264

CONVERSION OF A GRAPHICS PACKAGE TO SEQUENTIAL PASCAL

by

DANIEL THOMAS SNYDER

B. S., Ohio State University, 1973

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1978

Approved by:

[illegible]
Major Professor

TABLE OF CONTENTS

FIGURES

ACKNOWLEDGEMENTS

I wish to express my gratitude to Dr. William Hankley, my major professor, for his many contributions and the guidance he gave me toward the completion of this project. Also, I owe considerable thanks to Ms. M. C. Neal for the use of her Computek driver, which saved a great deal of time and initially helped get the project off the ground. And finally to my wife and partner, Gretchen, goes a great measure of affection and appreciation for her moral support and understanding.

I. INTRODUCTION

A. PURPOSE OF PAPER

The purpose of this paper is to describe porting of the Computer Science Graphics Package (CSGP) from a large computer environment, IBM/370 (Conversational Monitor System) [And75], to a minicomputer environment, INTERDATA 8/32 (SOLO Operating System) and rewriting the code from FORTRAN to Sequential Pascal (SPASCAL). Both implementations interact with the Computek 300 GT display terminal [Com72].

The target programming language, the Kansas State University (KSU) implementation of SPASCAL ([Han77a], [Nea77]), runs on the INTERDATA 8/32 as a job process under SOLO. SOLO [Nea77], a single user operating system written in Concurrent Pascal (CPASCAL), runs as a task under OS-32/MT. SPASCAL, CPASCAL, and SOLO were ported to KSU from PDP-11/45 implementations at the California Institute of Technology.

The CSGP allows interactive communication between a Computek terminal user and a remote computer which executes programs to construct, transform, and display three-dimensional straight-line pictures. The graphics package software consists of a set of routines which builds and manipulates a data structure image representation, called the Pseudo Display File (PDF), and translates the PDF into code suitable for display at the terminal. Image construction commands permit movement of the cursor, line drawing and erasing, and display of alphanumeric characters. Transformation commands allow movement, rotation, scaling, and reflection of a picture. The software structure is modular in both implementations for greater maintainability and extensibility of the package.

The rationale behind the desire to port the CSGP is concerned with the issues of moving from a large-scale computer to a minicomputer environment and converting the code from FORTRAN to SPASCAL. The change of computing environments was accomplished to move the CSGP away from a time sharing system to a system with a faster response mode. The change of programming languages involved several reasons. The KSU Computer Science Department's research emphasis and computing resources have currently been geared toward the utilization of Pascal. More importantly, however, SPASCAL possesses better features that enhance the maintainability and extensibility of the package. That is, the SPASCAL compiler enforces structure in data types, data structures, and program flow, and it provides more thorough error detection. In addition, the language allows user-defined data types and more powerful data structures, and the code is very readable.

The porting of the CSGP was more than a mere line-by-line translation of FORTRAN to SPASCAL. A PDF construction routine and an image transformation routine deemed unnecessary to the new implementation were deleted. Two routines to access and change the value of the current PDF index and a routine to allow user-defined screen coordinate units were added. FORTRAN CSGP routines providing the interface between the Computek terminal and the IBM/370 were completely deleted, and two new SOLO interface routines were written, one of which involved making minor revisions to a program written by another student for a project [Nea78] also using the Computek. The FORTRAN four-dimensional array containing the PDF, the two-dimensional array containing alphanumeric character string location and length information, and the linear array holding the character strings were converted into one SPASCAL array of records in order to improve readability of the code and clarity of the design. Due to the I/O limitations of SOLO and SPASCAL, a set of routines had to be written to support output of the PDF to the printer. Also, output of the PDF to the user's console (CRT) was added. Detection of errors in the user program was revised; such that, the output of error messages was removed from the points of detection and placed into a separate routine. Finally, a new set of user instructions was written to incorporate the changes due to the revisions made to the software and conversion to the new computing environment. The portions of the CSGP that required a line-by-line conversion of FORTRAN to SPASCAL were most of the routines that enter image building commands into the PDF and that perform transformation of images.

The entire FORTRAN package was not ported. Those FORTRAN modules that were ported along with their SPASCAL equivalents and those FORTRAN modules remaining to be ported are listed in Appendix C. Most of the code not moved is concerned with the clipping of pictures and the perspective viewing of images, i.e., the mapping of X, Y, and Z three-dimensional coordinates of image end points into two-dimensional X and Y coordinates. While perspective viewing has not been provided, the capability of specifying three dimensions (X, Y, and Z coordinates) in building and transforming images has been built into the package. A line-by-line high level conversion of the FORTRAN clipping and perspective viewing subroutines to SPASCAL is all that remains to complete the porting of these features.

The SPASCAL version of the CSGP consists of approximately 880 lines of code compared to the functional equivalent of about 550 lines of FORTRAN and IBM/370 assembler code. The greater number of SPASCAL lines is due in part to indentation of SPASCAL statements, where one statement may be split into as many as four lines to increase code readability. Also, programming with the current KSU implementation of SPASCAL requires more interaction (more code) with the operating system to perform output to the printer and Computek deviced (discussed in Chapter IV). The manhours expended during each phase of the project were as follows: Organization/Design, 130; Coding, 20; Test/Debug, 100; Documentation, 80; and Total, 330.

B. ORGANIZATION

The organization of the material presented in this paper is such that both those individuals who want only to use the CSGP and those who desire an in-depth knowledge of any part or all of the software can find the information of interest. The second chapter of the report contains the necessary instructions for users to gain access to the INTERDATA 8/32 and use the CSGP services. Providing a more detailed explanation of the software, Chapter III discusses the data structures, the routines, and the overall structure of the system. The SPASCAL CSGP code itself is included in Appendices A and B, and Appendix C provides a list of both the FORTRAN and SPASCAL modules. The final chapter gives an analysis of the problems encountered, lessons gained, and some concluding remarks concerning the effort of converting FORTRAN code to SPASCAL and of converting from a large computer to a minicomputer environment.

II. USER INFORMATION

A. GENERAL INFORMATION

This chapter is intended to serve as a user's guide to the Computer Science Graphics Package, written in SPASCAL and implemented on the INTERDATA 8/32 at Kansas State University. A complete working knowledge of SPASCAL or the INTERDATA 8/32 is not required in order to use the CSGP. In those few instances where information concerning these areas is necessary, a simple explanation will be given or an outside publication will be referenced.

The CSGP allows its user to display and manipulate straight-line pictures on the Computek 300 GT display terminal. To perform graphics operations, an SPASCAL program must be written, compiled, and executed on the INTERDATA 8/32. This SPASCAL program consists of the set of CSGP routines plus a main program, which is prepared by the user and contains a list of CSGP commands. CSGP commands are, in reality, calls to various CSGP routines designed to perform graphics functions. Section B of this chapter describes the CSGP commands available, and a sample user program in Section C give examples of their uses.

The Computek 300 GT terminal is a vector/alphanumeric terminal which allows a user to display text and straight lines either locally or under the control of a computer program. The terminal comes equipped with a graphic pen and tablet, but since this initial implementation of the CSGP does not employ their use, they will not be discussed here. The Computek terminal has a display screen measuring 7 inches high and 8 inches wide consisting of a grid of 256X256 discrete points. Text and lines are displayed by illuminating an appropriate set of these points, which are always either "on" or "off." There is no intensity or color control. Characters (64 ASCII character set) are generated from a 5X7 dot matrix, and the terminal screen can hold a maximum of 42 characters per line and a total of 24 lines. The Computek terminal has an electronic image retention memory; that is, once an image is received, it is stored in bit form at the Computek and used locally to refresh the CRT screen. The image memory does not need to be refreshed from the remote computer.

When the Computek terminal is displaying text, drawing lines, or performing any of its functions, it is under a unique mode and one of two possible status states. The following is a list of the Computek modes, status states, and controls applicable to this implementation:

Alphanumeric Mode - This mode allows the terminal to receive and display the upper-case ASCII symbol set. All other modes enter from and return to this mode.

Four-Byte Absolute Mode - This mode is used for specifying absolute vectors, and it is entered from the alphanumeric mode by sending octal 034 code to the terminal. An absolute vector is drawn from the last cursor position to a new position specified by a sequence of four consecutive bytes. The four bytes denote the X and Y coordinates of the end of the vector, whether a line is to be displayed or not, and whether the next instruction is for this mode or the alphanumeric mode.

<u>Erase Status</u> - This command (octal code 016), if received when the terminal is in the alphanumeric mode, places the Computek in erase status. While in erase status, all characters or symbols received will replace the character or symbol at the present cursor position, and all vectors received will be erased from the screen.

<u>Write Status</u> - This command (octal code 017), if received when the terminal is in the alphanumeric mode, returns the Computek to write status (its normal status).

<u>Home/Erase</u> - This command (octal code 014), if received when the terminal is in alphanumeric mode, erases the screen and positions the cursor at home, (0,0).

<u>Line Feed</u> - Causes a line feed upon receipt of octal code 012.

<u>Back Space</u> - Causes a back space upon receipt of octal code 010.

The above description of the Computek 300 GT display terminal is by no means comprehensive. More information can be found in [Com72] and [And75].

At this point a note is in order concerning the specification of numerical values in SPASCAL. When a CSGP command calls for an argument to be given in real or integer form, the number must be given in compliance with SPASCAL format and within SPASCAL limitations. The format for possible integer and real numerical constants can be found in [Han77a] or [Han77b]. The largest possible integer value is 32,767 and the smallest -32,767. The largest possible real value is 10^{38} and the smallest -10^{38}.

B. CSGP COMMANDS

CSGP commands are a set of SPASCAL statements which the user can use in a program to perform graphics image processing. This explanation of CSGP services will be focused on the following four main topics: Image Construction and Display Commands, Image Transformation Commands, PDF Output Commands, and Error Messages. The first three of these topics deal with the primary groups of CSGP commands. The description of each command begins with its name and argument list noted in a rectangular box. A short paragraph below the box describes what the command does and the meaning and proper format of its arguments. Following the description of the CSGP commands, an explanation of CSGP error diagnostics is provided to acquaint the user with the meaning of error messages given by the CSGP software.

<u>Image Construction and Display Commands</u>.

As briefly mentioned in Chapter I, the CSGP allows the user to enter a representation of a picture into a data structure called the Pseudo Display File (PDF). The PDF is a linear array containing a maximum of 600 records, each record containing a display instruction. An illustration of the format of a PDF record is given in Figure III.7 (Section B of Chapter III). All PDF images are represented in a hypothetical three-dimensional coordinate system with real (not integer) coordinate values. For display, only the X and Y values are converted to integers and mapped into the 0 to 255 range by the CSGP software that drives the Computek terminal. The variable INDEX is used as a pointer to the next empty (not containing a valid instruction) location in the PDF, and it is incremented by one each time a new command is entered.

After a picture has been constructed and perhaps manipulated by the transformation commands (described in the next group of commands), the portion of the PDF containing the image must be translated into a Computek compatible form for the picture to be displayed at the terminal. This translation is performed in two steps. First, the COMPIL procedure is called by the user, and the procedure begins scanning a specified portion of the PDF. During this scanning process, the PDF instructions are converted into pages of integer values, one page at a time, and sent to the Computek driver program. Within this program, each integer page is again scanned and converted into pages of ASCII characters, again one page at a time, and sent to the Computek terminal device, where the ASCII characters control its operation. For the reader who is interested, more information concerning these CSGP functions can be found in Section C of Chapter III.

START;	

The START procedure is called to start a new PDF by setting INDEX to one. Therefore, any old PDF entries will be overwritten and lost. START is normally the first command given in a CSGP user program.

SET_INDEX	(I);

SET_INDEX changes the current value of INDEX to the value of I. I must be given in integer form.

VAL_INDEX;	

The function VAL_INDEX returns the value of INDEX when called. INDEX points to the next unused location in the

PDF. VAL_INDEX can be used as an integer argument of another command, either alone or as part of an integer expression.

SET_SCALE	(X, Y);

SET_SCALE allows coordinates to be specified in units other than points along the 256X256 screen grid. X and Y are real values that signify the number of screen points per user-defined unit along the respective X and Y axes. For example, the Computek display screen is 7 inches high and 8 inches wide. If inch units are desired, the X argument would be given as 32.0 (256/8 = 32.0 X grid points per inch) and the Y argument would be 36.57 (256/7 = 36.57 Y grid points per inch). If SET_SCALE is not called, the START command initializes both X and Y scale factors to 1.0.

MOVE	(X, Y, Z);

The MOVE procedure is called to insert an instruction into the PDF to move the cursor from its last (current) position to the position indicated by the real-valued arguments X, Y, and Z. MOVE does not draw a line.

VECTOR	(X, Y, Z);

The VECTOR command inserts an instruction into the PDF to draw a straight line from the cursor's last position to the position indicated by the real-valued arguments X, Y, and Z.

CLEAR;	

The CLEAR procedure is called to insert an instruction into the PDF to clear (erase) the entire screen.

EMODE;	

The EMODE command inserts an instruction into the PDF to switch the display to the "erase" status.

WMODE;	

The WMODE command inserts an instruction into the PDF to switch the display to the "write" status.

DEBUG_CMPTK_DRVR;	

The DEBUG_CMPTK_DRVR procedure is called to insert an instruction into the PDF to cause a debugging facility inside the Computek driver program to be turned on. This debugging facility causes the hexadecimal representation of the ASCII terminal control characters being sent to the Computek to be displayed at the user's console.

SEND_PDF;	

The SEND_PDF command is used to mark the end of an image. This instruction is inserted into the PDF and is used by the COMPIL procedure to denote when to stop sending PDF commands to the Computek driver program.

HTEXT	(N, 'STRING');

The HTEXT command inserts an instruction into the PDF to write the character string STRING on the screen, starting at the current cursor position, in a horizontal directio left to right. N is the integer number of characters in STRING, and STRING must be an <u>even</u> <u>number</u> of characters in length. A maximum of 42 characters per horizontal line is permitted.

Note: Although of no importance when using the command, only 22 characters are stored with any HTEXT command PDF record. If STRING is larger than 22 characters, the remaining characters are stored in a continuation text (CTEXT) PDF entry immediately following the HTEXT command.

VTEXT	(N, 'STRING');

The VTEXT command inserts an instruction into the PDF to write the character string STRING on the screen, starting at the current cursor position, in a downward vertical direction. N is the integer number of characters in STRING, and STRING must be an <u>even number</u> of characters in length. A maximum number of 24 characters per vertical line is permitted.

Note: Although of no importance when using the command, only 22 characters are stored with any VTEXT PDF record. If STRING is larger than 22 characters, the remaining characters are stored in a continuation text (CTEXT) PDF entry immediately following the VTEXT command.

COMPIL	(I);

The COMPIL procedure is called to send a specified portion of the PDF to the Computek terminal for display. The integer argument I signifies the PDF index of the first display instruction of the image. COMPIL begins at I and continues translation and transmission of the image until a SEND_PDF command is reached.

Image Transformation Commands.

The procedures that comprise this group of commands are used to move, scale, and/or rotate any part or all of an image residing in the PDF. These procedures generate a global transformation matrix, T_MATRIX, of size 4X4 for three-dimensional transformations or 3X3 for two-dimensional transformations, which can be applied to any portion of the PDF to produce new transformed values for the entries. All of the transformation routines described below are compatible with each other; that is, any number of the transformations can be applied to a given portion of the PDF to produce the desired effect. Each transformation routine creates a temporary local matrix which will accomplish the transformation. This temporary matrix is then multiplied by the matrix T_MATRIX, resulting in a concatenation of the transformations. Although the transformations destroy the PDF entries to which they are applied, the original entries are recoverable by applying the inverse transformations to the PDF. For those who are interested in further information as to how image transformations are obtained, [New73] is an excellent reference.

INIT_T_MATRIX	(DIMENSION);

The INIT_T_MATRIX procedure is called to initialize the transformation matrix T_MATRIX to a 3X3 identity matrix (DIMENSION equals integer value 2) or a 4X4 identity matrix (DIMENSION equals integer value 3). This is usually the first call before a sequence of transformations. The argument DIMENSION (integer value 2 or 3) is stored in a variable common to all transformation procedures, and it is used to determine the dimension of the transformations.

TRANS	(XD, YD, ZD);

The TRANS procedure is called to translate (move) an image along the X, Y, and Z axis by the amounts specified by the real-valued arguments. A temporary matrix, the size of which depends on the dimension, is created which is capable of translating an image. This matrix is then multiplied by T_MATRIX to allow concatenation.

$$\text{2-D} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ XD & YD & 1 \end{bmatrix} \qquad \text{3-D} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ XD & YD & ZD & 1 \end{bmatrix}$$

Note: The TRANS command of the FORTRAN version of the CSGP moves the X, Y, and Z axis to obtain a movement of the object. The SPASCAL TRANS procedure moves the image and not the axes.

SCALE	(XS, YS, ZS);

The SCALE command performs scaling on an image by the factors XS, YS, and ZS (real values) in the respective axis. A scale of 2.0 doubles the size of the image. A scale of 1.0 leaves the size of the image the same, and a scale of 0.5 reduces the size of the image by one half.

The SCALE procedure produces a temporary matrix, the size of which depends on the dimension, which is capable of scaling an object. This matrix is then multiplied by T_MATRIX to perform concatenation.

$$\text{2-D} = \begin{bmatrix} XS & 0 & 0 \\ 0 & YS & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{3-D} = \begin{bmatrix} XS & 0 & 0 & 0 \\ 0 & YS & 0 & 0 \\ 0 & 0 & ZS & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

REFLEC	(XI, YI, ZI);

The REFLEC command is called to cause a reflection of an image or for an axis interchange. Each of the arguments must be either a positive or negative 1.0; if 1.0, the axis is not interchanged; if -1.0, the axis is interchanged. The procedure produces an appropriately sized matrix that, when applied to the PDF, produces the desired reflection. This matrix is then multiplied by T_MATRIX to allow concatenation.

$$\text{2-D} = \begin{bmatrix} XI & 0 & 0 \\ 0 & YI & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{3-D} = \begin{bmatrix} XI & 0 & 0 & 0 \\ 0 & YI & 0 & 0 \\ 0 & 0 & ZI & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note: Only positive X and Y coordinates are displayed on the Computek terminal. Thus, a reflected image will often need to be moved in order to view the resultant picture. An image move is accomplished by the TRANS command.

ROTATE	(I, θ);

The ROTATE procedure is called to perform rotation of an image about the Ith axis, where 1 = X axis, 2 = Y axis, and 3 = Z axis. The argument I must be given as the integer value 1, 2, or 3, and it denotes the axis of rotation as just stated. The argument θ must be a real value (positive or negative), and it indicates the number of degrees of rotation.

The rotation procedure produces a temporary matrix capable of rotating an image. The matrix is then multiplied by T_MATRIX to concatenate the rotation to any existing transformations.

$$\text{2-D} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{3-D (I=1)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{3-D (I=2)} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{3-D (I=3)} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note: Even though for two-dimensional rotations an axis of rotation is meaningless, an axis of rotation integer argument must still be included. Also, the rotations performed by ROTATE are only about the point (0,0). Rotations about any point can be performed by moving (TRANS) the object to (0,0), making the rotation (ROTATE), and then moving the object back to its original position.

APPLY_T_MATRIX	(I1, I2);

The procedure APPLY_T_MATRIX is called to apply the transformation(s), represented by the transformation matrix T_MATRIX, on a specified portion of the PDF. The integer arguments I1 and I2 represent the first and last entries in the PDF to which the transformations are to be applied.

