they are not included in this paper but are available for inspection either in the office of Professor Samuel L. Gulden or at the Mechanical Engineering Computer-Aided Design Laboratory at Lehigh University.

The initialization of the VS11 can be found in the subroutine VS_INIT in DEVICEVS.FOR. First, the required common blocks and declarations of variable types are declared. A check is then made to insure that the users terminal is a VS11 terminal and that it has been assigned to the VS11 system channel. If not, an error is displayed and the program is terminated. To make this assignment, the VAX/VMS command ASSIGN VS_∅ VS11 is issued before the program is run. The _ represents the letter designation of the specific users terminal (A, B, C, D, E, or F).

The system variables are then defined. The variable D1 sets the location of the starting address of the main VS11 segment which is defined in the MACRO program VSA32768. This program reserves a 32K area for the display file and page aligns it. D2 is the starting address of the status block in the VAX. D3 is the starting address of the AUXILIARY segment which is defined in the MACRO program ASCII (the commands for the ASCII character set). AUXLNG sets the size of this AUXILIARY segment. IO_START, IO_READSTATUS, IO_WAITSWITCH and IO_RESUME are used by the VS11 driver to perform I/O operations.

The opcode mnemonics for the bit patterns, in octal format, that are necessary for the display file commands are defined in 63 assignment statements.

The possible color combinations are defined next. The bit patterns for all 16 possible colors are defined in the array OPCDS. The 16 colors are displayed in groups starting with black and continuing with shades of green, blue, red, and purple, and ending with white. The array CCDS contains the 4 bit representation of these colors, they are arranged in the four color groups (green, blue, red and purple) with black and white appended to each group. Five shades of green and purple and six shades of blue and red are generated. These are the "pure" colors that can be displayed by the VS11. These colors are then blended together to form intermediate shades.

After all the variables are defined, a call is made to VS_CLEAR. This subroutine clears the VS11, both the VT100 and the graphic portion, as well as the image memories and monitor. The following action is performed:

```
DSLING = 1
CALL DSSET (LSTC.OR.CHAN3.OR.IM_RD.OR.IM_WRT)
CALL DSSET (LSTC.OR.CHAN2.OR.IM_RD.OR.IM_WRT)
CALL DSSET (LSTC.OR.CHAN1.OR.IM_RD.OR.IM_WRT)
CALL DSSET (LSTC.OR.IM_RD.OR.IM_WRT)
CALL DSSET (UNBLINK)
CALL DSSET (LAS.OR.CLEAR)
CALL VS_IO (IO_START,D1, ZERO)
```

The display file pointer, (DSLING), is set equal to 1. Each call to DSSET loads the bit pattern sent to it into the display file at location DSLING. DSLING is then incremented. The instruction LSTC sets the status register C to the channel selected, i.e., 0 to 3 (four channels are available in this system, but normally only one is used) and enables the channels for reading and writing.

The command UNBLINK turns off the blinking feature. The command LAS.OR.CLEAR loads the status register A with a stop command and clears the image memory.

At this point the display file would have six entries. No action is taken by the display processor until the VS_IO call is given. The VS_IO call tells the display processor where in memory the display list is located, which VS11 channel to use, sets the status register locations and then executes the display list through the LAS instruction. At that point the screen and image memory are cleared.

A debugging feature has been added to the program. If the variable DEBUG is set to .TRUE., a file is printed to list the display file location, bit pattern in octal and most of the corresponding mnemonics. A sample debugging output for the above instructions is presented:

```
Length of Display File = 6

  1     177460      LSTC
  2     177060      LSTC
  3     176460      LSTC
  4     176460      LSTC
  5     176060      UNBLINK
  6     173540      LASCL
```

Line Drawing

The basic type of output that MOVIE.BYU creates is a line. The subroutine LINETO and MOVETO of DEVICE.TK4 are the appropriate draw and move routines for the Tektronix 4027 terminal. The subroutines LVDRW and LVMOV, which draws a line between two end points and which moves the graphic cursor, respectively, mimic the 4027 subroutines. These subroutines are called in COMMANDLU in the routines PLTIN and LABELS. The latter routines had to be modified in order for the calls to be valid.

It should be noted that all modifications to MOVIE.BYU were done in a manner that did not interfere with the operating structure of the other devices that MOVIE.BYU supports. Thus, if a 4027 terminal is available, the program will run properly using it.

The simple box drawing (Figure 1b) demonstrates the line drawing capability, and would generate a display file with 34 entries (Table 1). The first entry would clear the screen. An absolute point command is then given to location x = Ø, y = Ø to establish a reference point. The lines are drawn in terms of "long" vectors. The "long" vector mode is specified and then all data entered into the display file would represent "long" vectors. The "long" vector commands require the next 2 display file entries to represent the change in x direction and change in y direction, respectively. If the 5th octal digit (from right to left) is a 6 or a 4 the line will be drawn. If it is neither 6 or 4, it will not be drawn and the command acts as a move. To draw the 7 lines of Figure 1b requires 7 draw commands plus 7 move commands; a move being issued before each draw. Finally the LAS command is given to indicate the end of the display file.

Length of Display File = 34

1	170140	LSTACL
2	114000	ABSP
3	0	DATA
4	0	DATA
5	113700	LVECCOL
6	1426	DATA
7	1512	DATA
8	60624	DATA
9	10	DATA
10	20142	DATA
11	20116	DATA
12	40142	DATA
13	116	DATA
14	20142	DATA
15	21126	DATA
16	40000	DATA
17	1010	DATA
18	766	DATA
19	21116	DATA
20	40000	DATA
21	1222	DATA
22	20624	DATA
23	21234	DATA
24	60142	DATA
25	116	DATA
26	142	DATA
27	20116	DATA
28	40000	DATA
29	1244	DATA
30	0	DATA
31	21244	DATA
32	40624	DATA
33	12	DATA
34	173400	LAS

TABLE 1

Display File Entries for Figure 1b

Text

MOVIE.BYU has the ability to number the polygons and their nodes. In order to implement this feature on the VS11, it was necessary to be able to put characters in the display file. Thus, subroutine VS_TEXT in DEVICEVS was created and subroutine LABELS had to be modified.

In subroutine LABELS the graphic cursor is first moved to the desired position with a LVMOV call. Since the characters that would be sent for display are numbers up to five digits long, they are kept in an array and passed along with a count of the number of digits to VS_TEXT. The current location of the graphic cursor is then moved to account for the displacement which occurred by printing the number.

Subroutine VS_TEXT sets up the display processor for character entry by issueing character initialization commands to the display file. They set the character base to the AUXILIARY segment where the character structure is stored and activates the character mode. This will assume that until another control opcode is encountered, all further entries in the display file consists of character data. The subroutine then examines each character and zeros out all bits of index greater than seven. Since each display file entry can handle two characters (16 bits), the digits are loaded in pairs into the display file. Moreover, if the number of characters was odd, a space is added after the last digit.

Color

With the VS11 being a color display terminal, MOVIE.BYU should use all of the terminal's capability. A limitation of MOVIE.BYU was that in the line drawing mode, it was not possible to specify the color in which lines would be drawn. For a model composed of one part, this is not a serious limitation. If a model is made up of many parts (MOVIE.BYU is currently dimensioned for 20 parts, but can be increased), it would be good, for clarity purposes, to display them in different colors.

Since the VS11 is only a 16 color display, rather than specify the colors as some form of red, blue and green combination, it is possible to display the 16 colors on the monitor and choose one of them by number. Subroutine VS_COLOR in DEVICEVS displays the possible VS11 colors with one of the numbers 1 through 16 under each color.

The VS_COLOR subroutine places the 16 possible colors across the top of the monitor. This is done by creating 16 horizontal histograms, 50 x 50 pixels in size, and filling them with each of the 16 colors. By using the long vector commands a white border is then placed around the histograms for clarity. The sixteen color histograms are number 1 to 16 by using calls to VS_TEXT. The VT100 portion of the display is advanced 5 lines down from the top of the screen so that all prompts appear below the color display.

The user of the program has the option of specifying color for various parameters in the display of the object. They are:

the border placed around the display, the coordinate axis or triad, the background of the display, and the model itself. The display of the border and coordinate triad are optional, but if either is used its color specifications are selected in subroutine SCOPE of COMMANDLU. If the border or coordinate triad are to be displayed, the color selection chart is first displayed and a choice of color is made. The variables BDKILL and IKILL equal zero if the border and coordinate axis are to be displayed and equal to one if not. The appropriate code, representing the color for the display file, is stored in BDCOL and TRDCOL, respectively. Since the border and coordinate axis will be drawn with long vectors after the model is displayed, the color specification for these vectors is the only information required.

Subroutine COLO of COMMANDLU chooses the color for the background and for each part that comprises the model. First, the user has the option of leaving the background black; in that case the variable BGKILL would equal one. If the user chooses a color for the background, BGKILL would equal zero. The color selection chart is again displayed and a choice of color is requested (a value from 1 to 16 is chosen). The appropriate code for the diaplay file is stored in BGCOL.

The above procedure is followed for all the parts that make up the model. Initially, the parts are set to white in the storing array PATCOL. If any part is to be drawn with a different color, the appropriate color display code is stored for that part in PATCOL.

This completes the color requirements for the display of a model in line drawing mode, with or without hidden lines removed. All the required parameters have been defined because the model is drawn as a collection of vectors and their appropriate colors have been specified.

Under the current operation a model composed of more than one part would be displayed in the following sequence. First the background is colored by filling the screen with the chosen color. This is done by using horizontal histograms to the height of the screen. Next the model is drawn one part at a time with each part drawn in the chosen color. Finally the border and coordinate triad are drawn. This sequence occurs so quickly that the completed screen appears to be drawn at once. ' A problem occurs if the hidden line removal feature is used. Since the parts are no longer considered separately, and the visible line segments are drawn in a top to bottom manner with no regard as to which part they belong; the complete model is drawn in the color of the last part specified.