PDF Output Commands.

The PDF output commands allow the user to dump the contents of all or a specified portion of the PDF to the user's console or the line printer. The index, name, and operands (excluding the string length of text commands) of the PDF instruction are provided. The following is the format of the output:

Format:

(index) (three-character-coded command name) (operands)

Coded Command Names:

CLR	=	CLEAR
EMD	=	EMODE
WMD	=	WMODE
DBG	=	DEBUG_CMFTK_DRVR
MOV	=	MOVE
VEC	=	VECTOR
HTX	=	HTEXT
VTX	=	VTEXT
CTX	=	CTEXT
SND	=	SEND_PDF

DUMPPDF_CONSOLE	(I1, I2);

The DUMPPDF_CONSOLE command is used to output the contents of the PDF to the user's CRT console. The I1 argument (integer) is the PDF index where the dump is to start, and the I2 argument (integer) is the PDF index where the dump is to end. An I2 argument of 0 specifies a desire to stop the dump when a SEND_PDF command is encountered. Whatever the arguments specified, however, the dump is halted prematurely if a total of 44 commands are dumped (the console screen is full) or when the next available location in the PDF (pointed to by INDEX) is reached, whichever occurs first.

DUMPPDF_PRINTER	(I1, I2);

The DUMPPDF_PRINTER command is used to output the contents of the PDF to the printer. The I1 integer argument is the PDF index where the dump is to begin, and the I2 integer argument is the PDF index where the dump is to end. An I2 argument of 0 signifies a desire to stop the dump when a SEND_PDF command is encountered. Whatever the arguments specified, however, the dump is halted prematurely if the next available location in the PDF (pointed to by INDEX) is reached.

Error Messages.

There are five messages that may be displayed on the user's console by the CSGP software to provide notification of errors made by the user while using CSGP commands or of system errors occurred while performing the output of data to the Computek terminal or the printer. The messages and their explanations are as follows:

***CSGP - INDEX > PDF UPPER BOUND

This error message signifies that an overflow of the PDF has occurred; that is, the value of INDEX equals 601, and one or more attempts have been made to enter commands into the PDF. When this condition occurs, the CSGP sets INDEX equal to 600 and inserts the command causing the overflow at that entry. The previous contents of PDF(600) are lost.

***COMPIL - ARGUMENT >= INDEX

This error message signifies that the argument specified for the COMPIL command is greater than or equal to the value of INDEX. This means that an attempt was made to display at the Computek an invalid portion of the PDF. The recovery action taken by the CSGP is to reset the value of the argument to one, so that the translation of the command begins at the first PDF entry.

***COMPIL - INDEX REACHED BEFORE SEND_PDF

This error message signifies that while translating PDF commands the end of the valid portion of the PDF was reached, and no SEND_PDF command to mark the end of the image was found. The CSGP recovers by assuming that the end of the image has been reached.

***PUT_DISPPAGE - ABNORMAL OUTPUT COMPLETION

This error message signifies that the Computek driver program, called by the COMPIL procedure, did not terminate successfully.

***PUT_PRNTPAGE - ABNORMAL OUTPUT COMPLETION

This error message signifies that the printer program, a system program that runs the line printer, did not terminate successfully when called by the DUMPPDF_PRINTER procedure.

C. SAMPLE PROGRAM

To utilize the graphics services of the CSGP, the user must write, compile, and execute an SPASCAL program. However, the user need only write a relatively small main program using the commands (SPASCAL statements) described in the preceding section. Once the user program is created, it must be appended to the CSGP to form a complete SPASCAL program. The program must then be compiled error-free before it may be finally executed to produce the results dictated by the CSGP commands.

The following notation will be used when describing, during this section, commands to be entered at the user's console:

'CR' represents pressing the carriage return key.

'BK' represents pressing the break key.

" " delimits system responses.

There are several steps that must be taken before the user is able to create a CSGP program. Signing on the system, obtaining access to a disk containing a copy of the CSGP text file, and gaining access to the KSU PASCAL INTERPRETER must be accomplished. These steps are relatively easy to perform and usually require but a few commands to be entered at the user's console. The details of these commands can be obtained from INTERDATA 8/32 operations personnel. Access to the Computek terminal must next be obtained. Insure first that the Computek is connected to the computer, i.e., the plug marked "INTERDATA" is connected to the plug marked "COMPUTEK," and that the thumb wheel switch in back of the Computek is set to "4" (1200 baud). Then the user's Pascal task must have Logical Unit 5 assigned to PA14 (the Computek terminal device). This is accomplished by entering the following sequence of commands at the user's console:

```
'BK'
PA 'CR'
"TASK PAUSED"
CL 5 'CR'
AS 5,PA14: 'CR'
CO 'CR'
```

The user is now ready to create a CSGP program by using the KSU PASCAL Editor (EDIT) to input a text file from the console, appending the text file to the CSGP (CONCAT), and compiling the combined text file (SPASCAL) to form an object code file of the user program. The instructions to perform these functions are thoroughly explained in [Nea77]. The text file must begin with the SPASCAL reserved word "BEGIN" (no following delimiter) and end with the reserved word "END." (must have the following period). Each CSGP command must be followed by a semicolon. If after compiling a complete CSGP program the system returns with the response "COMPILATION ERRORS," the compile listing of the program must be used to discover the location and cause of the error(s), and the KSU PASCAL Editor must be used to correct the error(s). The name of the user object code file typed on the console, followed by a carriage return, starts execution of the user program.

When compiling the text file containing the CSGP and the user program from a disk using the SOLO utility command SPASCAL, a destination medium must be specified to receive the listing of the text file followed by any compile errors that were discovered. At the time of this writing, there is no means implemented on KSU's INTERDATA 8/32 to prevent the user from receiving the listing of the CSGP in addition to the application program. There is currently work underway within the KSU Computer Science Department to build a precompiler which, when implemented, will allow the CSGP to be stored in a partially compiled form. A user will be able to append a CSGP application program to the CSGP, compile the result, and receive only a listing of the user's portion of the program.

The following commands are to be used when signing off the system:

```
'BK'
SIGNOFF 'CR'
"ELAPSED TIME=01:37:25
 OS/32MT TERMINAL MONITOR 00-01"
```

Signing off also causes any output directed to the printer during the console session to be printed. If the printer output is desired before signing off, the following command sequence is used:

```
'BK'
PA 'CR'
"TASK PAUSED"
CL 4 'CR'
AS 4,PR: 'CR'
CO 'CR'
```

An example console session is presented below to illustrate the use of CSGP commands to perform a simple image construction and transformation. The picture shown in Figure II.1(a) will be built. It will then be scaled, translated, and rotated to obtain the resultant image shown

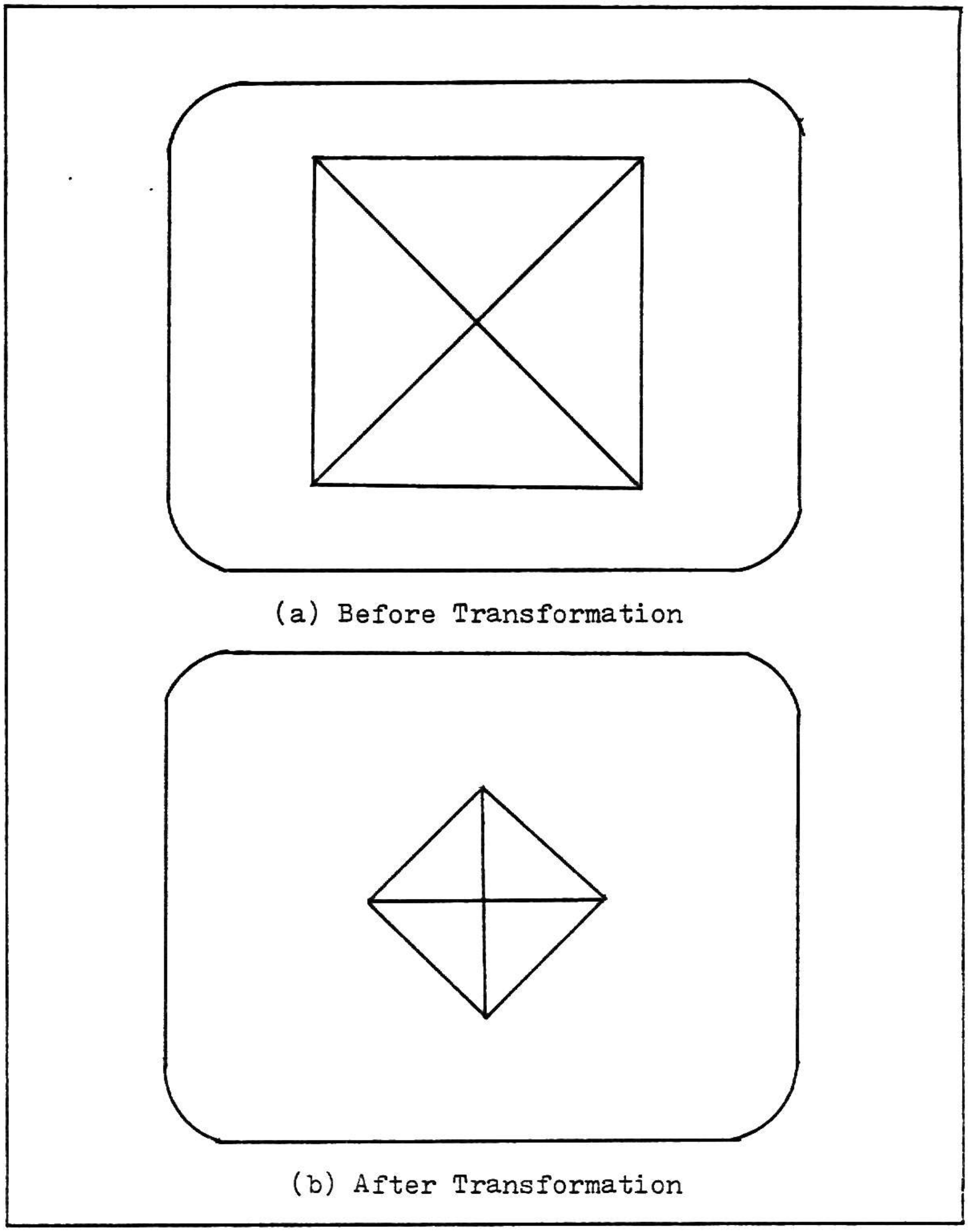

Figure II.1 Display of CSGP Sample Program.

in Figure II.1(b).

Console Session: (There is an implicit carriage return following each user-entered command.)

```
          SESSION                              COMMENTS

SIGNON DAN,13,CS720

PASCAL USR6:CS720C,PR:

"DO:"

EDIT(NULL,SAMPLETEXT)          (SAMPLETEXT is the name of the
                                text file that will hold the
                                CSGP program)

"EDIT:"

CR                             (enter input mode; CR = create
                                text edit command)

BEGIN                          (first line of user program)

START;

SET_SCALE(32.0,36.57);         (following coordinates to be
                                specified in inches)

CLEAR;

MOVE(1.0,2.0,0.0);

VECTOR(1.0,6.0,0.0);

VECTOR(5.0,6.0,0.0);

VECTOR(5.0,2.0,0.0);

VECTOR(1.0,2.0,0.0);

VECTOR(5.0,6.0,0.0);

MOVE(1.0,6.0,0.0);

VECTOR(5.0,2.0,0.0);

SEND_PDF;                      (mark the end of the image)

INIT_T_MATRIX(2);
```

```
TRANS(-3.0,-4.0,0.0);             (move center of object to
                                   (0,0))

SCALE(0.5,0.5,0.0);               (scale object by one half)

ROTATE(1,45.0);                   (rotate object 45 degrees)

TRANS(3.0,4.0,0.0);               (move object back to its
                                   original position)

APPLY_T_MATRIX(1,VAL_INDEX - 1);

COMPIL(1);                        (display the resultant object
                                   at the Computek terminal)

END.                              (last line of user program)

'CR'                              (this carriage return follows
                                   the carriage return after the
                                   END. and causes the input
                                   mode to be exited)

EN                                (exit editor)

"DO:"

CONCAT(CSGP,SAMPLETEXT,SAMPLETEXT) (attach CSGP to user
                                        program)

"DO:"

SPASCAL(SAMPLETEXT,PRINTER,SAMPLEOBJ) (compile CSGP program)

"DO:"

SAMPLEOBJ                         (start execution of CSGP
                                   program; the resultant image
                                   now appears on the Computek)
```

III. CSGP (SPASCAL) SOFTWARE

A. STRUCTURE

The structure of the software is that of an SPASCAL program. It consists of a set of standard prefix declarations (discussed in Chapter IV) followed by the main program. The body of the main program is, in fact, the user program, which is preceded by the usual global constant, type, variable, and routine declarations (Figure III.1). The graphics functions provided by the CSGP are performed by the routines, which are accessed by the user program during execution. The CSGP routines are divided into four functional groups. These groups and their constituent routines are listed below.

Image Construction Routines.

START.

SET_INDEX.

VAL_INDEX.

SET_SCALE.

EMODE.

WMODE.

CLEAR.

SEND_PDF.

MOVE.

VECTOR.

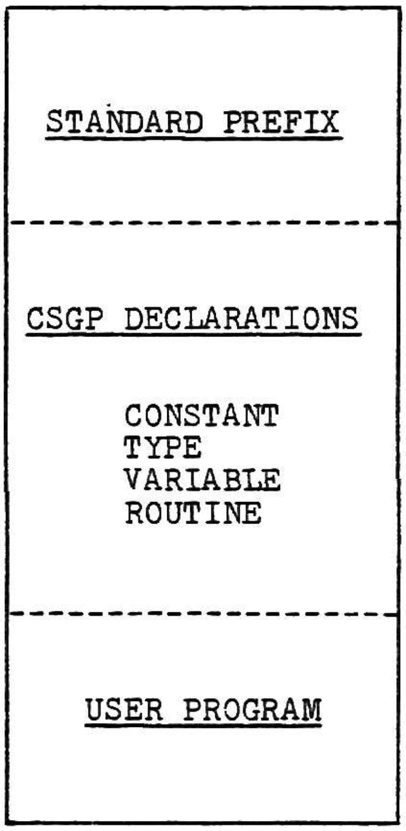

Figure III.1 CSGP Program Structure.

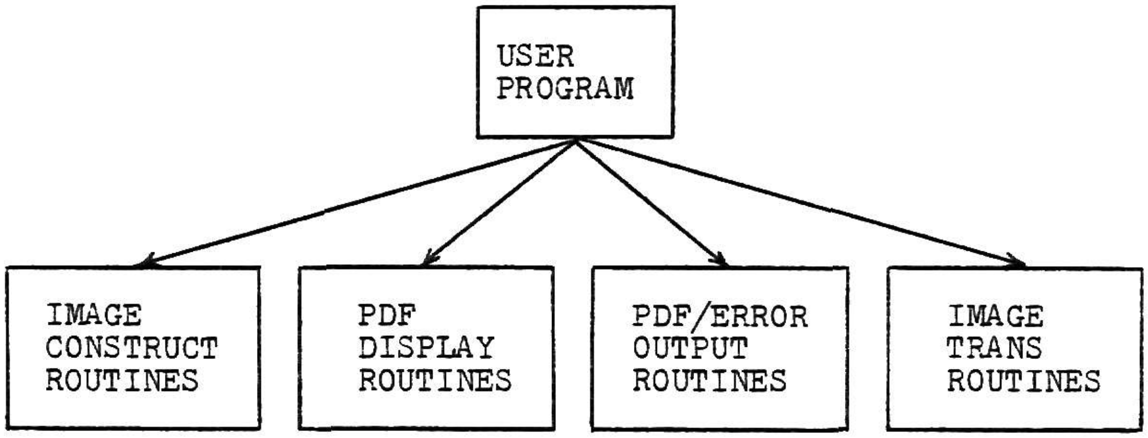

Figure III.2 CSGP Access Graph.

PUTTEXTINPDF.

HTEXT.

VTEXT.

DEBUG_CMPTK_DRVR.

PDF Display Routines.

COMPIL.

LOAD_CMPTK_DRVR.

PUT_DISPPAGE.

GET_TEXT.

PACK_CHAR.

PACK_REAL.

PDF/Error Output Routines.

DUMPPDF_PRINTER.

DUMPPDF_CONSOLE.

ERROR.

DISPSTRNG.

LOAD_PRINTER_PROG.

PUT_PRNTPAGE.

GET_TEXT_OUT.

INT_TO_STR.

REAL_TO_STR.

Image Transformation Routines.

INIT_T_MATRIX.

MATMUL.

APPLY_T_MATRIX.

TRANS.

SCALE.

REFLEC.

ROTATE.

TRIG.

The CSGP routines are accessable to the user program by procedure or function call (Figure III.2). The image construction routines are those procedures that insert commands into the PDF. Almost all of them are called by the user program (Figure III.3) except for PUTTEXTINPDF, which is called by HTEXT and VTEXT to insert alphanumeric text into the PDF.

The PDF display routines utilize COMPIL to convert the PDF into the appropriate form and display the information at the Computek. In Figure III.4 the dotted line from PUT_DISPPAGE to CSGPCOMPUTEK is to illustrate the data transfer between the CSGP program, which resides in SOLO's job process partition, and the Computek driver program located in the output process partition. The second dotted line shows the subsequent data transfer from CSGPCOMPUTEK to the Computek terminal device.

The PDF/error output routines consist of those procedures required to output the contents of the PDF to the user's console or the printer and to output error messages to the user's console. As in Figure III.4, the dotted lines in Figure III.5 denote data transmission between the job and output process partitions and between the output partition

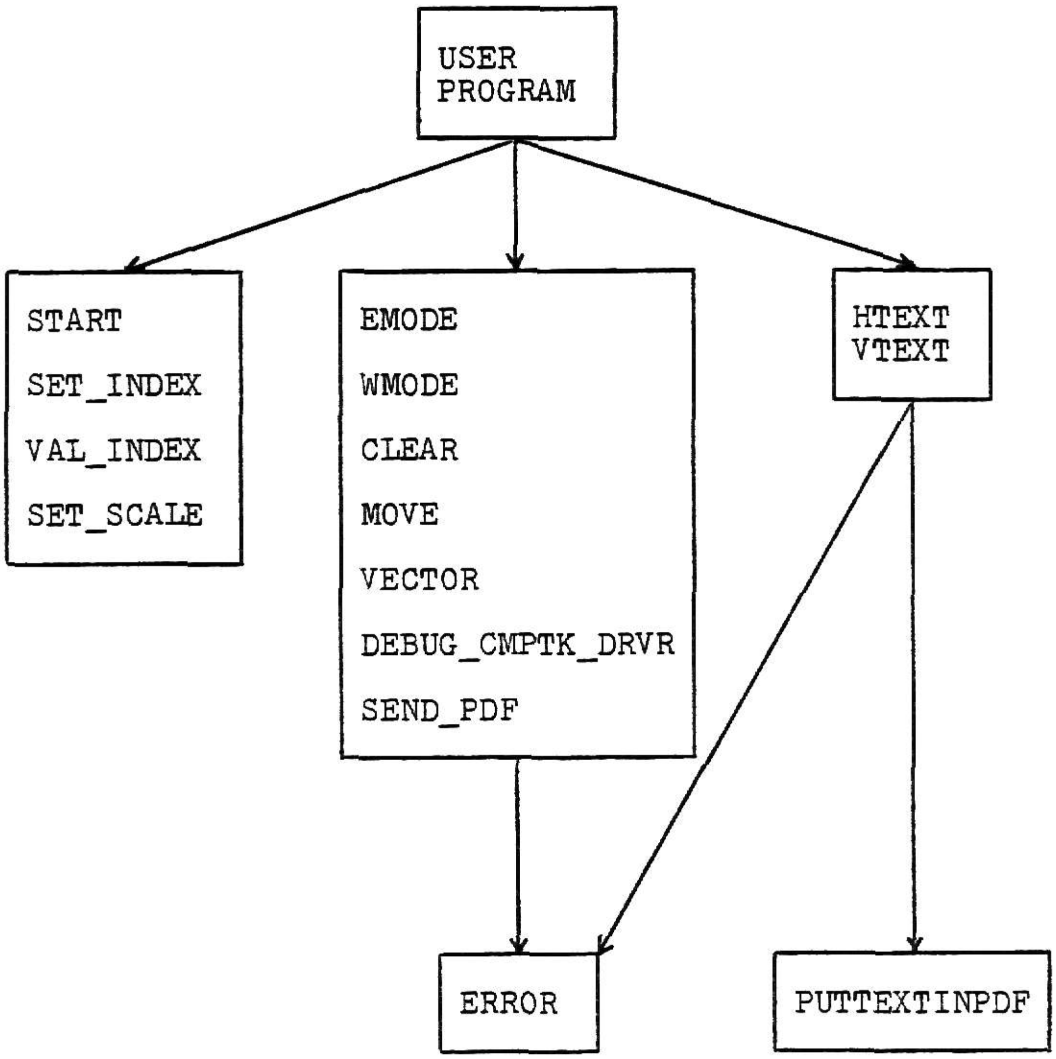

Figure III.3 Image Construction Routines Access Graph.