Patterning

To display the model with a shaded colored surface requires a completely different set of color specification descriptions. With a 16 color display only 3 shades of green, 4 shades of red and blue, and 3 shades of mixed color (purplish) are possible. If a model is to be shaded blue, the 4 available shades will cause the model to appear almost flat and it would be extremely difficult to determine what the object was supposed to be. In order to achieve realism, a blending of the available shades into additional shades in a patterning effect is required. Once this patterning scheme was created, the next problem was interfacing it to MOVIE.BYU.

In determining a patterning scheme, three goals were considered:

1. To simulate at least 256 colors. Since the shading information in MOVIE.BYU is presented in the form of 256 possible values for the 3 primary colors an ideal patterning effect should simulate these.

2. To limit the grainy appearance of the image. A pronounced pattern detracts from the sharpness of the picture and gives it a diffused and grainy appearance.

3. To make the patterning algorithm as efficient as possible. The patterning algorithm would have to be executed 512 x 480 = 245,760 times just to display one picture.

For practical reasons of resolution it is necessary to compromise between the first two goals. The more colors required, the greater the patterning and the grainier the appearance.

Table 2 organizes the possible VS11 colors into 4 color groups: green, blue, red and purple. White and black were appended to each group thus adding two more shades to them. This created 22 possible colors. Table 3a shows the first patterning scheme that was tried. As an example, the two colors black and dark green were chosen. An area of the model that would be shaded a certain color would be at least a few pixels wide. If an area were shaded color 1, the first 6 pixels would be: dark green, black, black, dark green, black, black. This pattern would be repeated

Table 2

VS11 Color Groups

Green Shades					
black	dark green	medium green	light green	white	
Blue Shades					
black	dark blue	medium1 blue	medium2 blue	light blue	white
Red Shades					
black	dark red	medium1 red	medium2 red	light red	white
Purple Shades					
black	dark purple	medium purple	light purple	white	

Table 3a

3 Group Patterning Scheme = 58 Colors

Example:

Black	1	2	Dark Green
BBB	GBB	GGB	GGG

Table 3b

4 Group Patterning Scheme = 76 Colors

Example:

Black	1	2	3	Dark Green
BBBB	GBBB	GBGB	GGGB	GGGG

for any area of the model colored in this shade. The added colors 1 and 2 are introduced between each pair of "pure" colors, this yields 13 shades of green and purple, and 16 shades of red and blue. The "grainyness" of the final picture is very slight and if the monitor is viewed from a few feet away the patterning disappears and the various areas appear to be shaded in new colors. This still did not provide enough shades for a realistic picture.

Table 3b shows the pattern for a 4 pixel group. This pattern would generate 3 points of the same color together and is noticeable as a stripe in the pattern. This scheme still provides a reasonable texture when the model is viewed from a distance of a few feet. The additional shades, 17 of green and purple, and 21 of red and blue, give the model an aspect of realism which is an improvement over the scheme of the 3 pixel grouping.

Any additional increase in the patterning would be too objectionable when viewed in a normal manner. Thus, the 4 pixel grouping was decided upon and implemented.

Implementation

The implementation of this patterning technique required the modification of two subroutines and a creation of a third. Subroutine COLO in COMMANDLU was one that had to be modified. The variables that were defined for the color selections in the line drawing mode are not used in the shading mode. In the shaded mode it is necessary that the color parameters be specified as integers between 0 and 255 for each of the primary colors (green, blue and red). Therefore, the chosen colors had to be translated into this format.

A color would be defined as being a certain shade if its value (0-255) was less than or equal to a defined subdivision in this range. Table 4a shows the subdivision points for the green, blue and red colors. The numbers with the * symbol above them represent the "pure", unpatterned VS11 colors. Therefore, the integers defining the "pure" colors would be stored when the color of the model was chosen. Table 4b shows the "pure" color values that would be used for the chosen VS11 colors. They are located in arrays GCD and RBCD.

In the background color selection, if medium green is chosen, the following variables would be set as indicated: IC1 = 134 (green), IC2 = 0 (blue) and IC3 = 0 (red). The variable values are then stored in one integer (IPB = IC1 * 2 ** 16 + IC2 * 2 ** 8 + IC3). In color selection for the parts of the model, the highest value in the chosen color group is taken. This is done so that the shading of the model will vary over the total range of possible shades. Thus, if medium green is chosen for the model; IC1 = 194, IC2 = 0, IC3 = 0. These values are stored in ICC which is defined in the same manner as IPB.

Subroutine SRL in DEVICEVS contains the general shading algorithm. It evaluates the shading information for the visible segments and then calls the appropriate routine for output. In order to implement

Table 4a

Primary Color Subdivision Scheme

Green   GDIV

$\overset{*}{14}$, 29, 44, 59, // $\overset{*}{74}$, 89, 104, 119 // $\overset{*}{134}$, 144, 164, 179, // $\overset{*}{194}$
209, 224, 240 //

Red and Blue   RBDIV

$\overset{*}{12}$, 24, 36, 49 // $\overset{*}{61}$, 73, 85, 98 // $\overset{*}{110}$, 122, 134, 147 // $\overset{*}{159}$, 171
183, 196 // $\overset{*}{208}$, 220, 232, 245 //

Table 4b

Pure Color Designations

Green

0, 74, 134, 194, 240

Red and Blue

0, 61, 110, 159, 208, 245

SRL, an appropriate insertion of a call to VS_SRL was required. The latter subroutine is the subroutine that drives the VS11. Data is output to VS_SRL one complete scan line at a time with the values of the green, blue and red component for each of the 512 pixels being passed in integer arrays.

Subroutine VS_SRL takes the separate green, blue and red values and translates them into a single 4 bit color value, its "pure" color. It also calculates which, if any, patterning effect should be applied.

The following Fortran code accomplishes this:

```
      OCTAL VALUES DEFINED IN INIT FOR CCDS
      CCDS/0,4,10,14,17,0,1,5,11,15,17,0,2,6,12,16,17,0,3,7,13,17/

      PIXLVAL=0
      DO 249 PC=1,512
        DO 210 I=1,16
          IF(LGREEN(PC).LE.GDIV(I))THEN
            PIXLVAL=CCDS((I+3)/4)
            PTRN(PC)=IMOD(I-1,4)
            GOTO 220
          END IF
210     CONTINUE
        PIXLVAL=15
        PTRN(PC)=0
        GOTO 245
220     DO 230 I=1,20
          IF(LBLUE(PC).LE.RBDIV(I))THEN
            PIXLVAL=PIXLVAL.OR.CCDS((I+23)/4)
            PTRN(PC)=IMAXO(PTRN(PC),IMOD(I-1,4))
            IF(PIXLVAL.EQ.15) THEN
            PTRN(PC)=0
            GOTO 245
          END IF
          GOTO 235
          END IF
230     CONTINUE
        PIXLVAL=15
        PTRN(PC)=0
        GOTO 245
```

```
235     DO 240 I=1,20
          IF(LRED(PC).LE.RBDIV(I))THEN
            PIXLVAL=LIXLVAL.OR.CCDS((I+47)/4)
            PTRN(PC)=IMAXO(PTRN(PC),IMOD(I-1,4))
            IF(PIXLVAL.EQ.15) PTRN(PC)=0
            GOTO 245
          END IF
240     CONTINUE
        PIXLVAL=15
        PTRN(PC)=0
245   DO 246 INDX=1,22
        IF(PIXLVAL.EQ.CCDS(INDX)) THEN
        SLDATA(PC)=INDX
        GOTO 249
      ENDIF
246   CONTINUE
249   CONTINUE
```

Array GDIV and RBDIV are the subdivision points from Table 4a. The array CCDS is included for clarity and shows the octal definition of the "pure" colors. The green value (LGREEN(PC)) for each pixel is compared to the GDIV array to find the subdivision point. When found, the appropriate color code is stored in PIXLVAL (PIXLVAL=CCDS ((I + 3)/4)). The pattern code (0, 1, 2, or 3) is then computed as (PTRN(PC)=IMOD(I-1,4)).

The blue PIXLVAL component (LBLUE(PC)) is logically OR'ed to the previous value of PIXLVAL. The variable PTRN(PC) is the maximum patterning effect due to the green and blue components and is then computed as (PTRN(PC)=IMAXO(PTRN(PC),IMOD(I-1,4)).

The red PIXLVAL component (LRED(PC)) is logically OR'ed to the value of PIXLVAL already defined. The maximum PTRN is computed again.

Finally, SLDATA(PC) equals the index value of the CCDS array that would create the defined PIXLVAL. The index rather than its value is used so that the pattern effect can easily be computed.

The next step is to load the display file with the appropriate commands for displaying pixel data and this is followed by the loading of the pixel data itself. The following display file commands are called for each scan line:

```
CALL DSSET (ABSP)
CALL DSSET (Ø)
CALL DSSET (958-SCAN_LN)
CALL DSSET (BMP1)
LOC = (DSLING - 1) * 2
CALL DSSET (LOC + 6)
CALL DSSET (DJMP)
CALL DSSET (LOC + 262)
```

These commands are described as follows: The graphic cursor is set to the top left hand corner of the monitor (Note: 958 = scan line 459) and the Bit Mapped mode is called. The address of the data in Display List is then specified. Finally, a jump command to the next location after the end of the data is issued. All numbers have to be specified at twice their value, thus, as each scan line is processed, the variable scan line is incremented by two. The data for the pixel colors are loaded into a 16 bit word, 4 bits at a time, thus, a 512 pixel display would require 128 16 bit words.

The physical display time requirements and the length of the display file determine how many scan lines are calculated before being displayed. An average image with the VAX dedicated to a single user would require approximately 2 minutes to display. It is psychologically better to have the picture appear in small segments than to stare at a blank monitor for 2 minutes. The display file is currently set to display 60 scan lines at a time. This implementation keeps the display file well within the 32K maximum size.