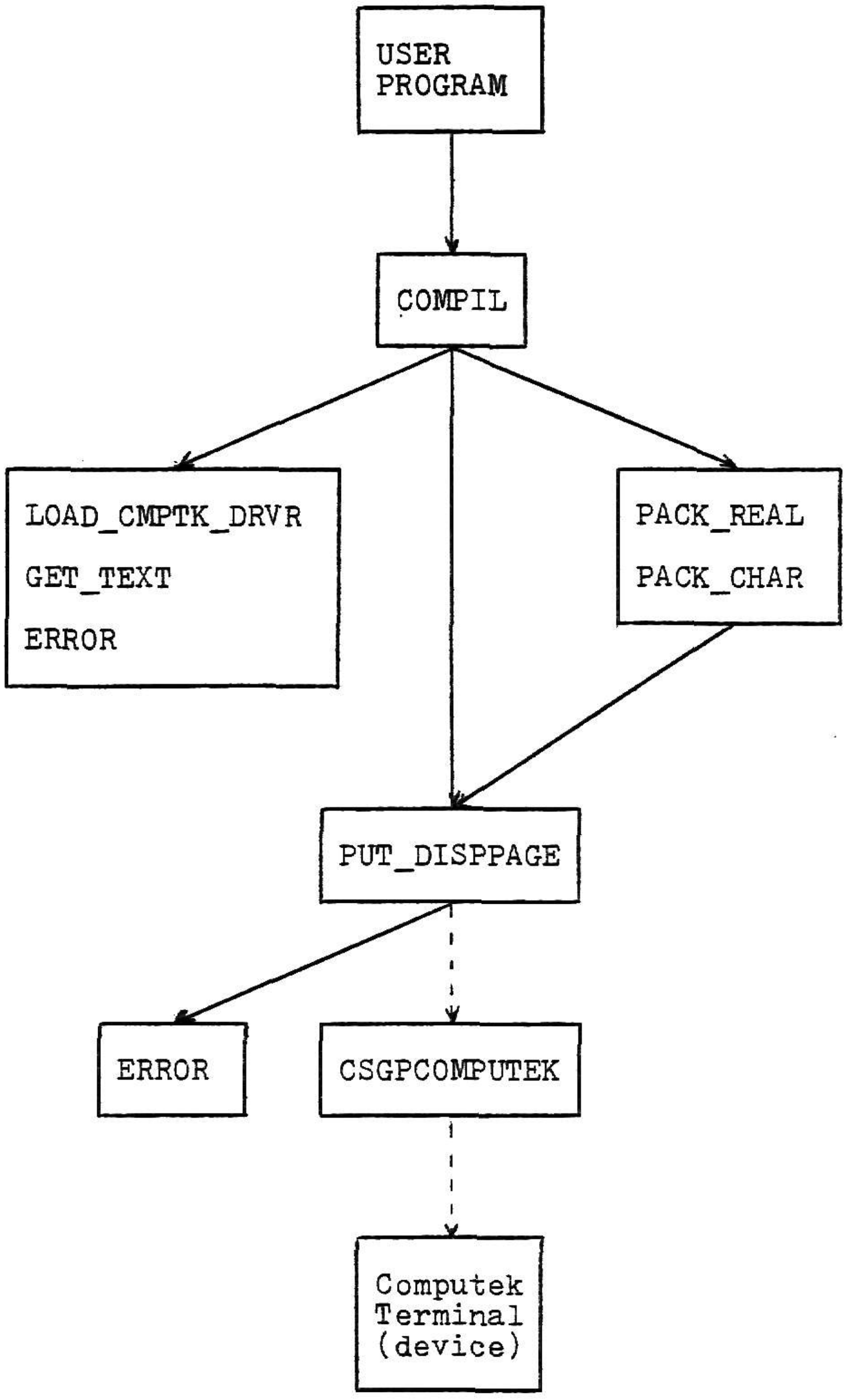

Figure III.4 PDF Display Routines Access Graph.

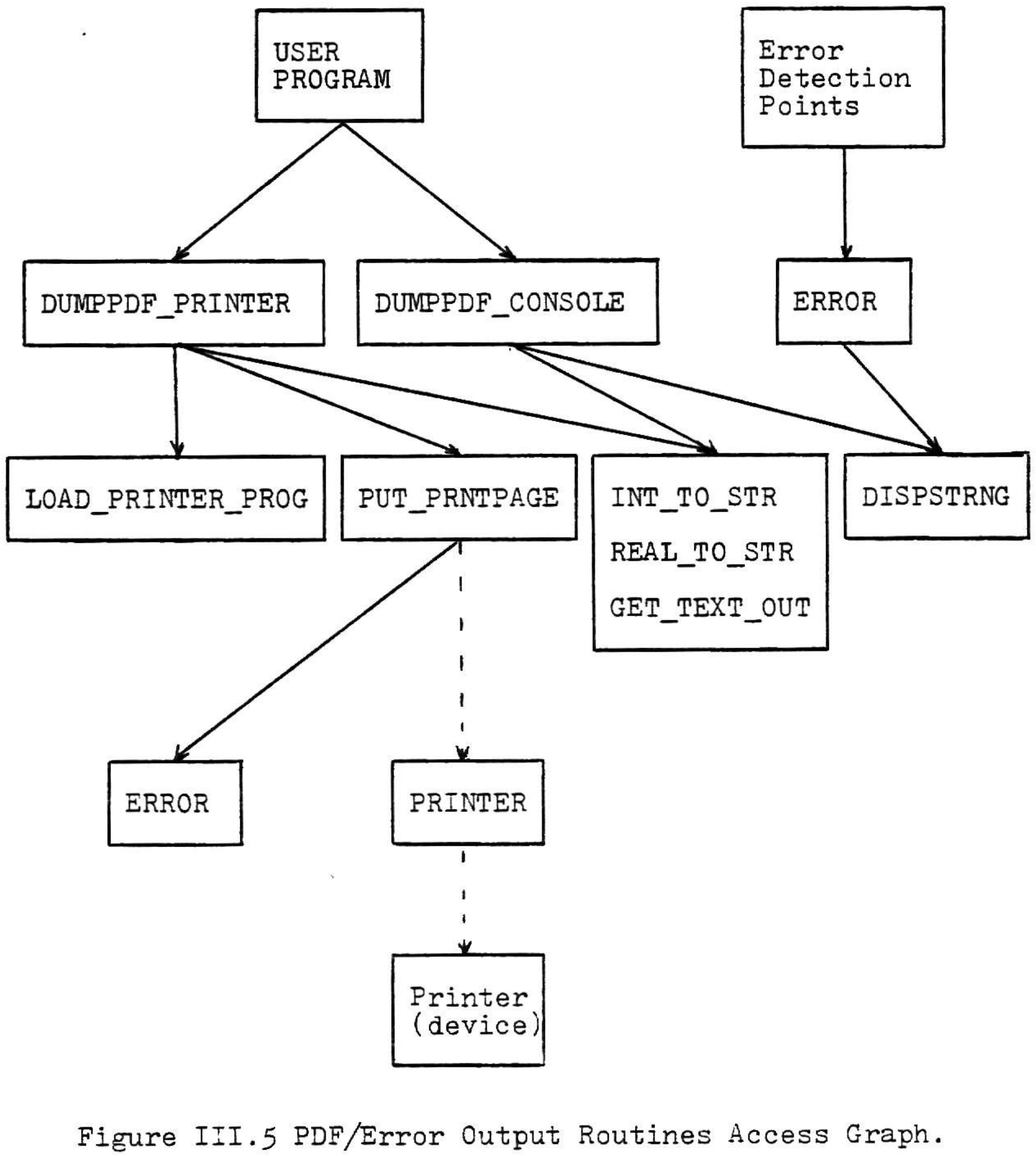

Figure III.5 PDF/Error Output Routines Access Graph.

and the output device. The user program calls DUMPPDF_PRINTER and DUMPPDF_CONSOLE to output the PDF, and ERROR is called, within the CSGP routine where the error is discovered, to output an error message.

The image transformation routines are the routines which perform a particular transformation upon the contents of the PDF. As illustrated in Figure III.6, the user program calls INIT_T_MATRIX to initialize a transformation matrix. The user may then call TRANS, SCALE, REFLEC, or ROTATE to create a particular transformation matrix, which is concatenated to any previous transformations by accessing MATMUL. ROTATE in building its matrix must also have access to the TRIG function to calculate sine and cosine values. Finally, the user program calls APPLY_T_MATRIX to apply the current transformation matrix to a specified portion of the PDF.

B. MAJOR DATA STRUCTURES

There are four vital data structures utilized by the CSGP software. The variables PDF, DISPPAGE, DISPLAY_PAGE, and OUTPAGE hold a list of display instructions during its translation from commands specified by the user to the actual ASCII characters transmitted to the Computek terminal.

The most important of these data structures is the PDF (Pseudo Display File), which is used to store the display instructions in the same order as they are encountered in the user program. The PDF is implemented as an array of

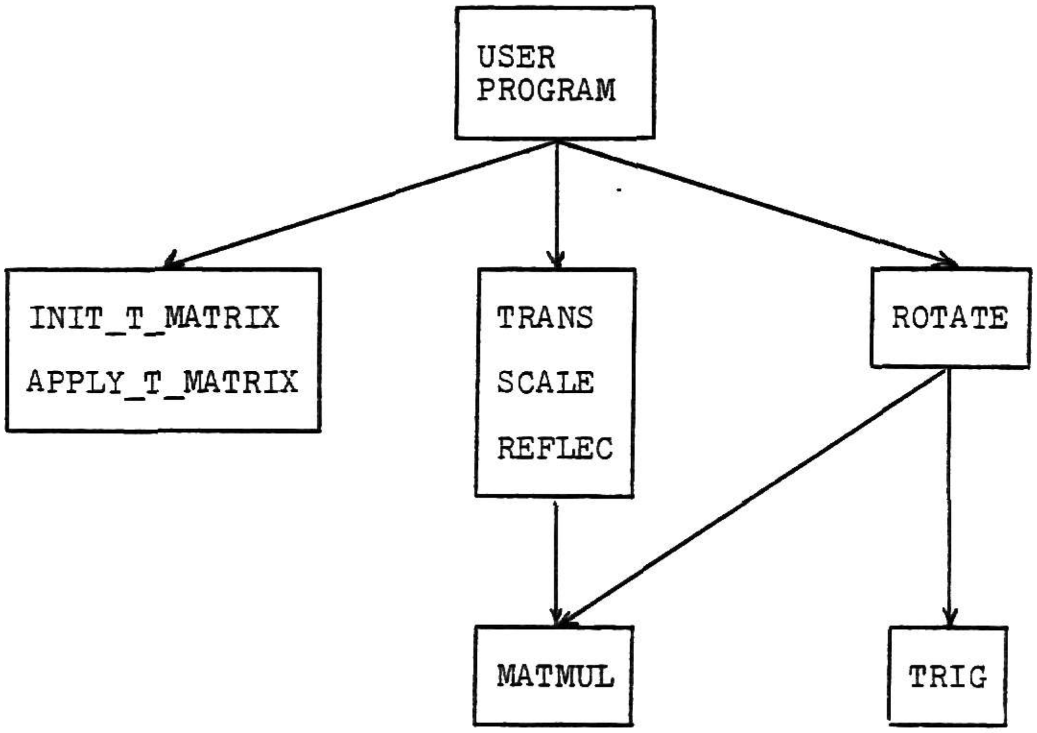

Figure III.6 Image Transformation Routines Access Graph.

variant records (Figure III.7), each record containing a particular display operation code along with its associated operands. The operand field is the varying component of the record, the contents of which depend on the particular class of operation being stored. If the operation is a MOVE or VECTOR command (PDFLINE tag), the operand field consists of the X, Y, and Z coordinates. If the command is HTEXT, VTEXT, or CTEXT (PDFTEXT tag), the operands are a character string and its length, and if the command is EMODE, WMODE, CLEAR, DEBUG_CMPTK_DRVR, or SEND_PDF (PDFMODE tag), then the operand field is null.

In order that the PDF array make the most efficient use of memory, the continuation text instruction (CTEXT) was added to hold additional horizontal or vertical text characters over the 22 maximum allowed per PDF record. In the case of SPASCAL variant records, memory is statically allocated for the maximum size component. Thus when declaring variant components, the objective is to make them as equal in size as possible. Besides the HTEXT/VTEXT class of instruction, the only other non-null variant is the MOVE/VECTOR class, which must be allocated a total of 12 words of storage for its X, Y, and Z real-valued coordinates (4 words per real). The text instruction class contains an integer string length (1 word per integer) and a character string, which is left with 11 words or space for 22 characters (2 characters per word).

OPCODE	0..9 (integer)
TAG	PDFLINE, PDFTEXT, PDFMODE (enumeration)
PDFLINE:	X Y Z Coordinates (real)
PDFTEXT:	String Length (integer) Text (array of 1 to 22 characters)
PDFMODE:	Null Component

Figure III.7 PDF Variant Record Format.

The variable DISPPAGE is a linear array of 256 integer elements. After the PDF is translated into integer form by the COMPIL procedure, DISPPAGE is used to store the display instructions during their transmission, 256 integers at a time to the Computek driver program.

The display instructions are received at CSGPCOMPUTEK into DISPLAY_PAGE, a linear array the same size and type as DISPPAGE. The commands are analyzed again and are translated further into a sequence of ASCII characters. The characters are loaded into a linear array of one-character elements, OUTPAGE, before they are finally transmitted to the Computek terminal for display.

C. ROUTINES

Before discussing the code comprising the four functional groups of CSGP routines, a note is in order concerning the declaration of parameters in SPASCAL. In Pascal there are two possible types of parameter linkage for procedures: by access value and by reference. For each parameter that is declared for a routine, the type of parameter linkage to be used must also be declared. Parameters using linkage by access value are called constant parameters, and parameters using linkage by reference are called variable parameters. A routine can use the values of both constant and variable parameters, but it can only change the value of a variable parameter. The distinction between constant and variable parameters is made by omitting or writing the symbol <u>var</u> before the parameters in the declaration parameter list.

Image Construction Routines.

The routines START, SET_INDEX, and VAL_INDEX manipulate INDEX, which is the global index into the PDF where the next command can be inserted. START initializes INDEX to the value of the first PDF index. SET_INDEX changes INDEX to the value of its integer parameter, and VAL_INDEX, a function procedure, returns the current value of INDEX. START also initializes a boolean overflow flag (discussed in conjunction with the ERROR routine) to false and the global coordinate scale variables, X_SCALE and Y_SCALE, to their default values of 1.0. The SET_SCALE routine changes the value of X_SCALE and Y_SCALE in accordance with its two real-valued parameters.

The next five routines, EMODE, WMODE, CLEAR, SEND_PDF, and DEBUG_CMPTK_DRVR, enter display commands into the PDF with a null variant component record. The opcode of the command (0, 1, 6, 7, or 8 respectively) is entered into the record along with the PDFMODE tag. Before the PDF record is assigned, however, INDEX is checked to insure it is in the proper range, with a call to the ERROR procedure if it is not. After the PDF assignment is made, INDEX is incremented by one.

The MOVE and VECTOR procedures insert commands into the

PDF to move the cursor and to draw a line. As described above, INDEX is first checked for validity. The PDF record subscripted by INDEX is then assigned an opcode of 2 (MOVE) or 3 (VECTOR), a tag of PDFLINE, and a variant component of the X, Y, and Z real-valued coordinates as received by parameter transmission. Again as above, the PDF index is then incremented before returning to the user program.

The final three procedures of the image construction routines are involved with entering horizontal and vertical text commands into the PDF, The HTEXT and VTEXT procedures contain the formal parameters STRNG_LEN and STRNG through which are received a character string and its length. Once again INDEX is checked and used as a subscript into the PDF. The opcode 4 (HTEXT) or 5 (VTEXT) and the PDFTEXT tag is assigned, and then the character string and its length are used as the actual parameters for a call to the PUTTEXTINPDF procedure. This procedure takes the character string (containing a maximum of 42 characters for HTEXT and 24 for VTEXT) and assigns the first 1 to 22 characters to the text string portion of the variant component of the record and the appropriate number of characters to the string length portion. If there are any characters remaining, a continuation text entry is added to the PDF with the appropriate check and increment of INDEX. The continuation text instruction possesses an opcode of 9, a tag of PDFTEXT, and a character string and length variant component. The

PUTTEXTINPDF procedure can actually handle an arbitrary number of CTEXT entries, but with the current limits of 42 and 24 characters programmed for HTEXT and VTEXT commands, only one is necessary. PUTTEXTINPDF returns to HTEXT or VTEXT before the final increment is made on INDEX, and the return is made to the user program.

<u>PDF Display Routines</u>.

The PDF display routines execute under the control of the procedure COMPIL. This procedure is responsible for translating the PDF, starting at a parameter-supplied index, into a form appropriate for input to the Computek driver program and for performing the transmission to the driver. Before the translation begins, the starting index is verified as being valid (less than INDEX), and ERROR is called if it is not valid. The translation ends when a SEND_PDF command is encountered in the PDF, or the last valid display instruction (PDF index of which is one less than INDEX) has been translated. In the latter case, an error message is generated since the SEND_PDF command is expected at the end of an image.

Before starting translation of the PDF, COMPIL calls LOAD_CMPTK_DRVR, which loads the Computek driver program into SOLO's output process partition. This is accomplished by calling the WRITEARG procedure (declared in the standard prefix) with the proper arguments to notify the IO program which driver to load into what partition. Interprocess data transmission in SOLO is discussed further in Section B of Chapter IV.