The following Fortran code to implement the patterning and to pack the display file is presented:

```
      DO 250 K=1,512,4
       COLOR=0
       DO 275 LOOP=1,4
        INX=(K+LOOP-1)
        CB=SLDATA(INX)
        IF (PTRN(INX).EQ.0)THEN
         ITEMP=CCDS(CB)
        ELSE IF (PTRN(INX).EQ.1)THEN
         IF(ROTATE(PTRN(INX),CB).EQ.0)THEN
          ITEMP=CCDS(CB+1)
         ELSE
          ITEMP=CCDS(CB)
         ENDIF
         ROTATE(PTRN(INX),CB)=IMOD(ROTATE(PTRN(INX),CB)+1,4)
        ELSE IF (PTRN(INX).EQ.2)THEN
         IF(ROTATE(PTRN(INX),CB).EQ.0.OR.
           ROTATE(PTRN(INX),CB).EQ.2)THEN
          ITEMP=CCDS(CB+1)
         ELSE
          ITEMP-CCDS(CB)
         ENDIF
         ROTATE(PTRN(INX),CB)=IMOD(ROTATE(PTRN(INX),CB)+1,4)
        ELSE
         IF(ROTATE(PTRN(INX),CB).EQ.3)THEN
          ITEMP-CCDS(CB)
         ELSE
          ITEMP=CCDS(CB+1)
         ENDIF
         ROTATE(PTRN(INX),CB)=IMOD(ROTATE(PTRN(INX),CB)+1,4)
        ENDIF
        ITEMP=ISHFT(ITEMP,(LOOP-1)*4)
        COLOR=COLOR.OR.ITEMP
   275 CONTINUE
       CALL DSSET(COLOR)
   250 CONTINUE
```

Thus: the 512 pixels of the scan line are loaded into the display file 4 pixels per word. CB equals the appropriate index value of the color code array. If the pixal to be displayed was a "pure" color, then no patterning is required (PTRN(INX)=0), and the pixel code is loaded into ITEMP (ITEMP=CCDS(CB)). ITEMP is then shifted the correct number of bits for its proper placement into the 16 bit word (ITEMP=ISHFT(ITEMP,(LOOP-1)*4), and logically OR'ed with the variable COLOR. When all 16 bits are defined in COLOR, they are stored in the Display File (CALL DSSET (COLOR)).

If patterning is required, the following is done: a two dimension array variable ROTATE(PTRN(INX),CB) is evaluated to choose the proper color to be assigned. ROTATE stores the number of entries into each type of pattern. The first time a specific pattern is accessed, ROTATE=0 and the color representing the first bit is sent to ITEMP. ROTATE is then incremented. The next access to that pattern would generate the color specified by the second bit in that pattern. ROTATE is incremented again. This is repeated for each access to that specific pattern with ROTATE having the possible values 0, 1, 2, or 3.

For example, when PTRN = 2, the bit pattern GBGB from Table 4b is specified. For the first entry into this shade, ROTATE = 0 and ITEMP = CCDS(CB+1). Note that the "pure" color for this group would be black and CCDS(CB) would equal black. Therefore, CCDS(CB+1) would be dark green. ROTATE is incremented to 1. The next entry would generate a black color (CCDS(CB)). Therefore, when ROTATE = 0 or 2, dark green is output, otherwise black is output.

The images created from generating shaded color pictures on a 16-color display is available for viewing in the Mechanical Engineering Computer-Aided Design Laboratory at Lehigh University. For a truly realistic image, however, the 76 simulated colors are not enough and a 256 color display is required.

CHAPTER V

UNIGRAPHICS FINITE ELEMENT MODULE TO MOVIE.BYU INTERFACE

To allow MOVIE.BYU to display models created on the Unigraphics Design System an interface program was written (UGFEMBYU). The output from the finite element module is stored in a files-11 formatted file on the VAX 11/780. The interface program reads this file, converts the data to the proper form for MOVIE.BYU, names the resultant file, and stores it on the VAX 11/780. An understanding of the structure of MOVIE.BYU's input specifications and data structures, as well as the Unigraphics element description and data structure,was demanded in order to convert the Unigraphics format to the MOVIE.BYU format.

MOVIE.BYU Geometric Input Requirements

MOVIE.BYU will accept the geometry of an object defined in terms of N-sided polygons. Figure 5 shows a 3-dimensional surface in space. It contains 4 polygons numbered within a circle. The nodal points that comprise these polygons are numbered in a counter-clockwise order around it. For polygon 1 the node numbers would be 1, 2, 5 and 4. The data structure of this surface would comprise a list of x, y and z coordinates defining each node (coordinate array) and a list of edges (connectivity array) that make up the

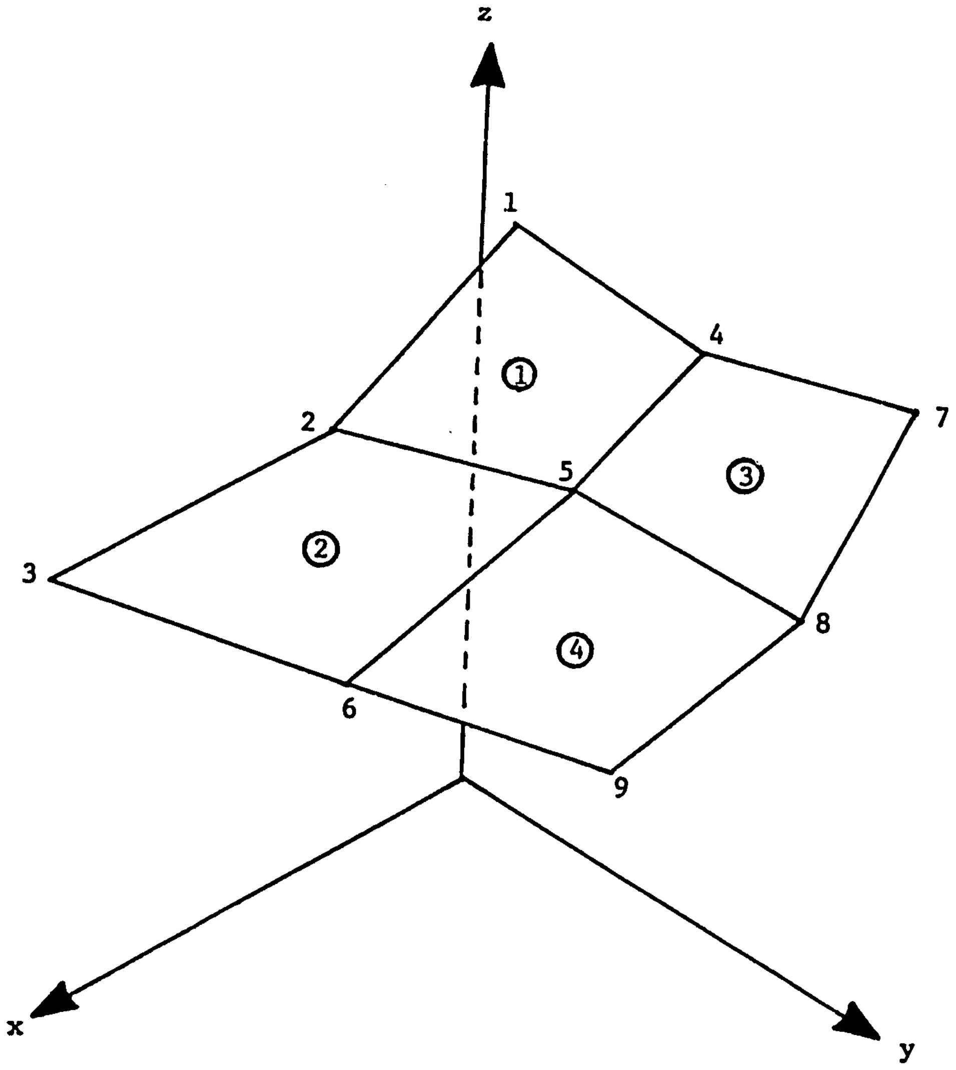


Figure 5

Three-Dimensional Surface in Space

Table 5

Coordinate and Connectivity Array Definition

Coordinate Array

Node Number	x	y	z
1	1.0	2.0	30.0
2	8.0	4.0	20.0
3	20.0	3.0	25.0
4	5.0	11.0	25.0
5	10.0	15.0	28.0
6	18.0	12.0	22.0
7	3.0	35.0	27.0
8	12.0	40.0	20.0
9	22.0	33.0	22.0

Connectivity Array

Element Number	Node Number
1	1
	2
	5
	−4
2	2
	3
	6
	−5
3	4
	5
	8
	−7
4	5
	6
	9
	−8

polygon. Table 5 shows how the coordinate array and connectivity array would be defined for Figure 5. Note that the last entry in the connectivity array for each element is a negative number. This signifies that all nodes on the perimeter of the element have been defined.

Certain control variables also had to be defined. These are: the number of parts comprising the model (NP = 1), the number of nodal coordinates (NJ = 9), the number of elements (NPT = 4), and the number of entries in the coordinate array (NCON = 4*4 = 16). Finally, the parts array had to be defined since a model can be composed of more than one part. A lower element number and an upper element number can define the continuous element connectivity that makes up the part. In the above example there is one part ranging from elements 1 through 4.

Specifically, the geometry of the model is read into MOVIE.BYU with the following Fortran statements:

```
READ (IUNIT, 120) NP,NJ,NPT,NCON,NTEST
READ (IUNIT, 120)((NPL(I,J),I+1,2),J=1,NP)
READ (IUNIT, 130)((X(I,J),I=1,3),J=1,NJ)
READ (IUNIT, 120)(IP(I),I=1,NCON)
120 FORMAT (16I5)
130 FORMAT (6E12.5)
```

The variables are defined as follows:

NP = the number of parts
NJ = the number of nodes
NDT = the number of elements or polygons
NCON = the number of entries in the connectivity array
NTEST = a format test variable (must equal 0)
NPL = the parts array
X = the coordinates of the nodes
IP = the connectivity of the elements

Therefore, the interface program UGFEMBYU writes the data out in this format.