The PDF translation consists of sequentially examining each opcode, determining via a case statement the basic type of the command, and then taking the appropriate action for that type. For each of the three basic types of instructions, the first action is to enter the opcode of the command into the linear array DISPPAGE (described in Section B of this chapter) via a call to PUT_DISPPAGE. The function of PUT_DISPPAGE is to enter the integer value of its formal parameter INTVAL into DISPPAGE, increment the DISPPAGE index, and transmit full DISPPAGEs (256 integers entered) to CSGPCOMPUTEK. This transmission is made by calling the SOLO prefix procedure WRITEPAGE. Each time DISPPAGE is sent to the Computek driver, the DISPPAGE index is reset to zero. In the case of PDFMODE commands, the call to PUT_DISPPAGE completes translation of the instruction. If at the end of the PDF translation SEND_PDF was not the last command examined, a SEND_PDF opcode is sent to PUT_DISPPAGE to signal the procedure to send the last page to CSGPCOMPUTEK and terminate transmission. After the last page is sent, a call to the SOLO prefix procedure READARG retrieves an output completion flag, which signifies normal (true) or abnormal (false) output completion. If the flag is false, ERROR is called to output an appropriate message to the user's console.

For MOVE and VECTOR commands, the last translation step is to retrieve the X and Y coordinates from the PDF, multiply them by the current scale values (X_SCALE and Y_SCALE), and send the results via procedure call to PACK_REAL. Through the use of universal type declarations of its formal parameters, PACK_REAL converts the two real-valued coordinates into two four-element integer arrays. Beginning with the X coordinate array, the procedure enters the eight integers one-by-one into DISPPAGE via PUT_DISPPAGE.

There are two more actions that must be performed to finish the translation of text instructions. First, the GET_TEXT procedure is called in order to retrieve the character string associated with the HTEXT/VTEXT command. This procedure loads into a variable parameter, of character array type, the text stored with the command plus the text from any continuation text (CTEXT) instructions. End of medium characters (EM) are entered as the last two character elements of the array. Upon return to COMPIL, the accumulated text is used as the argument for a call to PACK_CHAR. This procedure, with the same universal parameter declaration technique used in PACK_REAL, converts the character string into an array of integers and stores these integers into DISPPAGE by repeated calls to PUT_DISPPAGE.

The code for the Computek driver program is included in Appendix B. CSGPCOMPUTEK began as an SPASCAL program written by M. Neal for a computer graphics project also involving the

Computek terminal [Nea78]. CSGPCOMPUTEK is primarily the same program but for a few minor revisions which are dealt with in Chapter IV.

<u>PDF/Error Output Routines</u>.

These routines are involved with the output of the contents of the PDF to the printer or the user's console and the output of error messages to the user's console. The three primary procedures of this group of routines (ERROR, DUMPPDF_CONSOLE, and DUMPPDF_PRINTER) are discussed together in that they all perform an output function and share the use of many of the same routines.

The ERROR procedure accepts an integer error code through its constant parameter ERRCODE, and displays on the user's console a particular error message based on the value of the error code. If an ERR1 error code is received, a message informs the user that while attempting to enter a command into the PDF an overflow of that data structure has occurred. At the point of error recognition, the statement immediately after the call to ERROR sets INDEX to the largest valid PDF index (600); such that, the attempted instruction insertion will take place with the contents of the previous entry being lost. After the first recognition of a PDF overflow, an error message is displayed, and an error flag is set to true to prevent multiple overflow messages being displayed upon further instances of errors of this type.

An error code of ERR2 causes the output of a message signifying that the STARTINDEX parameter of COMPIL is greater than or equal to the value of INDEX. The recovery from this error takes place at the point of detection and involves setting a variable (PDFINDEX) to one, so that translation of the PDF begins at the first entry.

The ERR3 error code also involves COMPIL, and it notifies the user with an error message that the last PDF command translated (its index one less than INDEX) was not SEND_PDF. The error recovery action, also accomplished at the point of infraction, enters a SEND_PDF opcode into DISPPAGE, as previously mentioned in the discussion of the PDF display routines.

The final two error codes, ERR4 and ERR5, are sent from PUT_DISPPAGE and PUT_PRNTPAGE, respectively, denoting that the Computek driver program or the line printer program did not terminate successfully. Error recovery other than providing notification to the user is not accomplished.

The ERROR and DUMPPDF_CONSOLE procedures call the DISPSTRNG procedure to display character strings at the user's console. DISPSTRNG accepts as its formal parameters a character string of up to 72 characters in length and displays the characters one-by-one by using a character as the argument for a call to DISPLAY, a procedure defined by the SOLO standard prefix. The null character is used as the last character of the parameter string.

DUMPPDF_PRINTER and DUMPPDF_CONSOLE operate in basically the same manner. Their constant parameters, STARTINDEX and ENDINDEX, denote the PDF indices where the dump starts and ends, a zero ENDINDEX signifying that the dump is to continue until a SEND_PDF instruction is encountered. However, there is a limiting factor for DUMPPDF_CONSOLE, in that only a total of 44 commands (filling the entire screen) may be dumped during any one call of the procedure. Also in both procedures, the dump of the PDF automatically ends when the command whose index is one less than INDEX is dumped.

A "while do" loop accomplishes the PDF dump in both the printer and console procedures. Before this loop is entered in DUMPPDF_PRINTER, the printer program is loaded into the SOLO output process partition. This is accomplished in the same manner as in the LOAD_CMPTK_DRVR procedure previously discussed. The PDF dump in the console and printer procedures begins with a call to INT_TO_STR with arguments PDFIND, the index of the command to be output, and CHAR_INT, which will hold the character string representation of PDFIND upon return from the call. INT_TO_STR converts an integer to character form by counting the number of subtractions (of 100, 10, and then 1) from the index it takes to obtain a negative result. Since 600 is the largest possible PDF index, INT_TO_STR only converts integers from 0 to 999. The three counts obtained from the hundreds, tens, and ones place digits of the index. These place digits are converted to characters with the aid of the CHR built-in function and assigned to CHAR_INT with the blank and null characters as the last two in the string.

The next step taken in DUMPPDF_CONSOLE is to send the PDF index to DISPSTRNG for immediate display at the console. For the DUMPPDF_PRINTER procedure, however, the index is sent to PUT_PRNTPAGE. This procedure enters characters into PRNTPAGE, a linear array of 512 characters, and sends the page to the printer program when it is full. PUT_PRNTPAGE accomplishes its function in exactly the same fashion as PUT_DISPPAGE, WRITEPAGE again performing page transmissions. The minor exceptions are that in PUT_PRNTPAGE, characters are being transmitted instead of integers, and receipt of an end of medium character, versus the integer value 7 (SEND_PDF opcode), signifies that the last page is to be transmitted.

After the index of the command has been converted to character form and displayed at the console or placed in PRNTPAGE, the opcode is examined with the aid of a "case" statement, and a three-character-coded name of the command is displayed or placed into PRNTPAGE. For those instructions with no operands (PDFMODE tag), no other actions are necessary. However, MOVE and VECTOR commands require a call to REAL_TO_STR to retrieve their X, Y, and Z coordinates and convert them into a character string. Using the index of the command passed as one of its parameters, REAL_TO_STR retrieves the X coordinate; translates its sign, whole part digits, decimal point, and fractional part digits into characters; assigns these characters to its variable character string parameter; and then repeats the process for the Y and Z coordinates. The process by which a real number is converted to its constituent character digits is the same technique as is used for integers, with subtraction counts taken for 1000.0, 100.0, 10.0, 1.0, 0.1, and 0.01. Leading zeros before the ones place digit are suppressed. The same call to REAL_TO_STR is used for both DUMPPDF_PRINTER and DUMPPDF_CONSOLE, and the resultant character string is made the argument for a call to DISPSTRNG (DUMPPDF_CONSOLE) or PUT_PRNTPAGE (DUMPPDF_PRINTER).

In order to output the operands (excluding text length data) of text instructions, GET_TEXT_OUT is called to retrieve the text from the PDF and append to it a blank character for spacing on the output medium and a null character to delimit the string. GET_TEXT_OUT performs these functions in a straightforward manner with a constant integer parameter PDFIND and a variable character array parameter TEXT_STRNG. As for MOVE/VECTOR commands, TEXT_STRNG is then passed to DISPSTRNG or PUT_PRNTPAGE to be output to the console or printer.

Upon exit from the aforementioned "case" statement, DUMPPDF_CONSOLE directly uses the DISPLAY prefix procedure to send the new line character (NL) to the console to terminate the current line and begin a new line of output after every two commands have been dumped. The local variable containing the index of the current PDF instruction being dumped is incremented by one, and control is passed to the beginning of the "while do" loop to start on the next command. When the loop is finally exited, a DISPLAY(NL) call is again made, if only one command has been output on the current line of the console. The return to the user program is then made.

The DUMPPDF_PRINTER procedure must also output control characters after exit from the "case" statement. Using PUT_PRNTPAGE, it appends carriage return (CR) and line feed (NL) characters after each line of printer output (one command) and a form feed (FF) character after every page of printer output (30 commands). The next iteration of the "while do" loop is made, and upon its exit control returns to the user program.

Image Transformation Routines.

The use of the image transformation routines begin with the procedure INIT_T_MATRIX. Its lone parameter identifies the dimension (two or three) in which following transformations are to take place. The procedure first assigns this dimension to a global variable DIMENSION, which is later used by other transformation routines. An identity matrix of the appropriate size (3X3 for two dimensions and 4X4 for three) is then assigned to a global transformation matrix variable T_MATRIX. This matrix forms the initial transformation matrix upon which following transformations can be concatenated by matrix multiplication.

This matrix multiplication takes place in the MATMUL procedure, whose only parameter consists of a new transformation matrix, NEW_T_MATRIX, to be concatenated to the current T_MATRIX. Essentially, a matrix multiplication of NEW_T_MATRIX times T_MATRIX is performed for 3X3 matrices (DIMENSION = 2) or 4X4 matrices (DIMENSION = 3). Two nested loops are used to conduct the multiplication with a temporary matrix used to hold the intermediate results of the multiplications of NEW_T_MATRIX row vectors by T_MATRIX column vectors. The temporary matrix contains the result upon exit from the aforementioned loops and is assigned to T_MATRIX just before the procedure ends.

The transformations available within the SPASCAL CSGP are accessed by the user by calling the TRANS, SCALE, REFLEC, and ROTATE procedures. The basic scenario implemented by each procedure is the same. Based on the values of the arguments supplied by the user, each of these procedures construct a 3X3 (DIMENSION = 2) or 4X4 (DIMENSION = 3) matrix, that when applied to the PDF performs the particular transformation on the end points of the lines that comprise the screen image. After the matrix is built, it is then used as the argument for a call to the MATMUL procedure, where the matrix is combined with any existing transformations. Control is then returned to the user program.

The three real-valued parameters of the TRANS procedure specify the X, Y, and Z distances the object is to move. If DIMENSION equals 2, the translation transformation matrix is initialized to a 3X3 identity matrix, and the X and Y distance parameter values are assigned to positions (3,1) and (3,2) of that matrix. If DIMENSION equals 3, a 4X4 identity matrix is initialized, and the X, Y, and Z distances are entered into positions (4,1), (4,2), and (4,3). The concatenation of the matrix is then performed before the return to the user program is made.

The SCALE procedure creates a scale transformation matrix using parameter supplied X, Y, and Z scale factors. The values of the 3X3 or 4X4 matrix are initialized to zero. For a two-dimensional matrix, the real constant 1.0 is placed at the (3,3) position and the X and Y scale factors at (1,1) and (2,2). For the three-dimensional case, 1.0 is entered at (4,4) and the X, Y, and Z scale factors at (1,1), (2,2), and (3,3). Concatenation and procedure exit follow.

The reflection transformation matrix created by REFLEC also begins as a zero 3X3 or 4X4 matrix. If transformations are currently taking place in two dimensions, the X and Y axis interchange parameters, whose values are 1.0 or -1.0, are assigned to the (1,1) and (2,2) elements of the matrix, and 1.0 is entered at (3,3). If the current dimension is three, 1.0 is placed at (4,4), and the values of the X, Y, and Z axis interchange parameters are assigned to (1,1), (2,2), and (3,3).

A 3X3 or 4X4 identity matrix is initialized to begin the ROTATE procedure. The sine and cosine functions for the real-valued parameter THETA (angle of rotation in degrees) are then calculated via a function call to TRIG. These values are inserted into the initialized rotation matrix in one of four different schemes, depending on the value of DIMENSION and the parameter AXIS. If DIMENSION equals two, the value of AXIS is ignored, and a 3X3 two-dimensional matrix is the result. If DIMENSION equals three, the value of AXIS specifies the creation of a 4X4 rotation matrix about the X axis (AXIS = 1), Y axis (AXIS = 2), or Z axis (AXIS = 3). Concatenation of the rotation matrix then precedes the return to the user program.

TRIG is the real-valued function that returns the value of the sine or cosine (specified by the SIN_OR_COS parameter) function of the parameter angle THETA, given in positive or negative degrees. The value of THETA is first assigned to THETA1. Via the use of two "while do" loops, THETA1 is mapped into the range 0.0 <= THETA1 < 360.0. The sign of the sine (SIN_SIGN) or cosine (COS_SIGN) result is then initialized to 1.0. The quadrant containing THETA1 is determined, SIN_SIGN or COS_SIGN made negative if appropriate for that quadrant, and THETA1 mapped into the range 0.0 <= THETA1 <= 90.0, where COS(THETA1) = SIN(90.0 - THETA1). If the cosine of THETA is being determined, THETA1 is assigned the value of 90.0 - THETA1. The sine of THETA1 is then found by converting THETA1 into radians and plugging that value into a factored version of the power series approximation equation for the sine function. The result of this calculation is multiplied by SIN_SIGN or COS_SIGN before it is finally returned as the value of the TRIG function.

After the desired transformation(s) have been constructed and stored in T_MATRIX, the procedure APPLY_T_MATRIX is the means by which the PDF is transformed to obtain the resultant effect. The parameters of APPLY_T_MATRIX, STARTINDEX and ENDINDEX, delimit that portion of the PDF where T_MATRIX is to be applied. The body of the procedure consists of a loop containing one "if" statement, which tests the current PDF command for a PDFLINE tag. Only MOVE/VECTOR commands receive manipulation by T_MATRIX in the "then" statement sequence of the "if" statement just mentioned. To apply T_MATRIX to a MOVE/VECTOR command, the X, Y, and Z coordinates (X and Y only, if DIMENSION = 2) are retrieved from the PDF and formed into a vector, a two or three element array. This vector is then multiplied by T_MATRIX to obtain the transformed vector result, which is then stored back into the PDF as the new X and Y or X, Y, and Z coordinates of the MOVE/VECTOR instruction. The PDF may then be displayed in order to view the transformed image on the Computek terminal screen.

IV. METHOD OF PORTING

A. FORTRAN TO SPASCAL CODE CONVERSION

The conversion from FORTRAN to SPASCAL code involved more than a one-to-one mapping of FORTRAN statements to SPASCAL statements. The operations performed by the FORTRAN code were first analyzed and understood. A consideration was then made as to how best the new language could be used to accomplish the same functions. Thus when conducting the code conversion to SPASCAL, the overriding concern was to prevent being constrained by the programming techniques used and the structure of the FORTRAN code.

The major change in data structures that had the most significant impact on the code conversion was the consolidation of the three FORTRAN arrays containing the display instructions into one SPASCAL PDF array of variant records. In the original version of the package, a (4,400) array of reals contains all the PDF opcodes and the coordinates for MOVE/VECTOR instructions. For text instructions, the PDF array holds the opcode and an index into a (2,50) integer array containing the length of the text and the index location of a third array, a 200 element array of characters which holds the text itself. By taking advantage of the SPASCAL record data type, it is possible to define a variant record that contains the opcode for a command and, based on a tag which identifies the type of instruction being stored, its operands. An array of these records is then declared, and the entire PDF data can now be stored in one data structure versus three. This change makes the insertion and retrieval of data from the PDF more intuitive and easier to code.

The benefits from the new PDF data structure were realized during the conversion of the image construction routines. When entering a text command into the PDF via the HTEXT and VTEXT procedures, the insertion of the text characters within PUTTEXTINPDF involves dealing with only one data structure. However, the new PDF did introduce some complexity in that the continuation text PDF command (CTEXT) had to be added, and PUTTEXTINPDF has to create CTEXT entries when a call to HTEXT or VTEXT involves a character string with more than 22 characters. The code conversion of the other image construction routines was fairly straightforward.

The PDF display and console/printer output routines evolved more from the change in computing environments rather than the change in programming languages and, therefore, will be discussed in the next section (Section B) of this chapter. However, the changes made to the method of providing the output of error messages do not involve the new computing environment. In the FORTRAN CSGP, user notification of errors detected by the code is made by the output of an error message within the routine where the error is detected. It was felt that this function of the CSGP could be made more modular and easier to maintain if error message output was confined to one routine. The ERROR procedure was written to accept an error code and output to the console a message based on that code.

The code conversion of the image transformation routines was almost a line-for-line translation but for a few exceptions. In the FORTRAN INIT subroutine, the transformation matrix is always initialized to a 4X4 identity matrix no matter what the dimension specified by the parameter. The transformation matrices for the TRANS, SCALE, REFLEC, and ROTATE subroutines also begin with an initialization of a 4X4 matrix. However, whenever transformation matrices are concatenated using MATMUL and applied to the PDF using DAPPLY, the 4X4 matrix is only used whenever the dimension specified in INIT is three. The 3X3 portion of the 4X4 matrix is concatenated and applied when the global dimension variable is two. The SPASCAL implementation initializes a 3X3 matrix for two-dimensional transformations and a 4X4 matrix for three-dimensional transformations in INIT_T_MATRIX, TRANS, SCALE, REFLEC, and ROTATE. MATMUL and APPLY_T_MATRIX operate in the same manner as their FORTRAN counterparts.

A change to the expected sign of the values of the parameters of the TRANS routine was made. The FORTRAN TRANS subroutine requires the user to specify translation distances in the negative X, Y, and Z directions. This means, in a sense, that the axes are being moved to achieve a movement of the object. For the SPASCAL version of TRANS, it was felt that it would be more intuitive to the user if the object moves rather than the axes. Thus, the translation distances were programmed to be accepted in positive directions along the respective axes.

Another departure from the FORTRAN implementation scheme was the expected values for the parameters of REFLEC. The FORTRAN REFLEC subroutine calls for the value of its IX parameter to equal the integer value -1 if the X axis is to be interchanged and +1 if the X axis is to be undisturbed. The same idea holds for the Y axis (-2 and +2) for the IY parameter and the Z axis (-3 and +3) for the IZ parameter. The goal of the REFLEC routine is to create a 3X3 (two dimensions) or a 4X4 (three dimensions) identity matrix with -1 entered at positions (1,1), (2,2), and (3,3) if the respective X, Y, and Z axes are to be interchanged. A simpler solution used by the SPASCAL REFLEC calls for the real values of -1.0 or +1.0 to be given as the three arguments for the procedure call. The first argument signifies interchange or no change for the X axis, the second argument for the Y axis, and the third argument for the Z axis. Within the procedure the values of the parameters may now be assigned to the appropriate positions of a properly initialized matrix.

The final major code conversion problem of the image transformation routines involved the FORTRAN ROTATE subroutine. The values of the sine and cosine of the angle of rotation are obtained via built-in SIN and COS functions. Since no predefined functions exist (at the time of this writing) with the INTERDATA 8/32 implementation of SPASCAL, the SPASCAL TRIG function had to be written to calculate sine and cosine values.