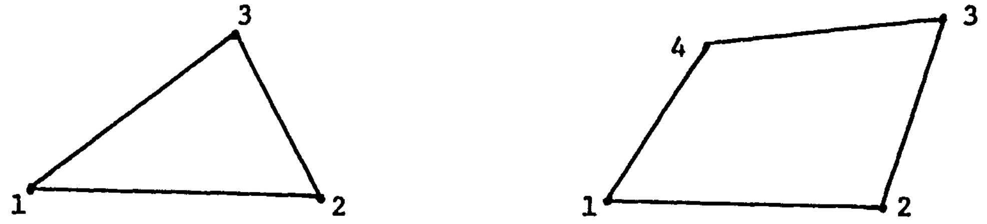
Unigraphics Output Specifications

There are six basic types of elements that are used in creating a finite element mesh on an object. The 2-dimensional elements are the triangle and the quadrilateral (Figure 6). These are 3- and 4-noded structures that are used if only a surface definition of a model is required. This is a sufficient condition for displaying a model with MOVIE.BYU since the viewable object is totally defined in terms of polygons.

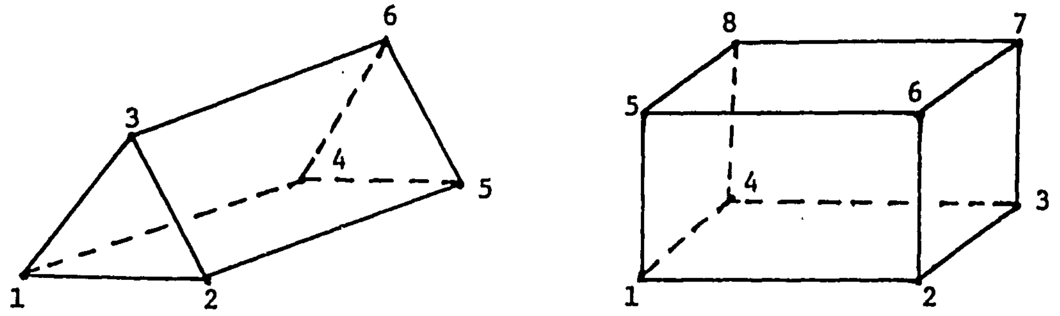
This type of finite element mesh is not as useful, from an engineering point of view, as when the model is defined in terms of solid volume elements. The volume definition is required for many mechanical engineering analysis programs and is the one that is used most often. Therefore, the 3-dimensional elements wedge, box, wedge with interior midpoint nodes (wedge M), and box with interior midpoint nodes (box M) (Figure 6) had tc be handled in the conversion process.

The output of the finite element module is described by the following:

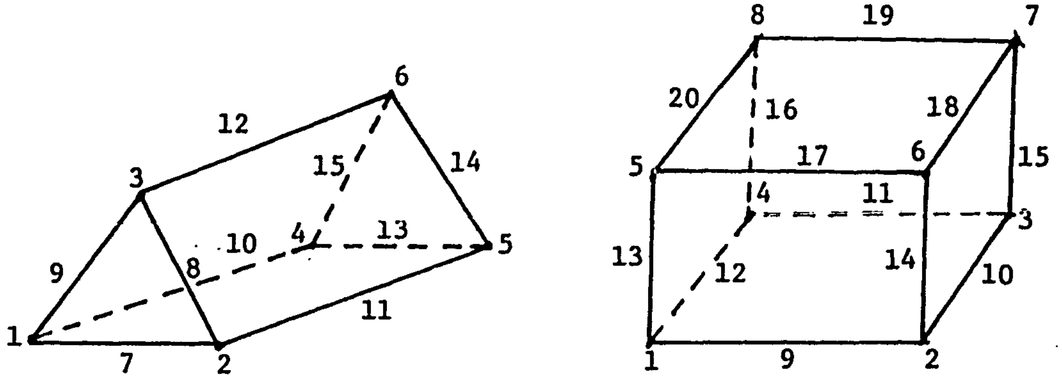
```
"title", application name, part name
"nodes", number of nodes
node label, node property ID, restraint list, x,y,z
   coordinates
.
.
.
```



Two-Dimensional Elements



Three-Dimensional Elements with No Interior Nodes



Three-Dimensional Elements with Interior Midpoint Nodes

Figure 6

Valid Element Types

```
"elements", number of element types, total number of elements
element name, number of elements of this type, number of
   lines composing the element, number of boundary points
element label, element property ID, node label, ...
.
.
.
```

The element name section is repeated for each element type.

Conversion Program

The conversion program has to read the parts in the above format. If the data is composed of 2-dimensional elements, it is directly converted into MOVIE.BYU format. If the data consists of 3-dimensional elements, two additional processes have to occur. In the case of wedge M and box M elements, the midpoint nodes are discarded, and their coordinate definitions are not used. Therefore, these elements are effectively converted into simple wedge and box elements.

Finally, the wedge and box elements are broken into their corresponding quadrilateral surfaces. For Figure 6, the quadrilaterals generated are defined by their nodal points as follows:

Wedge Elements		Box Elements	
1.	1,2,3	1.	1,2,6,5
2.	4,5,6	2.	4,3,7.8
3.	2,5,6,3	3,	1,2,3,4
4.	1,4,6,3	4.	5,6,7,8
5.	1,2,5,4	5.	2,3,7,6
		6.	1,4,8,5

This method creates a large number of polygons, many which are not on the surface and thus cannot be displayed. Currently MOVIE.BYU can handle models composed of 3000 polygons. However, it is very inefficient to be required to apply the Hidden Line Removal algorithm to polygons that are known not to be visible.

Figure 7, shows a typical block structure of a model. Quadrilaterals 1, 2, 3, and 4 are on the interior of the model and should not be passed to MOVIE.BYU since they would add no new information. The edges that define them are calculated from the remaining visible polygons and there is no need to calculate edges of polygons that would contribute no new visible edges. The program that accomplishes this interface and a detailed description of its operation follows.

Implementation

The program UGFEMBYU first initializes the variable limits for the sizes of the arrays. An input file can have as many as 7000 nodes, but the output file for MOVIE.BYU can only havc 3000 nodes and 3000 elements. Since interior nodes on an input file are removed, this ratio is reasonable.

Two temporary arrays are created and initialized to zero. The array VALNODE is indexed by the node label. This is necessary since the nodes are not always presented in a consecutive numerical order, the latter being a requirement of MOVIE.BYU. The array NODENUM is also indexed by the node label and holds a 1 in its first position if the node is used in the element connectivity array. Since the midpoint nodes are not used, their position in NODENUM is left equal to 0 and are thus not passed to MOVIE.BYU. The second position in NODENUM will be the new number that the node will have in MOVIE.BYU. It is

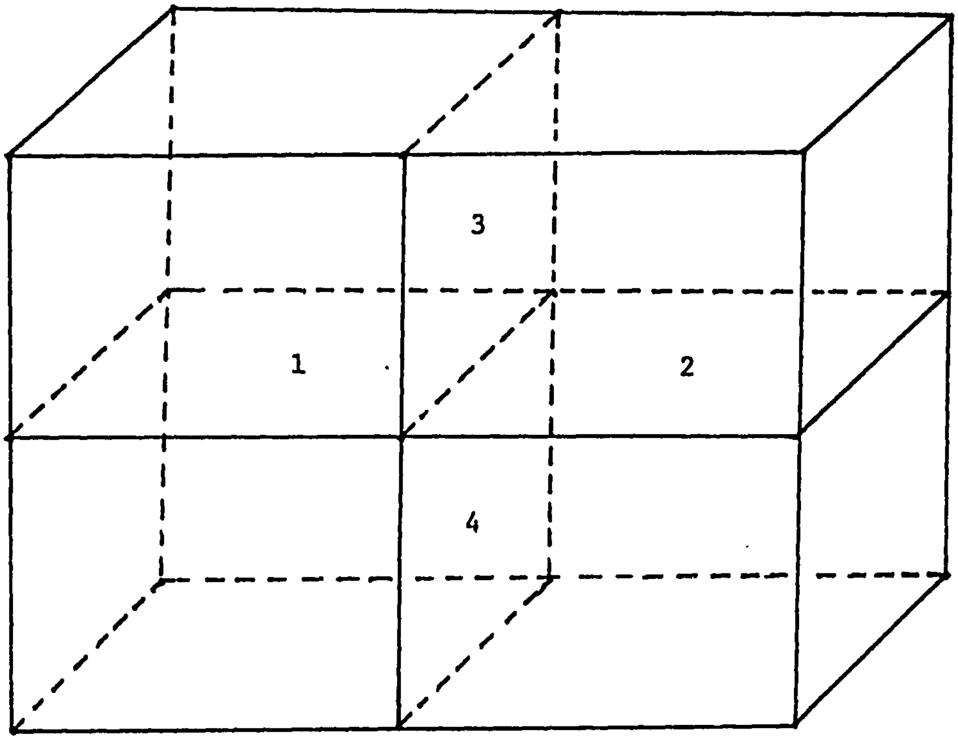


Figure 7

Interior Quadrilaterals from a Box Element Structure created when all interior nodes are selected and the resultant node list is numbered consecutively starting at 1.

ISTOR (J,I) is the array for storing the element connectivity. It is indexed by the element number and lists the node numbers that comprise its connectivity. It is initialized to 0.

After the nodes are read and stored, the number of element types (NETYP) is read and the total number of elements comprising the model (NPT) is read. Next, for each type of element, the number of elements in this group (NE) and the number of nodes that characterize this group of elements (NBPTS) is read.

If the number of nodes that comprise the element equals 15, then it is a wedge M element and only the first six nodes are significant and are read. NBPTS is set equal to 6 and the rest of the record is flagged not to be read (SKIPREC = 1). If NBPTS equals 20, then a box M element is represented and only the first 8 nodes are significant, NBPTS is set equal to 8, and the rest of the record is not read (SKIPREC = 2).

If NBPTS is less than 5, the polygons can be read and stored immediately. IEL equals the polygon number and ISTOR (1,IEL) would be equal to one more than the number of nodes defining the polygon. The node numbers defining the polygon would be stored from position 2 on. NODENUM (1, ISTOR (J,IEL)) would equal 1, since every node read is contributing node to the connectivity and must be flagged in NODENUM.

If NBPTS = 6, the wedge elements are read. Each element is divided into two 3-noded elements and three 4-noded elements. As will be seen later, it is important to store all the 4-noded elements from wedge type, as well as from box type elements together. To accomplish this, a Boolean variable, FLAG8, is used. If FLAG8 is true, then 4-noded elements have been entered and the new 4-noded elements created are stored first followed by the storage of all the 3-noded elements. If FLAG8 is false, the opposite will happen, with the triangle elements being stored first and the quadrilateral elements next. The location for the storage of these elements is computed in IEL3 and IEL4.

The wedge elements and their connectivity are read into a temporary array (EL). The appropriate flags are set in NODENUM for each active node. The triangular elements are then read into storage.

The 3 quadrilaterial elements are stored in ISTOR at IEL4 to IEL4 + 2 in a consistent node numbering scheme. The quadrilaterials are numbered counterclockwise as they are viewed from the exterior of the element. IEL3 and IEL4 are incremented appropriately and the variable ELCREATE specifies that 5 elements were created from the 1 wedge element. If the wedge element was really a wedge M element, the rest of the record is skipped (SKIPREC = 1).

If NBPTS equals 8, then the 8-noded box elements are read. The 8 nodes are read into EL and the appropriate NODENUM flags are set.

A consistent polygonal number scheme is used to store these 6 quadrilaterals. ELCREATE adds 6 more polygons to the count, and if the original element was a box M, then the rest of the record is skipped.

If any other element type is read, then an error message is displayed and the program terminates. Otherwise, the element storing procedure is repeated until all the elements and their connectivity is read.

The active nodes that were stored in VALNODE are now transferred to X, namely the packed storage for the nodal coordinate values. If NODENUM (1,_) = 0, then that node was not used and it is skipped. If it equals 1, the NODECNT is incremented and NODENUM (2,_) equals the new node number (NODECNT). A check is made to see if the maximum (NJMAX) node limit is exceeded. If not, the coordinates for the nodes that were stored in VALNODE are transferred to X.

The next step before storing the data in MOVIE.BYU format is to eliminate any polygons not defining an exterior surface. The following Fortran code shows this algorithm:

```
      VARIFY UNIQUE POLYGONS DELETE IF NOT
          K = CURRENT POLYGON
          L = POLYGON TESTED AGAINST

      IF(ISTOR(1,K).EQ. -1) GOTO 75           !POLYGON DELETED
      L=K
 66   L=L+1
      IF(ISTOR)1,L).EQ.-1) GOTO 66            !POLYGON DELETED
      IF(ISTOR(1,K).NE.ISTOR(1,L)) GOTO 68    !END OF ELEMENT GROUP
      IF(ISTOR(2,K).NE.ISTOR(2,L)) GOTO 66
      IF(ISTOR(3,K).NE.ISTOR(3,L)) GOTO 66
      IF(ISTOR(4,K).NE.ISTOR(4,L)) GOTO 66
      IF(ISTOR(1,K).EQ.4) THEN                   !4 NODED ELEMENTS
        IF(ISTOR(5,K).NE.ISTOR(5,L)) GOTO 66
      ENDIF
      ISTOR(1,L)= -1                             !FLAG TO DELETE POLYGON
      ELDEL=ELDEL+2
      GOTO 75·
   68 DO 70 J=2,ISTOR(1,K)                       !STORE POLYGON
        NEDGE=NEDGE+1
        JP(NEDGE)=NODENUM(2,ISTOR(J,K))
   70 CONTINUE
      JP(NEDGE) = - JP(NEDGE)
   75 CONTINUE
    WRITE(*,450)ELDEL
  450 FORMAT('0','COMMON PLANER SURFACE ELEMENTES DELETED = ',I4)
    NPT= ELCREATE - ELDEL
```

Since the polygons were stored in a consistent numbering scheme, each polygon is compared with every other remaining polygon having the same number of nodes. This is done to see if the compared node numbering schemes are identical. For this reason all 3- and 4-noded polygons have to be stored together. If the node numbering is identical then neither polygon is required. The current polygon being tested is just not stored and its duplicate is flagged with a -1 (ISTOR(1,_) = -1) and skipped over when reference is made to it. ELDEL is increased by 2 to signify that two more elements have been selected.

If no match is found when the end of the list or element group is encountered, the element is then stored in the connectivity array JP. This element is stored with the new nodal index that was computed for it in NODENUM (2,_). Finally, the maximum element count is checked and if it has not been exceeded, the program prompts the user for an output file name of fewer than 8 characters and stores the converted data on it. This terminates the program and the user is ready to run MOVIE.BYU. Appendix 1 shows a sample output with user responses that were generated when running this program.

CHAPTER VI

COMMAND STRUCTURE ADDITIONS AND MODIFICATIONS

Additions

The commands MODE, CLEAR and PRINT were added to the existing commands for additional capability and ease of operation. Previously, to change from Line Drawing mode to Shaded Color mode required the issuing of the SCOPE command. This would also necessitate reentering various color selections and resetting parameters that have not changed. To avoid this, the MODE command was added. It calls subroutine VS_MODE of DEVICEVS which changes the Line Drawing designation of the VS11 (IDVICE = -5) to the shaded color mode (IDVICE = 2) or vice versa, depending on the original status of the display. A message is printed which tells the user the current mode of the terminal.

The ability to clear the screen, both the graphic and VT100 portion, through a user command was desirable. The command CLEAR was added and it calls VS_CLEAR which performs this function.

A final feature that was desired but not implemented is the ability to copy the screen to a color hard copy device. The command PRINT was added that calls VS_PRINT. Since a color hardcopy device is not currently available, just the call to the appropriate subroutine is provided. It shows where the screencopy routine would be located if a color hardcopy device was to be interfaced to

MOVIE.BYU. These commands were added to the list of active commands and are listed when the HELP command is issued.

Default Values

If an object is to be displayed in shaded color mode, many parameters have to be defined. Among them are: choosing the color of the object, the light intensity and highlight effect, the type of shading, the amount of diffused light emmitted, the location of the light source and some obscure parameters such as the setting of the fringe colors. To avoid having to set all these parameters, when the LIGHT command is given, the first response from the user allows the setting of default values. Note that the LIGHT command can be issued directly or, if in the shaded color mode, it is issued automatically when a VIEW or DRAW command is used. If default values are desired then the following parameters are set:

- the regular light exponent for all parts = 1
- the highlight intensity for all parts = 0.2
- the highlight exponent for all parts = 1
- the light source is set at the eye of the observer
- the diffuse light intensity or all parts = 0.2
- the shading mode is set to flat shading
- the standard fringe colors are set
- the standard fringe colors are reflected about a white midpoint
- if smooth shading is requested then all parts are smooth shaded

If these default values are not desired, then all the parameters must be entered by issuing various commands. The choice of using the default values is made each time the LIGHT command is issued.

CHAPTER VII

CONCLUSION

With thc modifications, enhancements and inclusion of the device drivers, MOVIE.BYU can be used on the present hardware. With the inclusion of the interface program UGFEMBYU, a model created on Unigraphics with a finite element mesh applied to it can be displayed with hidden lines removed and as a shaded colored object.

The programs COMMANDLU.FOR, DEVICEVS.FOR, HIDDEN.FOR, VSA32768.MAR, ASCII.MAR, and UGFEMBYU.FOR are available for inspection at the office of Professor Samucl L. Gulden and at the Mechanical Engineering Computer-Aided Design Laboratory at Lehigh University.

Appendix I shows how a complicated part is saved on Unigraphics and transferred by UGFEMBYU into the proper format for MOVIE.BYU. MANDRAL.FEM was chosen because it was created with 20 noded elements and would use every feature of the interface program. Figure 8 shows one view of the object with hidden lines removed.

Appendix II shows how a tie rod (TROD.GEO), which has been transferred to MOVIE.BYU format by UGFEMBYU, is displayed by MOVIE.BYU. A step-by-step description of the system prompts and user responses is included. Figure 9 shows this part as it was originally designed. Figure 10 shows it with the finite element mesh applied to it, wedge and box elements were used. Figure 11 shows the part displayed on MOVIE.BYU with the hidden lines removed.

Appendix III lists all the subroutines of MOVIE.BYU that were modified and those that were created. This is included so that if a future release of MOVIE.BYU is acquired, the areas of modification are identified.

Further documentation for MOVIE.BYU is available at the Computer-Aided Design Laboratory under MOVIE.DOC.