During the conversion of the FORTRAN CSGP code into SPASCAL, several major differences between the two languages became apparent. Considering first the more positive aspects of the target language, SPASCAL is much more readable than FORTRAN. It took a great deal of time to read some of the original CSGP subroutines and understand how they were performing their functions. Logically indented SPASCAL code is very readable, especially when attempting to determine flow of data and control. The numerous control structures available in SPASCAL were also very useful. The relatively large number of "if...then...go to" constructs in the FORTRAN code that it took to perform a certain sequence of events based on a particular value of a variable were combined into a single "case" statement construct in SPASCAL. This became particularly evident while converting the COMPIL subroutine, where the translation steps for a particular command hinges on the value of its opcode. Some other control constructs in SPASCAL that are useful are the loop structures. While the FORTRAN code is limited to only one basic type of "do" loop, SPASCAL offers the "for to/downto do," "while do," and the "repeat until" loop structures. These made the code conversion a great deal easier and, in some instances, aided the readability of the code. The more powerful data structures and extensive type declaration facilities of SPASCAL were also put to use during the conversion. As mentioned many times previously, the record type and especially the capability of defining variant records were extremely helpful in devising a more natural data structure for the PDF. The programmer defined data types were of benefit in that the type of any variable can be found more quickly and can be specified in a more readable form. And finally, SPASCAL allows the names of identifiers to be up to 80 characters in length versus 6 in FORTRAN, which allows an identifier to be named more distinctly in the context of its function.

As to the more negative features of SPASCAL, it contains considerable limitations in I/O capabilities. In SPASCAL lower levels of I/O control are placed under the responsibility of the programmer. This means that in providing the CSGP with the procedures to dump the PDF and to output error messages, routines had to be written to convert integer and real numbers to character form, append control characters to lines and pages of output data, and send the output data to the appropriate device. The present KSU implementation of SPASCAL is also deficient in the number of predefined functions that are available. For example, the sine and cosine trigonometric functions, which are available as built-in functions in FORTRAN, had to be devised and written in SPASCAL. There was also some inconvenience when manipulating character strings in SPASCAL. SPASCAL character strings are treated as arrays of characters which at times was cumbersome to code and involved a lot of character-by-character transfers when assigning a character string to a string variable of greater size. Character strings must also be comprised of an even number of characters, which was not only a nuisance during the conversion but forces CSGP users to be careful when specifying text arguments for HTEXT/VTEXT commands.

Taking into consideration that I had never before done any Pascal programming and not a great deal of FORTRAN programming before working on this project, I found SPASCAL more to my liking than FORTRAN. It was reasonably easy to learn SPASCAL, and it seems to provide a more natural and direct approach to problem solving.

B. IBM/370 TO INTERDATA 8/32 ENVIRONMENT

In addition to having to write, debug, and execute programs under a different machine, the four areas that were greatly influenced by the computing environment conversion are as follows: interacting with the SOLO operating system, performing I/O, providing the computer-to-terminal interface, and writing the user instructions.

Learning how to use SOLO operating system services was one of the most time consuming tasks of the entire project. Designed and implemented by Per Brinch Hansen, SOLO is a single user multiprocessing system written in CPASCAL which allows users to edit, compile, and execute Pascal programs. SOLO runs as a task under the OS-32/MT operating system of the INTERDATA 8/32 at KSU; such that, each Pascal user has access to his own copy of SOLO while active on the INTERDATA 8/32. In terms of the computing environment conversion of the CSGP, the most important aspects of SOLO are the standard prefix and the interprocess communication between process partitions.

The standard prefix provides the interface between an SPASCAL program and the SOLO operating system. After its compilation, an SPASCAL program is stored on disk and executed by a user command from the console. The program accesses SOLO services by means of procedures implemented within the operating system. These procedures and their parameter types are declared in a prefix to a user's program, which allows type checking of calls to the operating system to occur at compile time.

The input, job, and output processes comprise the three process partitions of SOLO into which SPASCAL programs may be loaded and executed. A job process program controls the data flow between the three process partitions by controlling the loading of input and output programs through communication with the SOLO IO program. The IO program is initially loaded into both the input and output partitions, and it waits for an argument from the job process specifying which program to load and execute next. In the case of a CSGP program, which executes in the job process partition, it uses the WRITEARG prefix procedure to inform the output partition IO program to load the Computek driver program or the printer program. The IO program calls the requested program and then makes available a program completion status argument that the CSGP program can retrieve and examine by calling the READARG prefix procedure.

While a program is executing in the input or output process partition, the data transmission between it and the job process program can be performed character-at-a-time or page-at-a-time. The CSGP utilizes the page transmission method for sending data to the Computek driver or printer program. A page consists of any data type consisting of the equivalent of 512 bytes of storage. The job program sends a page by calling the prefix procedure WRITEPAGE and using the page variable as one of its arguments. The output program receives the page by calling READPAGE. The formal parameters of WRITEPAGE and READPAGE also include a boolean type variable which is false to specify that more pages will be sent and is true to denote that the current page contains no valid data and the transmission is at an end. The job program can then retrieve the output program completion argument via READARG.

Providing the computer-to-terminal interface was another major phase of changing the CSGP's machine environment. The FORTRAN package contains three modules that perform this interface: COMPIL, CMPUTK, and TTYIO. COMPIL is a FORTRAN subroutine that transforms the Pseudo Display File into Computek compatible control data. It passes PDF opcodes and operands to an assembler language subroutine, CMPUTK, which performs the actual translation of the data, and then COMPIL passes this data to another assembler language subroutine, TTYIO, which transmits the control data to the Computek terminal. The SPASCAL COMPIL procedure was written to perform the same function as its FORTRAN counterpart, the Computek driver program (CSGPCOMPUTEK) conducting the CMPUTK and TTYIO functions. However, the SPASCAL COMPIL procedure is more complex in that it has to translate the PDF data into an extra intermediate form compatible with CSGPCOMPUTEK and perform the necessary interaction with SOLO to send this intermediate data to the output process partition.

Another matter involving the conversion of the computer-to-terminal interface software was adapting the COMPUTEK program, obtained from M. Neal, for use with the CSGP. The adapted version, named CSGPCOMPUTEK (listing provided in Appendix B), contains minor changes to COMPUTEK to provide the capability of turning on and off the debugging facility

COMPUTEK provides and to provide a more space efficient method of receiving the text operand of HTEXT/VTEXT commands.

The debugging facility consists of providing, as output to the user's console, the hexadecimal representation of the pages of ASCII characters sent to the Computek terminal. To allow the user the option of generating this output, the DEBUG_CMPTK_DRVR command was added to the CSGP and given the opcode of 8. To implement the command in CSGPCOMPUTEK, the integer constant DEBUG and boolean variable DEBUG_FLAG were added. DEBUG_FLAG is set to false during the initialization phase of the program. In the procedure PROCESS_PAGE, a case for the constant DEBUG was added to the "case" statement which processes the various PDF commands based on their opcodes. When PROCESS_PAGE recognizes a DEBUG_CMPTK_DRVR instruction via the "case" statement, the debugging flag is set to true. In the SEND procedure, a test is made on DEBUG_FLAG before the procedure PRINTABS is called to output the Computek display data.

The method employed by COMPUTEK to retrieve HTEXT/VTEXT text characters is to force the sending program in the job process partition to pad the text characters, so that 132 characters are sent with every HTEXT/VTEXT command. Thus when COMPUTEK recognizes a text instruction, it always retrieves the following 66 integer entries (packed form of 132 characters transmitted by the job process program) of DISPLAY_PAGE and unpacks the characters into the 132 element array TEXT_LINE. The procedure PROCESS_TEXT then removes the characters from TEXT_LINE and enters them into the character page SEND_PAGE, until the EM character is found, for transmission to the Computek terminal. The objective in CSGPCOMPUTEK is to allow COMPIL to send only the text characters plus one or two EM characters to CSGPCOMPUTEK and have it retrieve the characters from DISPLAY_PAGE until an EM character is encountered. To implement this scheme, the COMPUTEK procedure GET_LINE was replaced with a new procedure GET_CHARPAIR. Called from the PROCESS_TEXT procedure, GET_CHARPAIR removes and unpacks from DISPLAY_PAGE only two characters at a time. These characters are entered into TEXT_LINE by PROCESS_TEXT until the EM character is found. Then the same code as in COMPUTEK fetches the characters from TEXT_LINE into SEND_PAGE again until the EM character is spotted. The declaration of TEXT_LINE (a global variable) was changed from an array of 132 characters to an array of 44 characters, which will hold 42 HTEXT characters or 24 VTEXT characters plus two EM characters.

Allowing the user to output the contents of the PDF and providing the capability to output error messages also involved the change of machine environment. In the large computer implementation of the CSGP, the operating system perform much of the I/O control functions; such that, a relatively simple FORTRAN "print" or "write" statement is all that is needed to output data to an output device. However when providing output capabilities under SOLO, the programmer must interact with the operating system to a much greater extent. The WRITEARG prefix procedure must first be called to load the appropriate output program in the output process partition. If the output data is of non-character type, it must then be converted into characters by the programmer. Output characters are then loaded into a page variable and sent page-by-page, by call to WRITEPAGE, to the output program for processing. While not terribly difficult to code, the PDF/error output routines became a much larger part of the SPASCAL CSGP than of the FORTRAN CSGP in terms of the number of lines of code.

Although not a particularly obscure impact of the new computing environment, establishing operating instructions for the SPASCAL CSGP was not a minor task. For the most part, the explanations of the CSGP commands and their usage were taken from the user's guide for the FORTRAN package [And75]. However, the instructions by which users enter on the INTERDATA 8/32 system and use the CSGP in that environment had to be completely rewritten. This was more a time consuming than technical matter.

<u>REFERENCES</u>

And75	Anderson, G. User's Guide for the Computer Science Graphics Package Running Under VM/370: CMS, KSU Department of Computer Science, 1975.
Com72	Computek Inc., 300 SERIES User's Manual, 009-00021, September 1975.
Han77a	Hankley, W. and Rawlinson, J. Sequential PASCAL Supplement for FORTRAN Programmers, Technical Report CS 76-18, KSU Department of Computer Science, January 1977.
Han77b	Brinch Hansen, P. <u>The Architecture of Concurrent Programs</u>, Prentice-Hall, Englewood Cliffs, N.J., 1977.
Nea77	Neal D. and North B. SOLO Tutorials, Technical Report CS 77-20, KSU Department of Computer Science, October 1977.
Nea78	Neal M. Design and Implementation of a Portable Interactive Graphics Language Interpreter, Master's Report, KSU Department of Computer Science, 1978.
New73	Newman, W. and Sproull, R. <u>Principles of Interactive Computer Graphics</u>, McGraw-Hill Book Co., 1973.