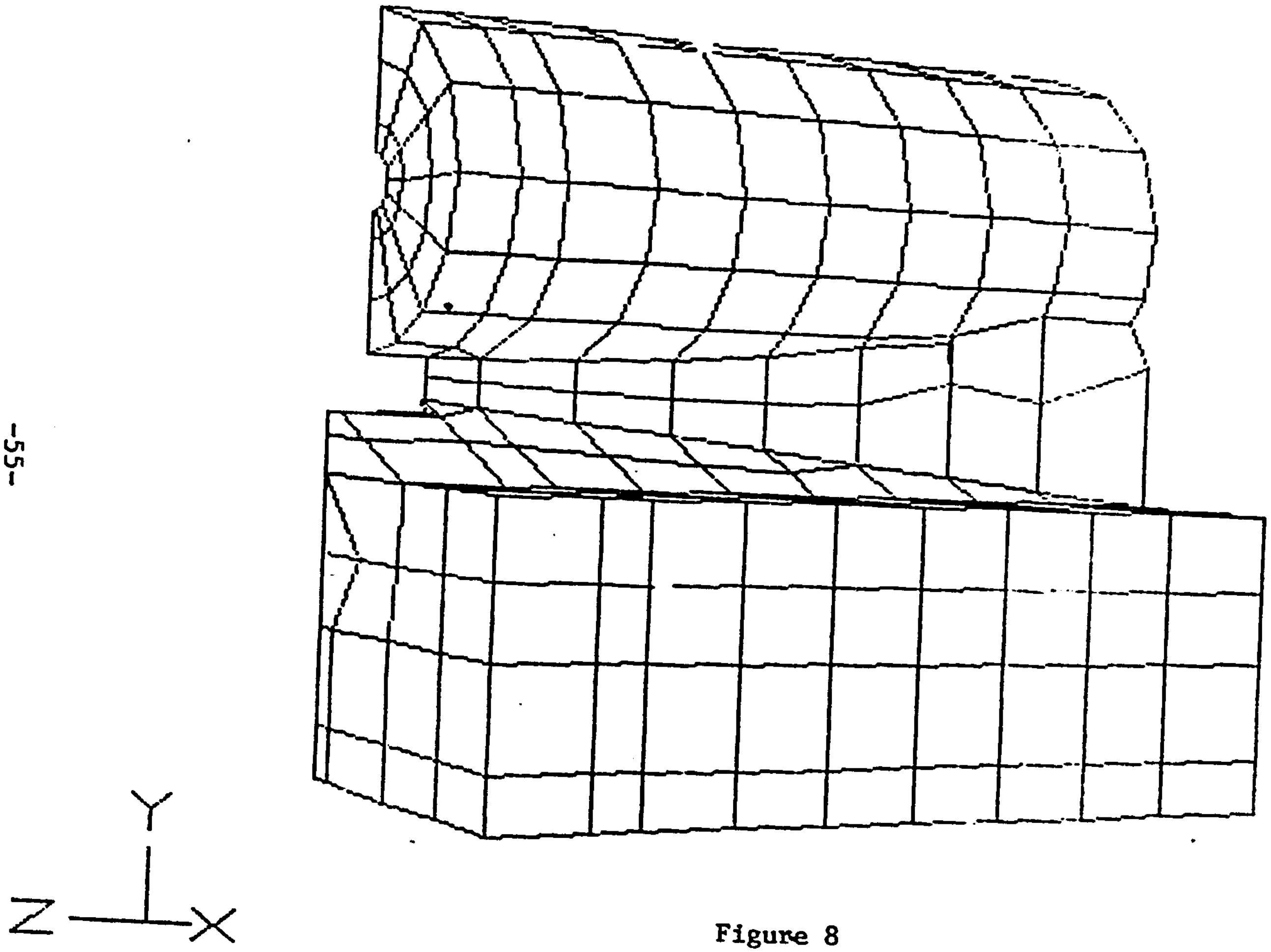


Figure 8

Hidden Line View of MANDRAL

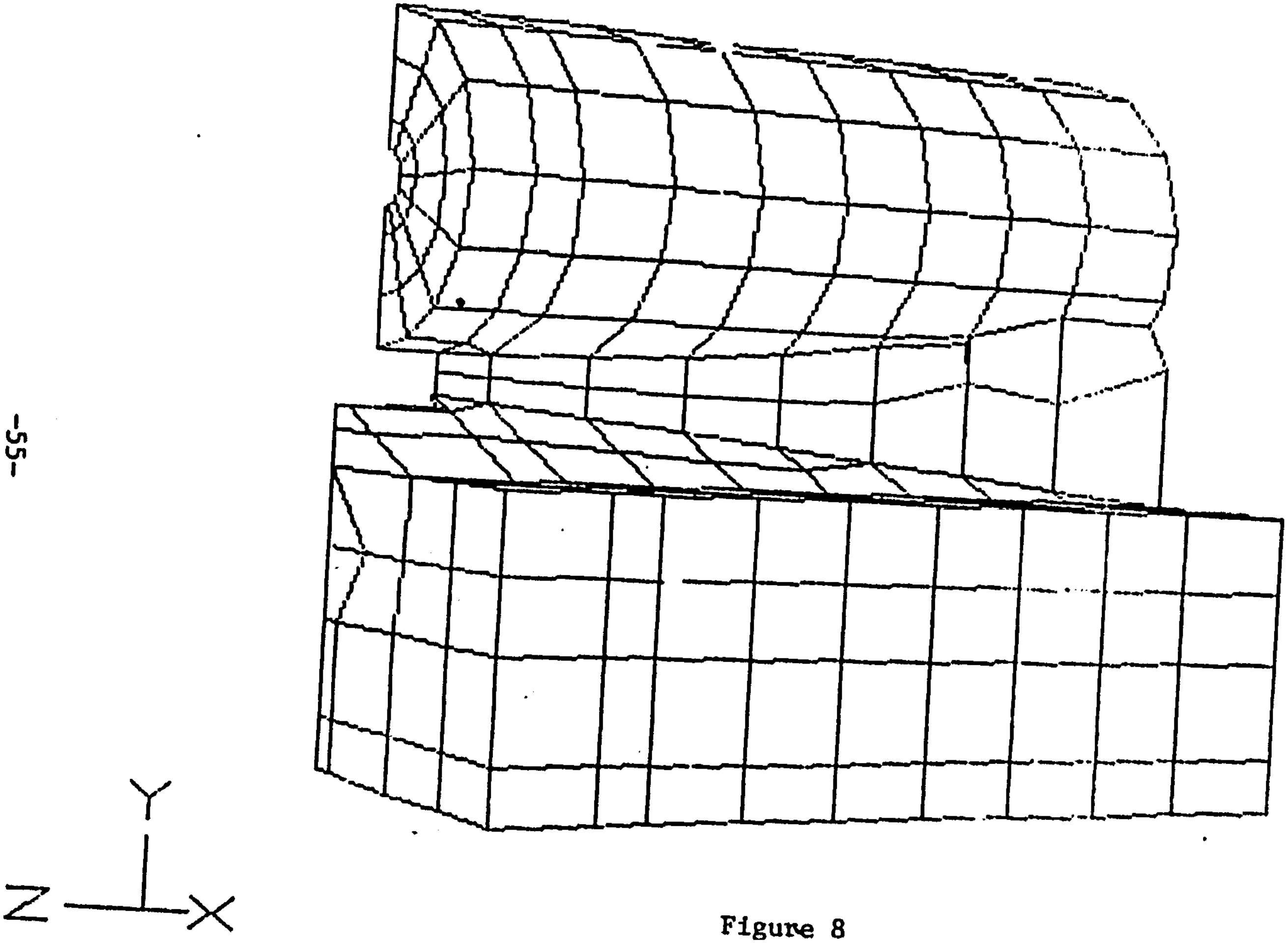


Figure 8

Hidden Line View of MANDRAL

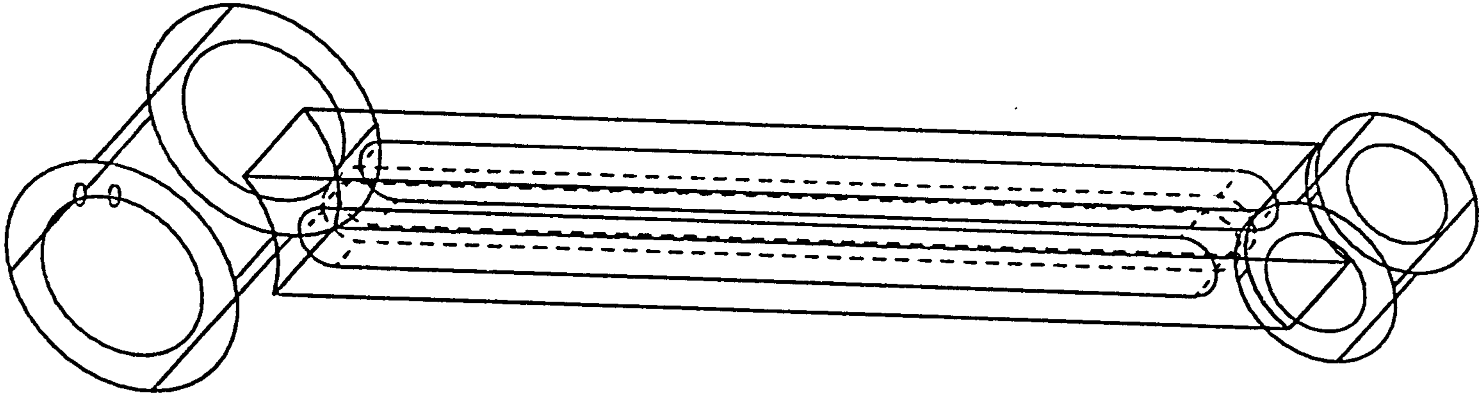


Figure 9-

Tie Rod Original Design

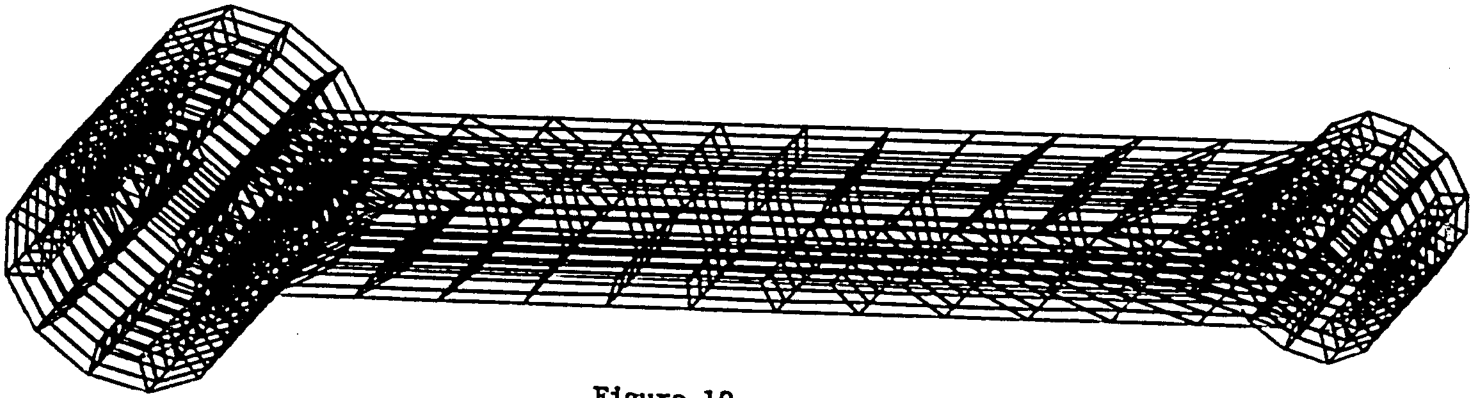


Figure 10

Tie Rod with Finite Element Mesh

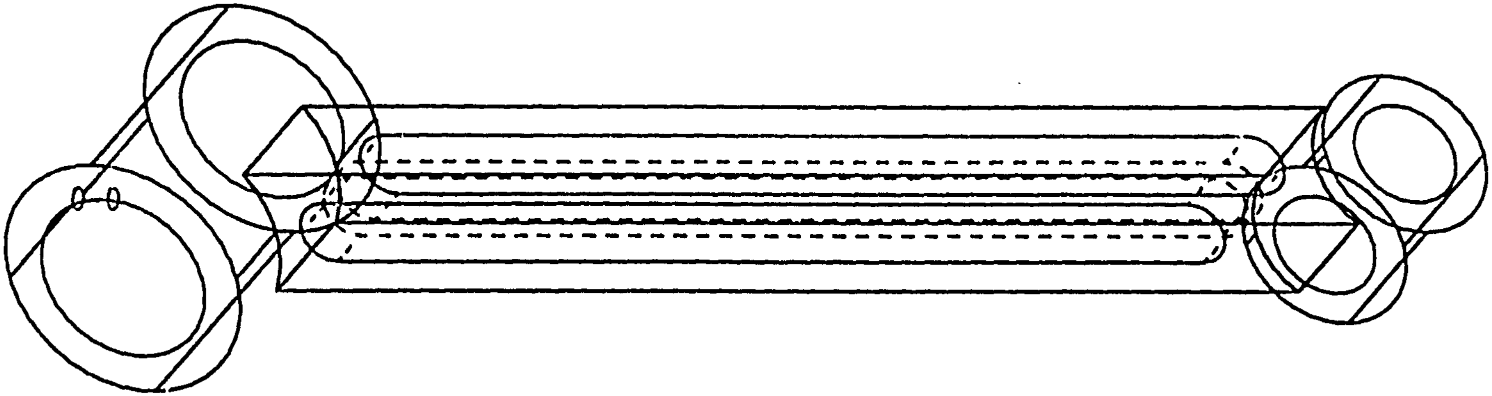


Figure 9-

Tie Rod Original Design

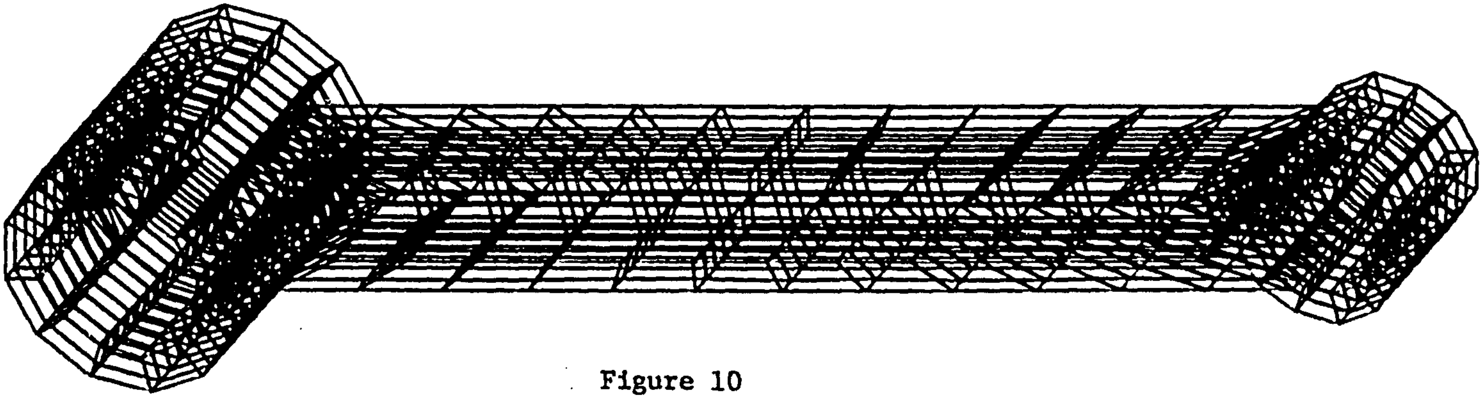


Figure 10

Tie Rod with Finite Element Mesh

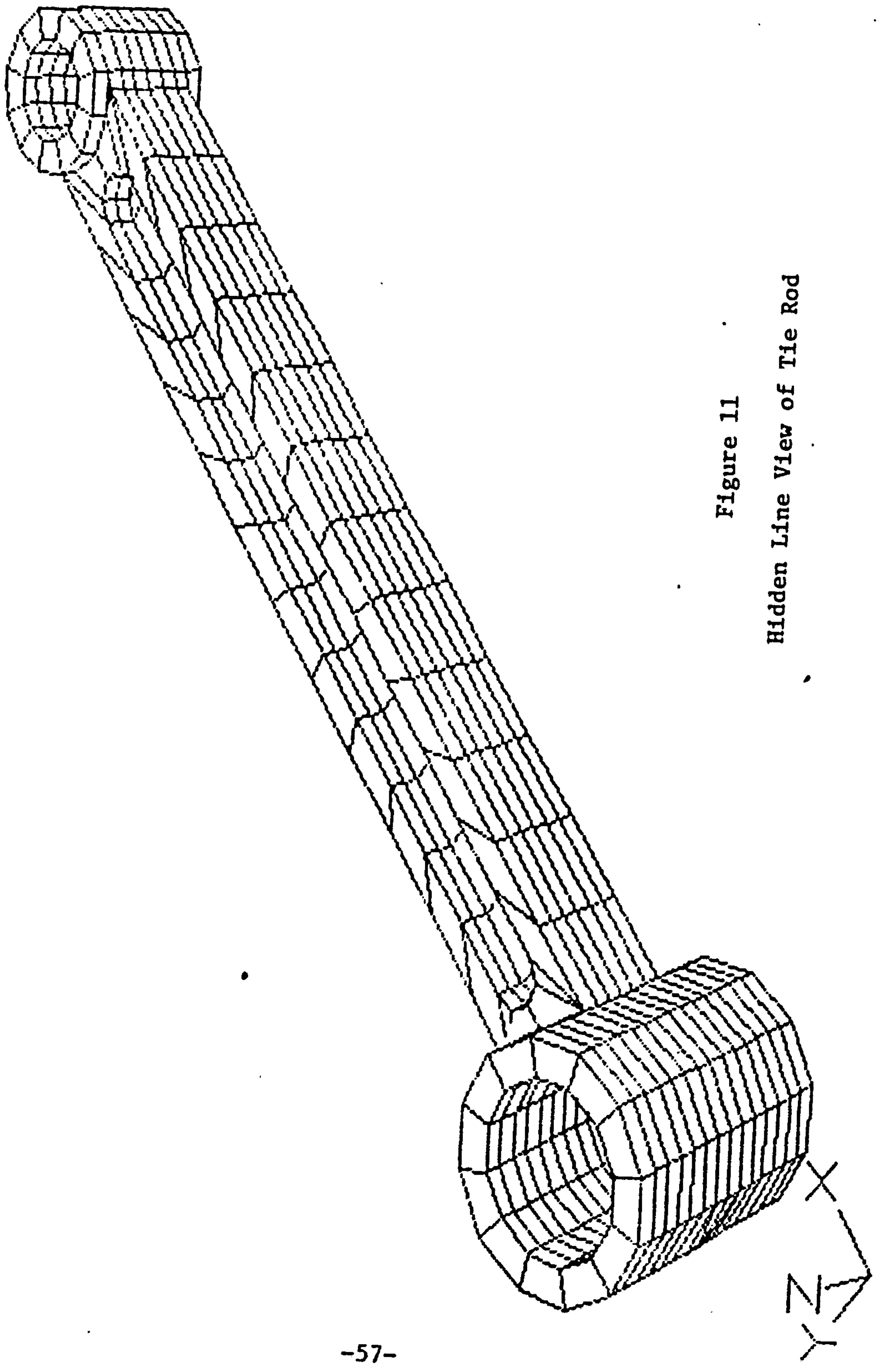


Figure 11

Hidden Line View of Tie Rod

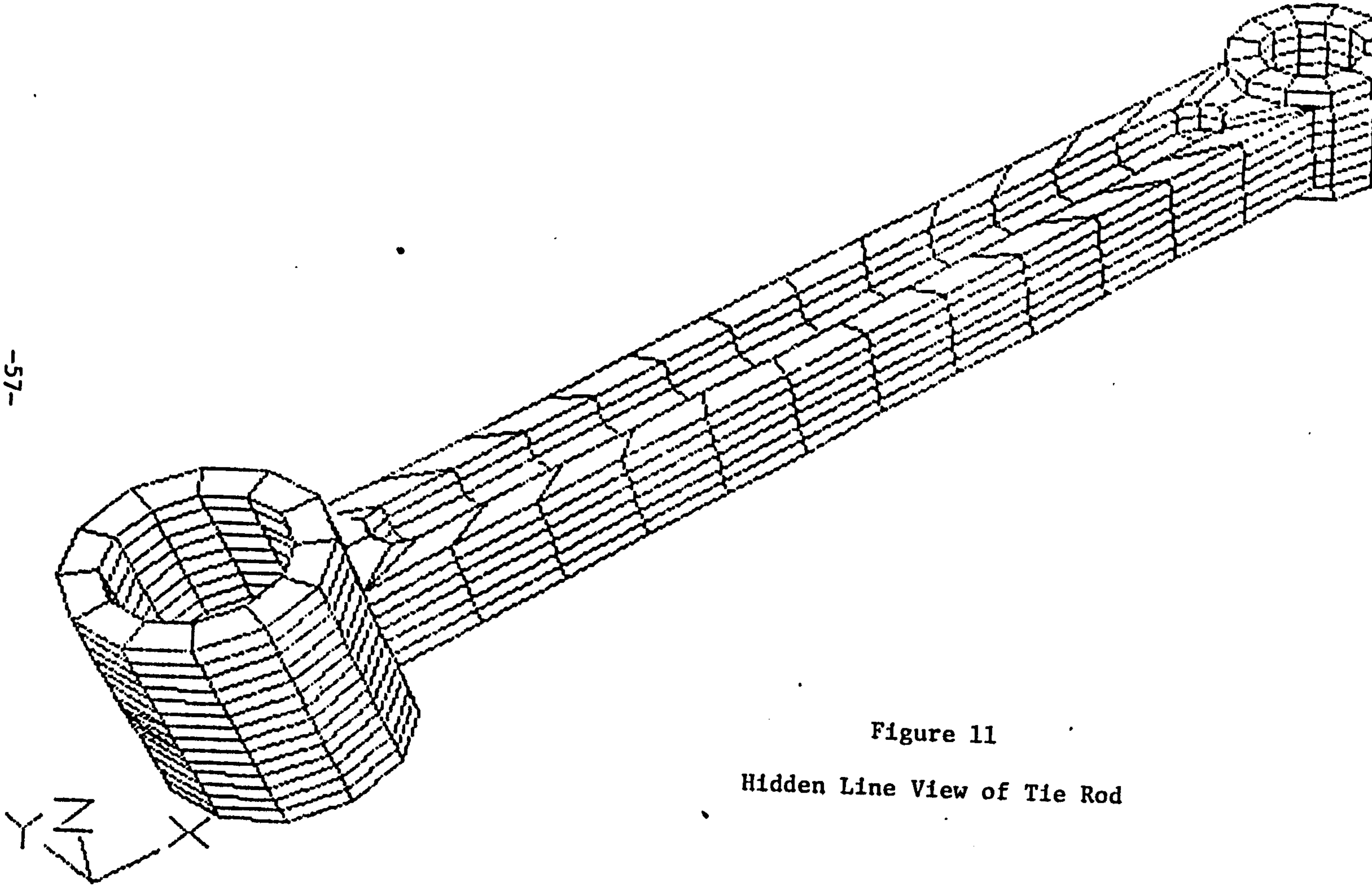


Figure 11

Hidden Line View of Tie Rod

BIBLIOGRAPHY

1. Boyse, John W., Jack E. Gilchrist: "GM SOLID: Interactive Modeling for Design and Analysis of Solids", IEEE Computer Graphics and Applications, Vol. 2:2, March 1982, pp. 27-40.

2. Christiansen, H. N.: June 1981 Edition MOVIE.BYU, Version 4.2, Brigham Young University, Provo, Utah.

3. Christiansen, H. N. and M. B. Stephenson: "MOVIE.BYU Training Manual", Brigham Young University, Provo, Utah, July 1981.

4. Christiansen, H. N.: MINI-MOVIE.BYU, Brigham Young University, Provo Utah.

5. Digital Equipment Corporation: "VAX/VMS Summary Descriptions and Glossary", AA-D022B-TE, Maynard, Massachusetts, March 1980.

6. Digital Equipment Special Systems: "VSV11/VS11 Raster Graphics System", Digital Equipment Corporation, Nashua, New Hampshire, February 1980.

7. Digital Equipment Special Systems: "VS/VSV11 VAX/VMS Version 2.0 S/W Driver's Users Guide", Digital Equipment Corporation, Nashua, New Hampshire, December 1980.

8. Gouraud, H.: "Computer Display of Curved Surfaces", University of Utah, Computer Science Department, UTEC-CSC-71-113, June 1971, NTIS AD-762018.

9. Hedgley, David R., Jr.: "A General Solution to the Hidden Line Problem", Computer Software Management and Information Center; Athens, Georgia, NASA Reference Publication 1085, 1982.

10. McDonnell Douglas Automation Company: "Unigraphics Finite Element Module", ACF 3796C, Cypress, California, May 1980.

11. Myers, Ware: "An Industrial Perspective on Solid Modeling", IEEE Computer Graphics and Applications, Vol. 2:2, March 1982, pp. 86-97.