APPENDIX A

CSGP (SPASCAL) CODE

```

(NUMBER)

"PER BRINCH HANSEN                      *   AS MODIFIED FOR THE INTERDATA
                                        *      8/32 UNDER OS/32-MT AT
 INFORMATION SCIENCE                    *   DEPARTMENT OF COMPUTER SCIENCE
 CALIFORNIA INSTITUTE OF TECHNOLOGY     *      KANSAS STATE UNIVERSITY
                                        *
 UTILITY PROGRAMS FOR                   *
 THE SOLO SYSTEM                        *
                                        *
 18 MAY 1975                            *               1 DEC 1976"


"###########
#  PREFIX  #
###########"


CONST NL = '(:10:)';    FF = '(:12:)';    CR = '(:13:)';    EM = '(:25:)';

CONST PAGELENGTH = 512;
TYPE PAGE = ARRAY (.1..PAGELENGTH.) OF CHAR;

CONST LINELENGTH = 132;
TYPE LINE = ARRAY (.1..LINELENGTH.) OF CHAR;

CONST IDLENGTH = 12;
TYPE IDENTIFIER = ARRAY (.1..IDLENGTH.) OF CHAR;

TYPE FILE = 1..2;

TYPE FILEKIND = (EMPTY, SCRATCH, ASCII, SEQCODE, CONCODE);

TYPE FILEATTR = RECORD
                    KIND: FILEKIND;
                    ADDR: INTEGER;
                    PROTECTED: BOOLEAN;
                    NOTUSED: ARRAY (.1..5.) OF INTEGER
                  END;

TYPE IODEVICE =
  (TYPEDEVICE, DISKDEVICE, TAPEDEVICE, PRINTDEVICE, CARDDEVICE);

TYPE IOOPERATION = (INPUT, OUTPUT, MOVE, CONTROL);

TYPE IOARG = (WRITEEOF, REWIND, UPSPACE, BACKSPACE);

TYPE IORESULT =
  (COMPLETE, INTERVENTION, TRANSMISSION, FAILURE,
   ENDFILE, ENDMEDIUM, STARTMEDIUM);

TYPE IOPARAM = RECORD
                   OPERATION: IOOPERATION;
                   STATUS: IORESULT;
                   ARG: IOARG
                 END;

TYPE TASKKIND = (INPUTTASK, JOBTASK, OUTPUTTASK);

TYPE ARGTAG =
  (NILTYPE, BOOLTYPE, INTTYPE, IDTYPE, PTRTYPE);

TYPE POINTER = @BOOLEAN;

TYPE ARGTYPE = RECORD
                 CASE TAG: ARGTAG OF
                   NILTYPE, BOOLTYPE: (BOOL: BOOLEAN);
                   INTTYPE: (INT: INTEGER);
                   IDTYPE: (ID: IDENTIFIER);
                   PTRTYPE: (PTR: POINTER)
               END;

CONST MAXARG = 10;
TYPE ARGLIST = ARRAY (.1..MAXARG.) OF ARGTYPE;

TYPE ARGSEQ = (INP, OUT);

TYPE PROGRESULT =
  (TERMINATED, OVERFLOW, POINTERERROR, RANGEERROR, VARIANTERROR,
   HEAPLIMIT, STACKLIMIT, CODELIMIT, TIMELIMIT, CALLERROR);

PROCEDURE READ(VAR C: CHAR);
PROCEDURE WRITE(C: CHAR);

PROCEDURE OPEN(F: FILE; ID: IDENTIFIER; VAR FOUND: BOOLEAN);
PROCEDURE CLOSE(F: FILE);
PROCEDURE GET(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);
PROCEDURE PUT(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);
FUNCTION LENGTH(F: FILE): INTEGER;

PROCEDURE MARK(VAR TOP: INTEGER);
PROCEDURE RELEASE(TOP: INTEGER);

PROCEDURE IDENTIFY(HEADER: LINE);
PROCEDURE ACCEPT(VAR C: CHAR);
PROCEDURE DISPLAY(C: CHAR);

PROCEDURE READPAGE(VAR BLOCK: UNIV PAGE; VAR EOF: BOOLEAN);
PROCEDURE WRITEPAGE(BLOCK: UNIV PAGE; EOF: BOOLEAN);
PROCEDURE READLINE(VAR TEXT: UNIV LINE);
PROCEDURE WRITELINE(TEXT: UNIV LINE);
PROCEDURE READARG(S: ARGSEQ; VAR ARG: ARGTYPE);
PROCEDURE WRITEARG(S: ARGSEQ; ARG: ARGTYPE);

PROCEDURE LOOKUP(ID: IDENTIFIER; VAR ATTR: FILEATTR; VAR FOUND: BOOLEAN);

PROCEDURE IOTRANSFER
  (DEVICE: IODEVICE; VAR PARAM: IOPARAM; VAR BLOCK: UNIV PAGE);

PROCEDURE IOMOVE(DEVICE: IODEVICE; VAR PARAM: IOPARAM);

FUNCTION TASK: TASKKIND;

PROCEDURE RUN(ID: IDENTIFIER; VAR PARAM: ARGLIST;
              VAR LINE: INTEGER; VAR RESULT: PROGRESULT);


PROGRAM P(VAR PARAM: ARGLIST);
```

```


"*******************************************************"
"*  CSGP CONSTANT, TYPE, AND VARIABLE DECLARATIONS  *"
"*******************************************************"

  CONST
    EMOD = 0;
    WMOD = 1;
    MOV  = 2;
    VEC  = 3;
    HTXT = 4;
    VTXT = 5;
    CLR  = 6;
    SPDF = 7;
    DBUG = 8;
    CTXT = 9;
    MAX_PDFTEXT_CHARS = 22;
    MAX_PDF_ELEMS = 600;
    ERR1 = 0;
    ERR2 = 1;
    ERR3 = 2;
    ERR4 = 3;
    ERR5 = 4;
    BLANK = '(:32:)';
    NUL = '(:0:)';
    SIN = 0;
    COS = 1;

  TYPE
    STRING6  = ARRAY(.1..6.)  OF CHAR;
    STRING22 = ARRAY(.1..22.) OF CHAR;
    STRING24 = ARRAY(.1..24.) OF CHAR;
    STRING28 = ARRAY(.1..28.) OF CHAR;
    STRING32 = ARRAY(.1..32.) OF CHAR;
    STRING42 = ARRAY(.1..42.) OF CHAR;
    STRING44 = ARRAY(.1..44.) OF CHAR;
    STRING72 = ARRAY(.1..72.) OF CHAR;
    PACKED_REAL = ARRAY(.1..4.) OF INTEGER;
    PACKED_TEXT = ARRAY(.1..22.) OF INTEGER;
    NULL_FIELD_TYPE = BOOLEAN;
    OP_TYPE = (PDFLINE, PDFTEXT, PDFMODE);
    PDF_ELEM = RECORD
                 OPCODE: INTEGER;
                 CASE TAG: OP_TYPE OF
                   PDFLINE: (X_COOR, Y_COOR, Z_COOR: REAL);
                   PDFTEXT: (TEXT_LEN: INTEGER;
                             TEXT_STR: STRING22);
                   PDFMODE: (NULL_FIELD: NULL_FIELD_TYPE)
               END; "RECORD AND CASE"
    PDF_TYPE = ARRAY(.1..MAX_PDF_ELEMS.) OF PDF_ELEM;
    MATRIX_TYPE = ARRAY(.1..4,1..4.) OF REAL;

  VAR
    PDF: PDF_TYPE;
    INDEX: INTEGER;
    X_SCALE: REAL;
    Y_SCALE: REAL;
    ERR_FLAG1: BOOLEAN;
    FILE_INDICATOR: ARGTYPE;
```

```
        OUTCOMPLETION: ARGTYPE;
        DISPPAGE: ARRAY(.1..256.) OF INTEGER;
        PRNTPAGE: ARRAY(.1..512.) OF CHAR;
        DPAGE_INDEX: INTEGER;
        PPAGE_INDEX: INTEGER;
        DIMENSION: INTEGER;
        T_MATRIX: MATRIX_TYPE;

"***********************************"
"*  CSGP PROCEDURE DECLARATIONS  *"
"***********************************"

"***************************DISPSTRNG*******************************************"
PROCEDURE DISPSTRNG(OUT_STRNG: STRING72);
"PURPOSE: DISPLAYS A CHARACTER STRING AT THE USER CONSOLE. "
"GLOBAL VARIABLES REFERENCED: NONE.                         "
"CALLING MODULES: DUMPPDF_CONSOLE/ERROR.                    "
    VAR CNTR_IND: INTEGER;
        CAR: CHAR;
    BEGIN
      CNTR_IND := 1;
      CAR := OUT_STRNG(.1.);
      WHILE CAR <> NUL DO
        BEGIN
          DISPLAY(CAR);
          CNTR_IND := SUCC(CNTR_IND);
          CAR := OUT_STRNG(.CNTR_IND.);
        END; "WHILE"
    END; "PROC DISPSTRNG"
"****************************ERROR*********************************************"
PROCEDURE ERROR(ERRCODE: INTEGER);
"PURPOSE: DISPLAYS ERROR MESSAGES ON USER'S CONSOLE.               "
"GLOBAL VARIABLES REFERENCED: ERR_FLAG1.                           "
"CALLING MODULES: EMODE/WMODE/CLEAR/SEND_PDF/MOVE/VECTOR/HTEXT/"
"                  VTEXT/PUTTEXTINPDF/COMPIL/DEBUG_CMPTK_DRVR/"
"                  PUT_DISPPAGE/PUT_PRNTPAGE.                  "
    BEGIN
      CASE ERRCODE OF
        ERR1: BEGIN
                ERR_FLAG1 := TRUE;
                DISPSTRNG('***CSGP - INDEX > PDF UPPER BOUND(:0:)');
              END; "ERR1 CASE"
        ERR2: DISPSTRNG('***COMPIL - ARGUMENT >= INDEX(:0:)');
        ERR3: DISPSTRNG('***COMPIL - INDEX REACHED BEFORE SEND_PDF(:0:)');
        ERR4: DISPSTRNG('***PUT_DISPPAGE -  ABNORMAL OUTPUT COMPLETION(:0:)');
        ERR5: DISPSTRNG('***PUT_PRNTPAGE -  ABNORMAL OUTPUT COMPLETION(:0:)')
      END; "CASE"
      DISPLAY(NL);
    END; "PROC ERROR"
"****************************START*********************************************"
PROCEDURE START;
"PURPOSE: INITIALIZES THE PDF INDEX TO 1, THE INDEX OVERFLOW  "
"          ERROR FLAG TO FALSE, AND THE DEFAULT VALUES OF     "
"          X_SCALE AND Y_SCALE TO 1.                          "
"GLOBAL VARIABLES REFERENCED: INDEX/ERR_FLAG1/X_SCALE/Y_SCALE."
    BEGIN
      ERR_FLAG1 := FALSE;
      X_SCALE := 1.0;
      Y_SCALE := 1.0;
      INDEX := 1;
```

```
   END; "PROC START"
"**************************SET_INDEX********************************************"
PROCEDURE SET_INDEX(IND: INTEGER);
"PURPOSE: CHANGES THE VALUE OF THE PDF INDEX."
"GLOBAL VARIABLES REFERENCED: INDEX.         "
"CALLING MODULES: USER PROGRAM.              "
   BEGIN
     IF ERR_FLAG1
       THEN ERR_FLAG1 := FALSE;
     INDEX := IND;
   END; "PROC SET_INDEX"
"**************************VAL_INDEX********************************************"
FUNCTION VAL_INDEX: INTEGER;
"PURPOSE: ACCESSES CURRENT VALUE OF THE PDF INDEX."
"GLOBAL VARIABLES REFERENCED: INDEX.              "
"CALLING MODULES: USER PROGRAM.                   "
   BEGIN
     VAL_INDEX := INDEX;
   END; "FUNC VAL_INDEX"
"**************************SET_SCALE******************************************"
PROCEDURE SET_SCALE(PTSPERUNIT_X, PTSPERUNIT_Y: REAL);
"PURPOSE: SETS THE VALUE OF X_SCALE AND Y_SCALE TO THE "
"            POINTS PER UNIT EACH X AND Y PDF COORDINATE"
"            WILL BE SCALED TO BEFORE BEING SENT TO THE "
"            COMPUTEK DRIVER.                           "
"GLOBAL VARIABLES REFERENCED: X_SCALE/Y_SCALE.          "
"CALLING MODULES: USER PROGRAM.                         "
   BEGIN
     X_SCALE := PTSPERUNIT_X;
     Y_SCALE := PTSPERUNIT_Y;
   END; "PROC SET_SCALE"
"**************************DEBUG_CMPTK_DRVR***********************************"
PROCEDURE DEBUG_CMPTK_DRVR;
"PURPOSE: CREATES A COMMAND IN THE PDF TO TURN ON A DEBUG      "
"            OPERATION IN THE COMPUTEK DRIVER PROGRAM THAT     "
"            DISPLAYS THE HEXADECIMAL REPRESENTATION OF THE    "
"            ASCII CHARACTERS SENT TO THE COMPUTEK TERMINAL.   "
"GLOBAL VARIABLES REFERENCED: PDF/INDEX.                       "
"CALLING MODULES: USER PROGRAM.                                "
   BEGIN
     IF INDEX > MAX_PDF_ELEMS
       THEN BEGIN
              IF NOT ERR_FLAG1
                THEN ERROR(ERR1);
              INDEX := MAX_PDF_ELEMS;
            END; "IF INDEX > MAX_PDF_ELEMS"
     PDF(.INDEX.).OPCODE := DBUG;
     PDF(.INDEX.).TAG := PDFMODE;
     INDEX := SUCC(INDEX);
   END; "PROC DEBUG_CMPTK_DRVR"
"**************************EMODE**********************************************"
PROCEDURE EMODE;
"PURPOSE: CREAT S A COMMAND IN THE PDF TO PLACE   "
"            THE COMPUTEK TERMINAL IN ERASE STATUS."
"GLOBAL VARIABLES REFERENCED: PDF/INDEX.          "
"CALLING MODULES: USER PROGRAM.                   "
   BEGIN
     IF INDEX > MAX_PDF_ELEMS
       THEN BEGIN
              IF NOT ERR_FLAG1
```

```
                THEN ERROR(ERR1);
              INDEX := MAX_PDF_ELEMS;
            END; "IF INDEX > MAX_PDF_ELEMS"
     PDF(.INDEX.).OPCODE := EMOD;
     PDF(.INDEX.).TAG := PDFMODE;
     INDEX := SUCC(INDEX);
   END; "PROC EMODE"
"***************************WMODE*********************************************"
 PROCEDURE WMODE;
 "PURPOSE: CREATES A COMMAND IN THE PDF TO PLACE    "
 "            THE COMPUTEK TERMINAL IN WRITE STATUS."
 "GLOBAL VARIABLES REFERENCED: PDF/INDEX,           "
 "CALLING MODULES: USER PROGRAM.                    "
   BEGIN
     IF INDEX > MAX_PDF_ELEMS
       THEN BEGIN
              IF NOT ERR_FLAG1
                THEN ERROR(ERR1);
              INDEX := MAX_PDF_ELEMS;
            END; "IF INDEX > MAX_PDF_ELEMS"
     PDF(.INDEX.).OPCODE := WMOD;
     PDF(.INDEX.).TAG := PDFMODE;
     INDEX := SUCC(INDEX);
   END; "PROC WMODE"
"***************************CLEAR*********************************************"
 PROCEDURE CLEAR;
 "PURPOSE: CREATES A COMMAND IN THE PDF TO CLEAR THE SCREEN"
 "            AND POSITION THE CURSOR AT HOME.             "
 "GLOBAL VARIABLES REFERENCED: PDF/INDEX,                  "
 "CALLING MODULES: USER PROGRAM.                           "
   BEGIN
     IF INDEX > MAX_PDF_ELEMS
       THEN BEGIN
              IF NOT ERR_FLAG1
                THEN ERROR(ERR1);
              INDEX := MAX_PDF_ELEMS;
            END; "IF INDEX > MAX_PDF_ELEMS"
     PDF(.INDEX.).OPCODE := CLR;
     PDF(.INDEX.).TAG := PDFMODE;
     INDEX := SUCC(INDEX);
   END; "PROC CLEAR"
"***************************SEND_PDF******************************************"
 PROCEDURE SEND_PDF;
 "PURPOSE: CREATES AN ENTRY IN THE PDF TO MARK THE END"
 "            OF AN IMAGE.                            "
 "GLOBAL VARIABLES REFERENCED: PDF/INDEX,             "
 "CALLING MODULES: USER PROGRAM.                      "
   BEGIN
     IF INDEX > MAX_PDF_ELEMS
       THEN BEGIN
              IF NOT ERR_FLAG1
                THEN ERROR(ERR1);
              INDEX := MAX_PDF_ELEMS;
            END; "IF INDEX > MAX_PDF_ELEMS"
     PDF(.INDEX.).OPCODE := SPDF;
     PDF(.INDEX.).TAG := PDFMODE;
     INDEX := SUCC(INDEX);
   END; "PROC SEND_PDF"
"***************************MOVE**********************************************"
 PROCEDURE MOVE(X, Y, Z: REAL);
```

```
  "PURPOSE: CREATES AN ENTRY IN THE PDF FOR"
  "              A MOVE INSTRUCTION.        "
  "GLOBAL VARIABLES REFERENCED: PDF/INDEX. "
  "CALLING MODULES: USER PROGRAM,          "
    BEGIN
      IF INDEX > MAX_PDF_ELEMS
        THEN BEGIN
               IF NOT ERR_FLAG1
                 THEN ERROR(ERR1);
               INDEX := MAX_PDF_ELEMS;
             END; "IF INDEX > MAX_PDF_ELEMS"
      WITH PDF(.INDEX.) DO
        BEGIN
          OPCODE := MOV;
          TAG := PDFLINE;
          X_COOR := X;
          Y_COOR := Y;
          Z_COOR := Z;
        END; "WITH"
      INDEX := SUCC(INDEX);
    END; "PROC MOVE"
"****************************VECTOR*********************************************"
  PROCEDURE VECTOR(X, Y, Z: REAL);
  "PURPOSE: INSERTS AN ENTRY INTO THE PDF FOR"
  "              A VECTOR COMMAND.            "
  "GLOBAL VARIABLES REFERENCED: PDF/INDEX,    "
  "CALLING MODULES: USER PROGRAM,             "
    BEGIN
      IF INDEX > MAX_PDF_ELEMS
        THEN BEGIN
               IF NOT ERR_FLAG1
                 THEN ERROR(ERR1);
               INDEX := MAX_PDF_ELEMS;
             END; "IF INDEX > MAX_PDF_ELEMS"
      WITH PDF(.INDEX.) DO
        BEGIN
          OPCODE := VEC;
          TAG := PDFLINE;
          X_COOR := X;
          Y_COOR := Y;
          Z_COOR := Z;
        END; "WITH"
      INDEX := SUCC(INDEX);
    END; "PROC VECTOR"
"***************************PUTTEXTINPDF***************************************"
  PROCEDURE PUTTEXTINPDF(STRNG_LEN: INTEGER; STRNG: STRING42);
  "PURPOSE: ENTERS HTEXT AND VTEXT TEXT INTO THE PDF          "
  "              AND CREATES CTEXT PDF ENTRIES WHEN REQUIRED."
  "GLOBAL VARIABLES REFERENCED: PDF/INDEX,                    "
  "CALLING MODULES: HTEXT/VTEXT.                              "
    VAR CHARINDEX1, CHARINDEX2, CTEXT_ENTRIES, LOOP_CNTR: INTEGER;
    BEGIN
      CHARINDEX1 := 0;
      REPEAT
        CHARINDEX1 := SUCC(CHARINDEX1);
        PDF(.INDEX.).TEXT_STR(.CHARINDEX1.) := STRNG(.CHARINDEX1.);
      UNTIL (CHARINDEX1 = STRNG_LEN) OR (CHARINDEX1 = MAX_PDFTEXT_CHARS);
      PDF(.INDEX.).TEXT_LEN := CHARINDEX1;
      IF STRNG_LEN MOD MAX_PDFTEXT_CHARS = 0
        THEN CTEXT_ENTRIES := (STRNG_LEN DIV MAX_PDFTEXT_CHARS) - 1
```

```
   ELSE CTEXT_ENTRIES := STRNG_LEN DIV MAX_PDFTEXT_CHARS;
  FOR LOOP_CNTR := 1 TO CTEXT_ENTRIES DO
   BEGIN
    CHARINDEX2 := 0;
    INDEX := SUCC(INDEX);
    IF INDEX > MAX_PDF_ELEMS
     THEN BEGIN
         IF NOT ERR_FLAG1
          THEN ERROR(ERR1);
         INDEX := MAX_PDF_ELEMS;
        END; "IF INDEX > MAX_PDF_ELEMS"
    PDF(.INDEX.).OPCODE := CTXT;
    PDF(.INDEX.).TAG := PDFTEXT;
    REPEAT
     CHARINDEX1 := SUCC(CHARINDEX1);
     CHARINDEX2 := SUCC(CHARINDEX2);
     PDF(.INDEX.).TEXT_STR(.CHARINDEX2.) := STRNG(.CHARINDEX1.);
    UNTIL (CHARINDEX1 = STRNG_LEN) OR (CHARINDEX2 = MAX_PDFTEXT_CHARS);
    PDF(.INDEX.).TEXT_LEN := CHARINDEX2;
   END; "FOR"
 END; "PROC PUTTEXTINPDF"
"***************************HTEXT*******************************************"
PROCEDURE HTEXT(STRNG_LEN: INTEGER; STRNG: STRING42);
"PURPOSE: INSERTS AN INSTRUCTION INTO THE PDF FOR"
"              DISPLAY OF HORIZONTAL TEXT.          "
"GLOBAL VARIABLES REFERENCED: PDF/INDEX.           "
"CALLING MODULES: USER PROGRAM.                    "
 BEGIN
  IF INDEX > MAX_PDF_ELEMS
   THEN BEGIN
       IF NOT ERR_FLAG1
        THEN ERROR(ERR1);
       INDEX := MAX_PDF_ELEMS;
      END; "IF INDEX > MAX_PDF_ELEMS"
  PDF(.INDEX.).OPCODE := HTXT;
  PDF(.INDEX.).TAG := PDFTEXT;
  PUTTEXTINPDF(STRNG_LEN, STRNG);
  INDEX := SUCC(INDEX);
 END; "PROC HTEXT"
"***************************VTEXT*******************************************"
PROCEDURE VTEXT(STRNG_LEN: INTEGER; STRNG: STRING24);
"PURPOSE: INSERTS AN INSTRUCTION INTO THE PDF FOR"
"              DISPLAY OF VERTICAL TEXT.            "
"GLOBAL VARIABLES REFERENCED: PDF/INDEX.           "
"CALLING MODULES: USER PROGRAM.                    "
 BEGIN
  IF INDEX > MAX_PDF_ELEMS
   THEN BEGIN
       IF NOT ERR_FLAG1
        THEN ERROR(ERR1);
       INDEX := MAX_PDF_ELEMS;
      END; "IF INDEX > MAX_PDF_ELEMS"
  PDF(.INDEX.).OPCODE := VTXT;
  PDF(.INDEX.).TAG := PDFTEXT;
  PUTTEXTINPDF(STRNG_LEN, STRNG);
  INDEX := SUCC(INDEX);
 END; "PROC VTEXT"
"***************************LOAD_CMPTK_DRVR*****************************"
PROCEDURE LOAD_CMPTK_DRVR;
"PURPOSE: LOADS THE COMPUTEK DRIVER INTO THE "
```

```
"             SOLO OUTPUT PROCESS PARTITION.   "
"GLOBAL VARIABLES REFERENCED: FILE_INDICATOR."
"CALLING MODULES: COMPIL.                      "
  BEGIN
    FILE_INDICATOR.TAG := IDTYPE;
    FILE_INDICATOR.ID := 'CSGPCMPTKOBJ';
    WRITEARG(OUT, FILE_INDICATOR);
  END; "PROC LOAD_CMPTK_DRVR"
"****************************GET_TEXT*******************************************"
PROCEDURE GET_TEXT(VAR PDFIND: INTEGER; VAR TEXT_STRNG: STRING44);
"PURPOSE: RETRIEVES FROM THE PDF THE TEXT ASSOCIATED WITH THE"
"             HTEXT/VTEXT COMMAND STARTING AT INDEX PDFIND.    "
"GLOBAL VARIABLES REFERENCED: PDF,                             "
"CALLING MODULES: COMPIL.                                      "
  VAR PDFTEXTIND, TEXTSTRIND1, TEXTSTRIND2: INTEGER;
  BEGIN
    TEXTSTRIND1 := 1;
    REPEAT
      FOR PDFTEXTIND := 1 TO PDF(.PDFIND.).TEXT_LEN DO
        BEGIN
          TEXT_STRNG(.TEXTSTRIND1.) := PDF(.PDFIND.).TEXT_STR(.PDFTEXTIND.);
          TEXTSTRIND1 := SUCC(TEXTSTRIND1);
        END; "FOR PDFTEXTIND"
      PDFIND := SUCC(PDFIND);
    UNTIL PDF(.PDFIND.).OPCODE <> CTXT;
    PDFIND := PRED(PDFIND);
    TEXT_STRNG(.TEXTSTRIND1.) := EM;
    TEXT_STRNG(.SUCC(TEXTSTRIND1).) := EM;
  END; "PROC GET_TEXT"
"****************************PUT_DISPPAGE***************************************"
PROCEDURE PUT_DISPPAGE(INTVAL: INTEGER);
"PURPOSE: ENTERS PDF OP CODES AND TEXT IN INTEGER FORM INTO     "
"             DISPPAGE BEFORE ITS TRANSFER TO THE COMPUTEK DRIVER."
"GLOBAL VARIABLES REFERENCED: DISPPAGE/DPAGE_INDEX.             "
"CALLING MODULES: COMPIL/PACK_REAL/PACK_CHAR.                   "
  BEGIN
    DISPPAGE(.DPAGE_INDEX.) := INTVAL;
    DPAGE_INDEX := SUCC(DPAGE_INDEX);
    IF (DPAGE_INDEX > 256) OR (INTVAL = SPDF)
      THEN BEGIN
             WRITEPAGE(DISPPAGE, FALSE);
             IF INTVAL <> SPDF
               THEN DPAGE_INDEX := 1
               ELSE BEGIN
                      WRITEPAGE(DISPPAGE,TRUE);
                      READARG(OUT,OUTCOMPLETION);
                      IF NOT OUTCOMPLETION.BOOL
                        THEN ERROR(ERR4);
                    END; "IF INTVAL <> SPDF"
           END; "IF (DPAGE_INDEX > 256) OR..."
  END; "PROC PUT_DISPPAGE"
"****************************PACK_REAL******************************************"
PROCEDURE PACK_REAL(XINT, YINT: UNIV PACKED_REAL);
"PURPOSE: PACKS THE X,Y REAL COORDINATES OF A MOVE/VEC         "
"             INSTRUCTION INTO 8 INTEGER ELEMENTS OF DISPPAGE."
"GLOBAL VARIABLES REFERENCED: NONE.                            "
"CALLING MODULES: COMPIL.                                      "
  VAR CNTR_IND: INTEGER;
  BEGIN
    FOR CNTR_IND := 1 TO 4 DO
```

```
        PUT_DISPPAGE(XINT(.CNTR_IND,));
      FOR CNTR_IND := 1 TO 4 DO
        PUT_DISPPAGE(YINT(.CNTR_IND,));
    END; "PROC PACK_REAL"
"***************************PACK_CHAR*******************************************"
  PROCEDURE PACK_CHAR(PDFIND: INTEGER; TEXTINT: UNIV PACKED_TEXT);
  "PURPOSE: PACKS HTEXT/VTEXT CHARACTER STRING INTO INTEGER"
  "           ELEMENTS OF DISPPAGE.                          "
  "GLOBAL VARIABL S REFERENCED: NONE.                        "
  "CALLING MODULES: COMPIL.                                  "
    VAR CNTR_IND, LOOP_LIMIT: INTEGER;
    BEGIN
      CNTR_IND := PDFIND;
      LOOP_LIMIT := PDF(.PDFIND.).TEXT_LEN;
      WHILE (PDF(.CNTR_IND.).OPCODE = CTXT) DO
        BEGIN
          CNTR_IND := PRED(CNTR_IND);
          LOOP_LIMIT := LOOP_LIMIT + PDF(.CNTR_IND.).TEXT_LEN;
        END; "WHILE"
      LOOP_LIMIT := (LOOP_LIMIT DIV 2) + 1;
      FOR CNTR_IND := 1 TO LOOP_LIMIT DO
        PUT_DISPPAGE(TEXTINT(.CNTR_IND.));
    END; "PROC PACK_CHAR"
"***************************COMPIL**********************************************"
  PROCEDURE COMPIL(FIRSTPDFINDEX: INTEGER);
  "PURPOSE: ENCODES PDF COMMANDS (STARTING AT FIRSTPDFINDEX    "
  "           AND ENDING WHEN A SEND-PDF COMMAND IS FOUND) INTO"
  "           A FORM ACCEPTABLE TO THE COMPUTEK DRIVER PROGRAM "
  "           AND THEN TRANSMITS THE ENCODED COMMANDS TO THE   "
  "           COMPUTEK TERMINAL DRIVER FOR DISPLAY.            "
  "GLOBAL VARIABLES REFERENCED: PDF/DPAGE_INDEX.               "
  "CALLING MODULES: USER PROGRAM.                              "
    VAR PDFINDEX, PDFOPCODE: INTEGER;
        TEXT_STRNG: STRING44;
        X_REAL, Y_REAL: REAL;
    BEGIN
      DPAGE_INDEX := 1;
      OUTCOMPLETION.BOOL := FALSE;
      IF FIRSTPDFINDEX >= INDEX
        THEN BEGIN
               ERROR(ERR2);
               PDFINDEX := 1;
             END "THEN"
        ELSE PDFINDEX := FIRSTPDFINDEX; "END IF FIRSTPDFINDEX..."
      LOAD_CMPTK_DRVR;
      REPEAT
        PDFOPCODE := PDF(.PDFINDEX.).OPCODE;
        CASE PDFOPCODE OF
          MOV,VEC   : BEGIN
                        PUT_DISPPAGE(PDFOPCODE);
                        X_REAL := PDF(.PDFINDEX.).X_COOR * X_SCALE;
                        Y_REAL := PDF(.PDFINDEX.).Y_COOR * Y_SCALE;
                        PACK_REAL(X_REAL, Y_REAL);
                      END; "MOV,VEC CASE"
          HTXT,VTXT : BEGIN
                        PUT_DISPPAGE(PDFOPCODE);
                        GET_TEXT(PDFINDEX, TEXT_STRNG);
                        PACK_CHAR(PDFINDEX, TEXT_STRNG);
                      END; "HTXT,VTXT CASE"
          EMOD,WMOD,
         DBUG,SPDF,
               CLR : PUT_DISPPAGE(PDFOPCODE)
       END; "CASE"
       PDFINDEX := SUCC(PDFINDEX);
     UNTIL (PDFOPCODE = SPDF) OR (PDFINDEX >= INDEX);
     IF PDFOPCODE <> SPDF
       THEN BEGIN
              ERROR(ERR3);
              PUT_DISPPAGE(SPDF);
            END; "IF PDFOPCODE <> SPDF"
   END; "PROC COMPIL"
"****************************INT_TO_STR*****************************************"
 PROCEDURE INT_TO_STR(INT: INTEGER; VAR CHAR_INT: STRING6);
 "PURPOSE: CONVERTS AN INTEGER (PDF INDEX)         "
 "           INTO CHARACTER STRING FORM.           "
 "GLOBAL VARIABLES REFERENCED: NONE.               "
 "CALLING MODULES: DUMPPDF_CONSOLE/DUMPPDF_PRINTER."
    VAR INTNUM, DIGIT, PLACE, CHARINDEX: INTEGER;
   BEGIN
     INTNUM := INT;
     DIGIT := 0;
     PLACE := 100;
     CHARINDEX := 1;
     REPEAT
       INTNUM := INTNUM - PLACE;
       IF INTNUM < 0
         THEN BEGIN
                CHAR_INT(.CHARINDEX.) := CHR(DIGIT + 48);
                INTNUM := INTNUM + PLACE;
                PLACE := 10;
                DIGIT := 0;
                CHARINDEX := SUCC(CHARINDEX);
              END "THEN"
         ELSE DIGIT := SUCC(DIGIT); "END IF INTNUM < 0"
     UNTIL CHARINDEX = 3;
     CHAR_INT(.CHARINDEX.) := CHR(INTNUM + 48);
     CHAR_INT(.4.) := BLANK;
     CHAR_INT(.5.) := NUL;
   END; "PROC INT_TO_STR"
"*****************************REAL_TO_STR***************************************"
 PROCEDURE REAL_TO_STR(PDFIND: INTEGER; VAR CHAR_REAL: STRING28);
 "PURPOSE: CONVERTS X,Y,Z (REAL) COORDINATES OF A MOVE/VECTOR"
 "            PDF COMMAND INTO CHARACTER STRING FORM.         "
 "GLOBAL VARIABLES REFERENCED: PDF.                           "
 "CALLING MODULES: DUMPPDF_CONSOLE/DUMPPDF_PRINTER.           "
   VAR DIGIT, CHARINDEX, CNTR, CNTR_IND: INTEGER;
       REALNUM, PLACE: REAL;
       NONZERO_DIGIT_FLAG: BOOLEAN;
   BEGIN
     REALNUM := PDF(.PDFIND.).X_COOR; "ASSUME -10,000<REALNUM<10,000"
     DIGIT := 0;
     CHARINDEX := 1;
     FOR CNTR := 1 TO 3 DO
       BEGIN
         PLACE := 1000.0;
         NONZERO_DIGIT_FLAG := FALSE;
         IF REALNUM < 0.0
           THEN BEGIN
                  CHAR_REAL(.CHARINDEX.) := '-';
                  CHARINDEX := SUCC(CHARINDEX);
```

```
                 REALNUM := ABS(REALNUM);
               END; "IF REALNUM < 0.0"
        REPEAT
          REALNUM := REALNUM - PLACE;
          IF REALNUM <= -0.006
            THEN BEGIN
                   IF DIGIT <> 0
                     THEN NONZERO_DIGIT_FLAG := TRUE;
                   IF (NONZERO_DIGIT_FLAG = TRUE) OR (PLACE <= 1.0)
                     THEN BEGIN
                            CHAR_REAL(.CHARINDEX.) := CHR(DIGIT + 48);
                            CHARINDEX := SUCC(CHARINDEX);
                          END; "IF (NONZERO_DIGIT_FLAG = TRUE) OR..."
                   IF PLACE = 1.0
                     THEN BEGIN
                            CHAR_REAL(.CHARINDEX.) := '.';
                            CHARINDEX := SUCC(CHARINDEX);
                          END; "IF PLACE = 1.0"
                   REALNUM := REALNUM + PLACE;
                   PLACE := PLACE / 10.0;
                   DIGIT := 0;
                 END "THEN"
            ELSE DIGIT := SUCC(DIGIT); "END IF REALNUM <=..."
        UNTIL PLACE <= 0.001;
        IF CNTR <= 2
          THEN BEGIN
                 CHAR_REAL(.CHARINDEX.) := ',';
                 CHARINDEX := SUCC(CHARINDEX);
                 IF CNTR = 1
                   THEN REALNUM := PDF(.PDFIND.).Y_COOR
                   ELSE REALNUM := PDF(.PDFIND.).Z_COOR;
               END; "IF CNTR <= 2"
      END; "FOR CNTR"
    FOR CNTR_IND := CHARINDEX TO 27 DO
      CHAR_REAL(.CNTR_IND.) := BLANK;
    CHAR_REAL(.28.) := NUL;
  END; "PROC REAL_TO_STR"
"***************************GET_TEXT_OUT*****************************************"
 PROCEDURE GET_TEXT_OUT(PDFIND: INTEGER; VAR TEXT_STRNG: STRING28);
 "PURPOSE: RETRIEVES AND PREPARES TEXT FROM THE PDF FOR  "
 "            SUBSEQUENT OUTPUT TO THE CONSOLE OR PRINTER."
 "GLOBAL VARIABL S REFERENCED: PDF.                      "
 "CALLING MODULES: DUMPPDF_CONSOLE/DUMPPDF_PRINTER.      "
  VAR CHARINDEX, CNTR_IND: INTEGER;
  BEGIN
    CHARINDEX := 1;
    REPEAT
      TEXT_STRNG(.CHARINDEX.) := PDF(.PDFIND.).TEXT_STR(.CHARINDEX.);
      CHARINDEX := SUCC(CHARINDEX);
    UNTIL CHARINDEX > PDF(.PDFIND.).TEXT_LEN;
    FOR CNTR_IND := CHARINDEX TO 27 DO
      TEXT_STRNG(.CNTR_IND.) := BLANK;
    TEXT_STRNG(.28.) := NUL;
  END; "PROC GET_TEXT_OUT"
"***************************DUMPPDF_CONSOLE*************************************"
 PROCEDURE DUMPPDF_CONSOLE(STARTINDEX, ENDINDEX: INTEGER);
 "PURPOSE: DUMPS CONTENTS OF PDF TO USER'S CONSOLE UNTIL ENDINDEX "
 "            IS REACHED, THE CONSOLE IS FULL (44 COMMANDS DUMPED),"
 "            OR INDEX IS REACHED. IF ENDINDEX = 0 THE DUMP       "
 "            CONTINUES UNTIL A SEND_PDF COMMAND HAS BEEN FOUND.  "
```

```
"GLOBAL VARIABL S REFERENCED: PDF,                              "
"CALLING MODULES: USER PROGRAM,                                 "
  VAR PDFIND, PDFOPCODE: INTEGER;
      CHAR_INT: STRING6;
      CHAR_REAL, TEXTSTRNG: STRING28;
  BEGIN
    PDFIND := STARTINDEX;
    WHILE ((PDFIND <= ENDINDEX) & (ENDINDEX <> 0) OR
           (PDF(.PDFIND.).OPCODE <> SPDF) & (ENDINDEX = 0)) &
          (PDFIND - STARTINDEX < 49)                       &
          (PDFIND < INDEX) DO
      BEGIN
        INT_TO_STR(PDFIND, CHAR_INT);
        DISPSTRNG(CHAR_INT);
        PDFOPCODE := PDF(.PDFIND.).OPCODE;
        CASE PDFOPCODE OF
          MOV : BEGIN
                  DISPSTRNG('MOV (:0:)(:0:)');
                  REAL_TO_STR(PDFIND, CHAR_REAL);
                  DISPSTRNG(CHAR_REAL);
                END; "MOV CASE"
          VEC : BEGIN
                  DISPSTRNG('VEC (:0:)(:0:)');
                  REAL_TO_STR(PDFIND, CHAR_REAL);
                  DISPSTRNG(CHAR_REAL);
                END; "VEC CASE"
          HTXT: BEGIN
                  DISPSTRNG('HTX (:0:)(:0:)');
                  GET_TEXT_OUT(PDFIND, TEXTSTRNG);
                  DISPSTRNG(TEXTSTRNG);
                END; "HTXT CASE"
          VTXT: BEGIN
                  DISPSTRNG('VTX (:0:)(:0:)');
                  GET_TEXT_OUT(PDFIND, TEXTSTRNG);
                  DISPSTRNG(TEXTSTRNG);
                END; "VTXT CASE"
          CTXT: BEGIN
                  DISPSTRNG('CTX (:0:)(:0:)');
                  GET_TEXT_OUT(PDFIND, TEXTSTRNG);
                  DISPSTRNG(TEXTSTRNG);
                END; "CTXT CASE"
          CLR : DISPSTRNG('CLR                                   (:0:)');
          EMOD: DISPSTRNG('EMD                                   (:0:)');
          WMOD: DISPSTRNG('WMD                                   (:0:)');
          DBUG: DISPSTRNG('DBG                                   (:0:)');
          SPDF: DISPSTRNG('SND                                   (:0:)')
        END; "CASE"
        IF ((PDFIND - STARTINDEX) + 1) MOD 2 = 0
          THEN DISPLAY(NL);
        PDFIND := SUCC(PDFIND);
      END; "WHILE"
      IF ((PDFIND - STARTINDEX) + 1) MOD 2 = 0
        THEN DISPLAY(NL);
  END; "PROC DUMPPDF_CONSOLE"
"****************************LOAD_PRINTER_PROG********************************"
PROCEDURE LOAD_PRINTER_PROG;
"PURPOSE: LOADS THE PRINTER PROGRAM INTO THE "
"            SOLO OUTPUT PROCESS PARTITION.   "
"GLOBAL VARIABLES REFERENCED: FILE_INDICATOR."
"CALLING MODULES: DUMPPDF_PRINTER,            "
```

```
   BEGIN
     FILE_INDICATOR.TAG := IDTYPE;
     FILE_INDICATOR.ID := 'PRINTER     ';
     WRITEARG(OUT, FILE_INDICATOR);
   END; "PROC LOAD_PRINTER_PROG"
"**************************PUT_PRNTPAGE**************************************"
 PROCEDURE PUT_PRNTPAGE(OUT_STRNG: STRING32);
 "PURPOSE: ENTERS A CHARACTER STRING INTO PRNTPAGE. WHEN              "
 "            PRNTPAGE IS FULL, IT IS SENT TO THE PRINTER PROGRAM."
 "GLOBAL VARIABLES REFERENCED: PRNTPAGE/PPAGE_INDEX,                  "
 "CALLING MODULES: DUMPPDF_PRINTER,                                   "
   VAR CHARINDEX: INTEGER;
   BEGIN
     CHARINDEX := 1;
     REPEAT
       PRNTPAGE(.PPAGE_INDEX.) := OUT_STRNG(.CHARINDEX.);
       IF (PPAGE_INDEX = PAGELENGTH) OR
         (OUT_STRNG(.CHARINDEX.) = EM)
         THEN BEGIN
                WRITEPAGE(PRNTPAGE, FALSE);
                IF OUT_STRNG(.CHARINDEX.) = EM
                  THEN BEGIN
                        WRITEPAGE(PRNTPAGE,TRUE);
                        READARG(OUT,OUTCOMPLETION);
                        IF NOT OUTCOMPLETION.BOOL
                          THEN ERROR(ERR5);
                      END; "IF OUT-STRNG(.CHARINDEX.) = EM"
                PPAGE_INDEX := 0;
              END; "IF (PPAGE_INDEX = PAGELENGTH) OR..."
       PPAGE_INDEX := SUCC(PPAGE_INDEX);
       CHARINDEX := SUCC(CHARINDEX);
     UNTIL OUT_STRNG(.CHARINDEX.) = NUL;
   END; "PROC PUT_PRNTPAGE"
"***************************DUMPPDF_PRINTER**********************************"
 PROCEDURE DUMPPDF_PRINTER(STARTINDEX, ENDINDEX: INTEGER);
 "PURPOSE: DUMPS CONTENTS OF PDF TO PRINTER UNTIL ENDINDEX OR INDEX"
 "            IS REACHED. IF ENDINDEX = 0, DUMP CONTINUES UNTIL      "
 "            A SEND_PDF COMMAND IS FOUND.                           "
 "GLOBAL VARIABLES REFERENCED: PDF/PPAGE_INDEX,                      "
 "CALLING MODULES: USER PROGRAM.                                     "
   VAR PDFIND, PDFOPCODE: INTEGER;
       CHAR_INT: STRING6;
       CHAR_REAL, TEXTSTRNG: STRING28;
   BEGIN
     PPAGE_INDEX := 1;
     OUTCOMPLETION.BOOL := FALSE;
     PDFIND := STARTINDEX;
     LOAD_PRINTER_PROG;
     WHILE ((PDFIND <= ENDINDEX) & (ENDINDEX <> 0) OR
            (PDF(.PDFIND.).OPCODE <> SPDF) & (ENDINDEX = 0)) &
          (PDFIND < INDEX) DO
       BEGIN
         INT_TO_STR(PDFIND, CHAR_INT);
         PUT_PRNTPAGE(CHAR_INT);
         PDFOPCODE := PDF(.PDFIND.).OPCODE;
         CASE PDFOPCODE OF
           MOV : BEGIN
                  PUT_PRNTPAGE('MOV (:0:)(:0:)');
                  REAL_TO_STR(PDFIND, CHAR_REAL);
                  PUT_PRNTPAGE(CHAR_REAL);
                END: "MOV CASE"
          VEC : BEGIN
                  PUT_PRNTPAGE('VEC (:0:)(:0:)');
                  REAL_TO_STR(PDFIND, CHAR_REAL);
                  PUT_PRNTPAGE(CHAR_REAL);
                END: "VEC CASE"
          HTXT: BEGIN
                  PUT_PRNTPAGE('HTX (:0:)(:0:)');
                  GET_TEXT_OUT(PDFIND, TEXTSTRNG);
                  PUT_PRNTPAGE(TEXTSTRNG);
                END: "HTXT CASE"
          VTXT: BEGIN
                  PUT_PRNTPAGE('VTX (:0:)(:0:)');
                  GET_TEXT_OUT(PDFIND, TEXTSTRNG);
                  PUT_PRNTPAGE(TEXTSTRNG);
                END: "VTXT CASE"
          CTXT: BEGIN
                  PUT_PRNTPAGE('CTX (:0:)(:0:)');
                  GET_TEXT_OUT(PDFIND, TEXTSTRNG);
                  PUT_PRNTPAGE(TEXTSTRNG);
                END: "CTXT CASE"
          CLR : PUT_PRNTPAGE('CLR                                (:0:)');
          EMOD: PUT_PRNTPAGE('EMD                                (:0:)');
          WMOD: PUT_PRNTPAGE('WMD                                (:0:)');
          DBUG: PUT_PRNTPAGE('DBG                                (:0:)');
          SPDF: PUT_PRNTPAGE('SND                                (:0:)')
        END; "CASE"
        PUT_PRNTPAGE('(:13:)(:10:)(:0:)(:0:)'); "(:13:)=CR,(:10:)=NL"
        IF ((PDFIND - STARTINDEX) + 1) MOD 30 = 0
          THEN PUT_PRNTPAGE('(:12:)(:0:)'); "(:12:)=FF"
        PDFIND := SUCC(PDFIND);
      END; "WHILE"
      PUT_PRNTPAGE('(:25:)(:0:)'); "(:25:)=EM"
  END; "PROC DUMPPDF_PRINTER"
"*****************************INIT_T_MATRIX***************************************"
 PROCEDURE INIT_T_MATRIX(DIM: INTEGER);
 "PURPOSE: INITIALIZES AN APPROPRIATELY SIZED IDENTITY MATRIX       "
 "            AND SETS THE GLOBAL DIMENSION VARIABLE (DIMENSION) TO"
 "            THE DIMENSION (2 OR 3) IN WHICH FOLLOWING            "
 "            TRANSFORMATIONS ARE TO TAKE PLACE.                   "
 "GLOBAL VARIABLES REFERENCED: DIMENSION/T_MATRIX.                 "
 "CALLING MODULES: USER PROGRAM.                                   "
  VAR CNTR_IND1, CNTR_IND2, MTRX_LIMIT: INTEGER;
  BEGIN
    DIMENSION := DIM;
    MTRX_LIMIT := SUCC(DIM);
    FOR CNTR_IND1 := 1 TO MTRX_LIMIT DO
      BEGIN
        FOR CNTR_IND2 := 1 TO MTRX_LIMIT DO
          T_MATRIX(.CNTR_IND1,CNTR_IND2.) := 0.0; "END FOR CNTR_IND2"
        T_MATRIX(.CNTR_IND1,CNTR_IND1.) := 1.0;
      END; "FOR CNTR_IND1"
  END; "PROC INIT_T_MATRIX"
"*****************************MATMUL**********************************************"
 PROCEDURE MATMUL(NEW_T_MATRIX: MATRIX_TYPE);
 "PURPOSE: PERFORMS CONCATENATION OF TRANSFORMATIONS, USING THE    "
 "            APPROPRIATE DIMENSION, T_MATRIX IS MULTIPLIED BY     "
 "            NEW_T_MATRIX, AND THE RESULT IS ASSIGNED TO T_MATRIX."
 "GLOBAL VARIABLES REFERENCED: DIMENSION/T_MATRIX.                 "
 "CALLING MODULES: TRANS/ROTATE/SCALE/REFLEC.                      "
 VAR CNTR_IND1, CNTR_IND2, CNTR_IND3, MTRX_LIMIT: INTEGER;
   TEMP_MATRIX: MATRIX_TYPE;
 BEGIN
  MTRX_LIMIT := SUCC(DIMENSION);
  FOR CNTR_IND1 := 1 TO MTRX_LIMIT DO
   FOR CNTR_IND2 := 1 TO MTRX_LIMIT DO
    BEGIN
     TEMP_MATRIX(.CNTR_IND1,CNTR_IND2.) := 0.0;
     FOR CNTR_IND3 := 1 TO MTRX_LIMIT DO
      TEMP_MATRIX(.CNTR_IND1,CNTR_IND2.) := T_MATRIX(.CNTR_IND1,
       CNTR_IND3.) * NEW_T_MATRIX(.CNTR_IND3,CNTR_IND2.)
       + TEMP_MATRIX(.CNTR_IND1,CNTR_IND2.); "END FOR CNTR_IND3"
    END; "FOR CNTR_IND1,FOR CNTR_IND2"
  T_MATRIX := TEMP_MATRIX;
 END; "PROC MATMUL"
"****************************APPLY_T_MATRIX*************************************"
PROCEDURE APPLY_T_MATRIX(STARTINDEX, ENDINDEX: INTEGER);
"PURPOSE: APPLIES THE TRANSFORMATION REPRESENTED BY THE            "
"          TRANSFORMATION MATRIX (T_MATRIX) ON A SPECIFIED         "
"          PORTION OF THE PDF (MOVE AND VECTOR COMMANDS ONLY)."
"GLOBAL VARIABLES REFERENCED: DIMENSION/T_MATRIX.                  "
"CALLING MODULES: USER PROGRAM.                                    "
 VAR CNTR_IND1, CNTR_IND2, CNTR_IND3: INTEGER;
  VECTER, TEMP_VEC: ARRAY(.1..3.) OF REAL;
 BEGIN
  FOR CNTR_IND1 := STARTINDEX TO ENDINDEX DO
   IF PDF(.CNTR_IND1.).TAG = PDFLINE
    THEN BEGIN "APPLY T_MATRIX TO THIS PDF COMMAND"
       VECTER(.1.) := PDF(.CNTR_IND1.).X_COOR;
       VECTER(.2.) := PDF(.CNTR_IND1.).Y_COOR;
       IF DIMENSION = 3
        THEN VECTER(.3.) := PDF(.CNTR_IND1.).Z_COOR;
       FOR CNTR_IND2 := 1 TO DIMENSION DO
        BEGIN
         TEMP_VEC(.CNTR_IND2.) := 0.0;
         FOR CNTR_IND3 := 1 TO DIMENSION DO
          TEMP_VEC(.CNTR_IND2.) := VECTER(.CNTR_IND3.) *
           T_MATRIX(.CNTR_IND3,CNTR_IND2.) +
           TEMP_VEC(.CNTR_IND2.); "END FOR(CNTR_IND3)"
         TEMP_VEC(.CNTR_IND2.) := TEMP_VEC(.CNTR_IND2.) +
          T_MATRIX(.(SUCC(DIMENSION)),CNTR_IND2.);
        END; "FOR CNTR_IND2"
       PDF(.CNTR_IND1.).X_COOR := TEMP_VEC(.1.);
       PDF(.CNTR_IND1.).Y_COOR := TEMP_VEC(.2.);
       IF DIMENSION = 3
        THEN PDF(.CNTR_IND1.).Z_COOR := TEMP_VEC(.3.);
      END; "IF (PDF(.CNTR_IND1.).TAG = PDFLINE),FOR CNTR_IND1"
 END; "PROC APPLY_T_MATRIX"
"*****************************TRANS*********************************************"
PROCEDURE TRANS(X_DIST, Y_DIST, Z_DIST: REAL);
"PURPOSE: CREATES A TRANSLATION TRANSFORMATION MATRIX AND          "
"          CONCATENATES IT TO ANY EXISTING TRANSFORMATIONS, THE    "
"          PARAMETERS DENOTE THE X, Y, AND Z DISTANCES THE OBJECT"
"          IS TO MOVE.                                             "
"GLOBAL VARIABLES REFERENCED: DIMENSION.                           "
"CALLING MODULES: USER PROGRAM.                                    "
 VAR CNTR_IND1, CNTR_IND2, MTRX_LIMIT: INTEGER;
   TRANS_MATRIX: MATRIX_TYPE; "TRANSLATION MATRIX"
 BEGIN
  MTRX_LIMIT := SUCC(DIMENSION);
```

```
    FOR CNTR_IND1 := 1 TO MTRX_LIMIT DO
      BEGIN
        FOR CNTR_IND2 := 1 TO MTRX_LIMIT DO
          TRANS_MATRIX(.CNTR_IND1,CNTR_IND2.) := 0.0;
          "END FOR(CNTR_IND2)"
        TRANS_MATRIX(.CNTR_IND1,CNTR_IND1.) := 1.0;
      END; "FOR CNTR_IND1"
    IF DIMENSION = 2
      THEN BEGIN
             TRANS_MATRIX(.3,1.) := X_DIST;
             TRANS_MATRIX(.3,2.) := Y_DIST
           END "THEN"
      ELSE BEGIN "DIMENSION = 3"
             TRANS_MATRIX(.4,1.) := X_DIST;
             TRANS_MATRIX(.4,2.) := Y_DIST;
             TRANS_MATRIX(.4,3.) := Z_DIST
           END; "IF DIMENSION = 2"
    MATMUL(TRANS_MATRIX); "CONCATENATE TRANS_MATRIX"
  END; "PROC TRANS"
"****************************SCALE*********************************************"
PROCEDURE SCALE(X_FACTOR, Y_FACTOR, Z_FACTOR: REAL);
"PURPOSE: CREATES A SCALE TRANSFORMATION MATRIX AND CONCATENATES"
"           IT TO ANY EXISTING TRANSFORMATIONS. THE PARAMETERS  "
"           DENOTE SCALE FACTORS ALONG THE X, Y, AND Z AXES.    "
"GLOBAL VARIABLES REFERENCED: DIMENSION,                        "
"CALLING MODULES: USER PROGRAM.                                 "
  VAR CNTR_IND1, CNTR_IND2, MTRX_LIMIT: INTEGER;
      SCALE_MATRIX: MATRIX_TYPE;
  BEGIN
    MTRX_LIMIT := SUCC(DIMENSION);
    FOR CNTR_IND1 := 1 TO MTRX_LIMIT DO
      FOR CNTR_IND2 := 1 TO MTRX_LIMIT DO
        SCALE_MATRIX(.CNTR_IND1,CNTR_IND2.) := 0.0;
        "END FOR CNTR_IND1,FOR CNTR_IND2"
    SCALE_MATRIX(.1,1.) := X_FACTOR;
    SCALE_MATRIX(.2,2.) := Y_FACTOR;
    IF DIMENSION = 2
      THEN SCALE_MATRIX(.3,3.) := 1.0
      ELSE BEGIN "DIMENSION = 3"
             SCALE_MATRIX(.3,3.) := Z_FACTOR;
             SCALE_MATRIX(.4,4.) := 1.0
           END; "IF DIMENSION = 2"
    MATMUL(SCALE_MATRIX); "CONCATENATE SCALE_MATRIX"
  END; "PROC SCALE"
"****************************REFLEC********************************************"
PROCEDURE REFLEC(INTRCHNG_X, INTRCHNG_Y, INTRCHNG_Z: REAL);
"PURPOSE: CREATES A REFLECTION OR AXIS INTERCHANGE TRANSFORMATION "
"           MATRIX AND CONCATENATES IT TO ANY EXISTING            "
"           TRANSFORMATIONS. IF A PARAMETER EQUALS -1.0, THAT AXIS"
"           IS INTERCHANGED. IF IT EQUALS +1.0, THE AXIS REMAINS  "
"           THE SAME.                                             "
"GLOBAL VARIABLES REFERENCED: DIMENSION,                          "
"CALLING MODULES: USER PROGRAM.                                   "
  VAR CNTR_IND1, CNTR_IND2, MTRX_LIMIT: INTEGER;
      REFLEC_MATRIX: MATRIX_TYPE;
  BEGIN
    MTRX_LIMIT := SUCC(DIMENSION);
    FOR CNTR_IND1 := 1 TO MTRX_LIMIT DO
      FOR CNTR_IND2 := 1 TO MTRX_LIMIT DO
        REFLEC_MATRIX(.CNTR_IND1,CNTR_IND2.) := 0.0;
        "END FOR CNTR_IND1,FOR CNTR_IND2"
    REFLEC_MATRIX(.1,1.) := INTRCHNG_X;
    REFLEC_MATRIX(.2,2.) := INTRCHNG_Y;
    IF DIMENSION = 2
      THEN REFLEC_MATRIX(.3,3.) := 1.0
      ELSE BEGIN "DIMENSION = 3"
             REFLEC_MATRIX(.3,3.) := INTRCHNG_Z;
             REFLEC_MATRIX(.4,4.) := 1.0
           END; "IF DIMENSION = 2"
    MATMUL(REFLEC_MATRIX); "CONCATENATE REFLEC_MATRIX"
  END; "PROC REFLEC"
"****************************TRIG*******************************************"
FUNCTION TRIG(SIN_OR_COS: INTEGER; THETA: REAL): REAL;
"PURPOSE: CALCULATES THE SINE OR COSINE FUNCTION OF THE VALUE"
"           THETA (GIVEN IN DEGREES).                         "
"GLOBAL VARIABLES REFERENCED: NONE.                           "
"CALLING MODULES: ROTATE.                                     "
  VAR CNTR: INTEGER;
      THETA1, THETA1_SQUARED, TEMP_TRIG,
      SIN_SIGN, COS_SIGN, FACT: REAL;
  BEGIN
    THETA1 := THETA;
    WHILE (THETA1 >= 360.0) DO "MAP THETA1 INTO 0.0 <= THETA1 < 360.0"
      THETA1 := THETA1 - 360.0; "END WHILE"
    WHILE (THETA1 < 0.0) DO "MAP THETA1 INTO 0.0 <= THETA1 < 360.0"
      THETA1 := THETA1 + 360.0; "END WHILE"
    IF SIN_OR_COS = SIN
      THEN SIN_SIGN := 1.0
      ELSE COS_SIGN := 1.0;

    "MAP THETA1 INTO 0.0 <= THETA1 <= 90.0"
    IF THETA1 <= 90.0 "THETA1 IN QUADRANT I"
      THEN THETA1 := THETA1 "DUMMY STATEMENT"
      ELSE IF THETA1 <= 270.0 "THETA1 IN QUADRANT II OR III"
             THEN BEGIN
                    IF SIN_OR_COS = COS
                      THEN COS_SIGN := -1.0
                      ELSE IF THETA1 > 180.0 "THETA1 IN QUADRANT III"
                             THEN SIN_SIGN := -1.0; "END IF SIN_OR_COS..."
                    THETA1 := ABS(180.0 - THETA1);
                  END "THEN"
             ELSE BEGIN "THETA1 IN QUADRANT IV"
                    IF SIN_OR_COS = SIN
                      THEN SIN_SIGN := -1.0;
                    THETA1 := 360.0 - THETA1;
                  END; "IF THETA1 <= 90.0, IF THETA1 <= 270.0"
    IF SIN_OR_COS = COS                "0.0 <= ANGLE <= 90.0 IMPLIES    "
      THEN THETA1 := 90.0 - THETA1; "COS(ANGLE) := SIN(90.0 - ANGLE)"

    "CALCULATE SIN(THETA1) USING A POWER SERIES APPROXIMATION"
    THETA1 := (3.14159265 / 180.0) * THETA1; "CONVERT TO RADIANS"
    FACT := 11.0;
    THETA1_SQUARED := THETA1 * THETA1;
    TEMP_TRIG := 0.99604583; "= AVERAGE ERROR OF 7TH TERM OF POWER"
                             "SERIES APPROXIMATION OF SINE        "
    FOR CNTR := 1 TO 5 DO
      BEGIN
        TEMP_TRIG := 1.0 - ((THETA1_SQUARED / (FACT * (FACT - 1.0)))
                            * TEMP_TRIG);
        FACT := FACT - 2.0;
        END; "FOR CNTR"
      TEMP_TRIG := TEMP_TRIG * THETA1;
      IF SIN_OR_COS = SIN
        THEN TRIG := TEMP_TRIG * SIN_SIGN
        ELSE TRIG := TEMP_TRIG * COS_SIGN;
    END; "FUNC TRIG"
"*****************************ROTATE*******************************************"
PROCEDURE ROTATE(AXIS: INTEGER; THETA: REAL);
"PURPOSE: CREATES A ROTATION TRANSFORMATION MATRIX AND CONCATENATES "
"           IT TO ANY EXISTING TRANSFORMATIONS. THE PARAMETER AXIS  "
"           DENOTES THE AXIS OF ROTATION (1=X,2=Y,3=Z), AND THETA   "
"           SIGNIFIES DEGREES OF ROTATION IN THE CLOCKWISE (THETA   "
"           POSITIVE) OR COUNTERCLOCKWISE (THETA NEGATIVE) DIRECTION"
"           ABOUT THE SPECIFIED AXIS.                               "
"GLOBAL VARIABLES REFERENCED: DIMENSION.                             "
"CALLING MODULES: USER PROGRAM.                                      "
  VAR CNTR_IND1, CNTR_IND2, MTRX_LIMIT: INTEGER;
      SIN_THETA, COS_THETA: REAL;
      ROTATE_MATRIX: MATRIX_TYPE;
  BEGIN
    MTRX_LIMIT := SUCC(DIMENSION);
    FOR CNTR_IND1 := 1 TO MTRX_LIMIT DO
      BEGIN
        FOR CNTR_IND2 := 1 TO MTRX_LIMIT DO
          ROTATE_MATRIX(.CNTR_IND1,CNTR_IND2.) := 0.0; "END FOR CNTR_IND2"
        ROTATE_MATRIX(.CNTR_IND1,CNTR_IND1.) := 1.0;
      END; "FOR CNTR_IND1"
    SIN_THETA := TRIG(SIN,THETA);
    COS_THETA := TRIG(COS,THETA);
    IF (DIMENSION = 2) OR (AXIS = 3)
      THEN BEGIN
             ROTATE_MATRIX(.1,1.) := COS_THETA;
             ROTATE_MATRIX(.1,2.) := - SIN_THETA;
             ROTATE_MATRIX(.2,1.) := SIN_THETA;
             ROTATE_MATRIX(.2,2.) := COS_THETA;
           END; "IF (DIMENSION = 2) OR..."
    IF DIMENSION = 3
      THEN CASE AXIS OF
             1: BEGIN "X AXIS"
                  ROTATE_MATRIX(.2,2.) := COS_THETA;
                  ROTATE_MATRIX(.2,3.) := - SIN_THETA;
                  ROTATE_MATRIX(.3,2.) := SIN_THETA;
                  ROTATE_MATRIX(.3,3.) := COS_THETA;
                END; "X AXIS CASE"
             2: BEGIN "Y AXIS"
                  ROTATE_MATRIX(.1,1.) := COS_THETA;
                  ROTATE_MATRIX(.1,3.) := SIN_THETA;
                  ROTATE_MATRIX(.3,1.) := - SIN_THETA;
                  ROTATE_MATRIX(.3,3.) := COS_THETA;
                END; "Y AXIS CASE"
             3: ROTATE_MATRIX(.4,4.) := 1.0 "DUMMY STATEMENT"
           END; "CASE,IF DIMENSION = 3"
    MATMUL(ROTATE_MATRIX); "CONCATENATE ROTATE_MATRIX"
  END; "PROC ROTATE"
"****************************APPLICATION PROGRAM********************************"

"*******************************"
"*  USER PROGRAM STARTS HERE   *"
"*******************************"