12. Newman, William M. and Robert F. Sproull: Principles of Interactive Computer Graphics, Second Edition, New York, New York, McGraw Hill, 1979.

13. Sutherland, I. E., R. F. Sproull and R. A. Schumacker: "A Characterization of Ten Hidden-Surface Algorithms", Computer Survey 6(1):1, March 1974.

14. Watkins, C. S.: "A Real-Time Visible Surface Algorithm", Computer Science Department, University of Utah, UTECH-CSC-70-101, June 1970.

APPENDIX I

Instructions to save a part in the Unigraphics finite element module and convert it (using UGEEMBYU) to MOVIE.BYU format.

Note: User responses are underlined.

```
***  UNIGRAPHICS FINITE ELEMENT MODULE  ***

     After finite element mesh has been applied to model
     Format   OPTION 6
     Enter GRIP source file name    MANDRAL
     FILE PART
     LOGOFF AND EXIT
     RUN UGUØ6
     3. Extract a Unigraphics file   3
     5. GRIP Source Library          5
     Give file-11 name (new output file name)   MANDRAL.DAT
     Key in Unigraphics name (GRIP source file name)   MANDRAL
     0. Return to Unigraphics   0
     LOGOFF


***  UGFEMBYU  ***

     RUN UGFEMBYU
     NAME OF DATA FILE FROM UGFEM?  MANDRAL.DAT
     ORIGINAL NUMBER OF NODES = 2428
     2 TYPES OF ELEMENTS WITH A TOTAL OF 310 ELEMENTS
     THERE ARE 22 ELEMENTS WITH 15 NODES
     THERE ARE 288 ELEMENTS WITH 20 NODES
     PLANER SURFACE ELEMENTS CREATED = 1838
     UNUSED INTERIOR NODES DELETED = 1809
     NUMBER OF NODES USED = 619
     COMMON PLANER SURFACE ELEMENTS DELETED = 956
     TOTAL PLANER SURFACE ELEMENTS STORED = 882
     NAME OF GEOM FILE FOR MOVIE.BYU (*******.GEO)?   MANDRAL.GEO
     CONVERSION COMPLETE
```

APPENDIX II

Example of MOVIE.BYU Operation

Note: User responses are underlined

```
RUN DISPLAY
<MOVIE SYSTEM DISPLAY>
<READ GEOM FILE>   TIEROD.GEO
<READ DISP FILE>   CR
<READ FUNC FILE>   CR
<DEVICE>   VS11
<LINE DRAWING OR COLOR>    L
<SUPPRESS PICTURE BORDER?>   N
<ENTER COLOR NUMBER>    8  (Pick color 1 to 16, 8=lt. blue)
<SUPPRESS COORINDATE TRAID?>    N
<ENTER COLOR NUMBER>    7  (7=med. blue)
>>ROTA  (Rotate figure)
<AXIS,ANGLE>    X  -45  Y  30  Z  30 (Rotate X,45°;Y,30°;Z,30°)
>>DIST
<DISTANCE TO ORIGIN (30.27)>    18.0  (Acts as zoom, new distance
                                       to origin = 18.0)
>>DRAW  (Draws figure with white lines, no hidden line removal)
>>VIEW  (Draws figure with white lines, hidden lines removed)
>>MODE  (Changes from line drawing mode to shaded color mode)
<SHADED COLOR MODE>
>>VIEW  (VIEW or DRAW will issue further prompts and then display
         the figure as a shaded color image)
<DEFAULT VALUES?>   Y
<BACKGROUND BLACK?>   Y  (If NO then a color can be chosen for
                          the background)
<PART I1/I2,COLOR NUMBER>
>>>1  1  9  (Chooses a color, 9=red, for each part that comprises
             the model.  Parts 1 to 1 will be colored RED)
>>>CR  (Ends color selection mode)
Figure will now be displayed in shades of red.
```

APPENDIX III

Modified Subroutines in MOVIE.BYU

Type of changes:

1. MAJOR 2. MINOR 3. ARRAY SIZES

COMMAND.FOR

Subroutine	Change	Subroutine	Change
MAIN	2,3	NORMAL	3
ANIMAT	3	OPEN	1
CLEAR	2,3	PAINT	3
COARRO	2,3	PART	3
COLO	1,3	PIVOT	3
CONEL	3	POINTS	3
COORD	3	POLNUM	3
DIFF	2,3	POLYD	3
DRAW	3	POLYV	3
EDGE	3	READ	3
EXPL	3	REST	3
FAST	3	ROTA	3
FRIN	3	SCOP	1
HELP	1	SEPART	3
IMMUNE	3	SHADE	2,3
INIT	2,3	SHRINK	3
INTHID	3	TRAN	3
ISHADE	3	VIEDRA	1,3
LASTC	3	VISIP	3
LIGHT	1,3	VISIT	3
LINE	3		
MULTDC	3		
MULTDD	3		
NODNUM	3		
NORAV	3		
NORMI	3		

HIDDEN.FOR

Subroutine	Change
CLIP	3
CONSHO	3
DRAWIT	3
EDGMAK	3
GETBLK	3
HIDDEN	2,3
INSECT	3
LINSHO	3
LSTSET	3
PACKER	3
POLMAK	3
POLSNP	3
RETBLK	3
UNPACK	3

DEVICE.TK4

Subroutine	Change
BGNFRM	1
ENDFRM	1
LABELS	1
PLTLIN	1
SRL	1

Created Subroutines in MOVIE.BYU

DEVICE.TK4-DEVICEVS.FOR

Subroutine

DSSET
LVDRW
LVMOV
VS_CLEAR
VS_COLOR
VS_INIT
VS_MODE
VS_PRINT
VS_IO
VS_SRL
VS_TEXT

VITA

Jan Carl Silverman was born to Alice and Stanley Silverman in Wilkes-Barre, Pennsylvania on June 29, 1950. He attended Wyoming Valley West High School in 1968 where he was graduated with honors. He then attended Rensselaer Polytechnic Institute in Troy, New York from 1968 to 1972, and was graduated with a degree in Mechanical Engineering. For the next ten years he managed his own business, until he enrolled at Lehigh University, where he received an M.S. degree in Computing Science in 1982.

Mr. Silverman is presently employed at Sanders Associates, Nashua, New Hampshire, developing their implementation of the Core Graphic Standards. He is a member of the IEEE and ACM.