```

APPENDIX B

COMPUTEK DRIVER PROGRAM CODE

```

(NUMBER)

"PER BRINCH HANSEN                          *   AS MODIFIED FOR THE INTERDATA
                                            *      8/32 UNDER OS/32-MT AT
 INFORMATION SCIENCE                        *   DEPARTMENT OF COMPUTER SCIENCE
 CALIFORNIA INSTITUTE OF TECHNOLOGY         *      KANSAS STATE UNIVERSITY
                                            *
 UTILITY PROGRAMS FOR                       *
 THE SOLO SYSTEM                            *
                                            *
 18 MAY 1975                                *              1 DEC 1976"


"###########
#  PREFIX  #
###########"


CONST NL = '(:10:)';   FF = '(:12:)';   CR = '(:13:)';   EM = '(:25:)';

CONST PAGELENGTH = 512;
TYPE PAGE = ARRAY (.1..PAGELENGTH.) OF CHAR;

CONST LINELENGTH = 132;
TYPE LINE = ARRAY (.1..LINELENGTH.) OF CHAR;

CONST IDLENGTH = 12;
TYPE IDENTIFIER = ARRAY (.1..IDLENGTH.) OF CHAR;

TYPE FILE = 1..2;

TYPE FILEKIND = (EMPTY, SCRATCH, ASCII, SEQCODE, CONCODE);

TYPE FILEATTR = RECORD
                  KIND: FILEKIND;
                  ADDR: INTEGER;
                  PROTECTED: BOOLEAN;
                  NOTUSED: ARRAY (.1..5.) OF INTEGER
                END;

TYPE IODEVICE =
  (TYPEDEVICE, DISKDEVICE, TAPEDEVICE, PRINTDEVICE, CARDDEVICE,
   A,B,COMPUTER);

TYPE IOOPERATION = (INPUT, OUTPUT, MOVE, CONTROL);

TYPE IOARG = (WRITEEOF, REWIND, UPSPACE, BACKSPACE);

TYPE IORESULT =
  (COMPLETE, INTERVENTION, TRANSMISSION, FAILURE,
   ENDFILE, ENDMEDIUM, STARTMEDIUM);

TYPE IOPARAM = RECORD
                 OPERATION: IOOPERATION;
                 STATUS: IORESULT;
                 COUNT: INTEGER
               END;

```

```
TYPE TASKKIND = (INPUTTASK, JOBTASK, OUTPUTTASK);

TYPE ARGTAG =
  (NILTYPE, BOOLTYPE, INTTYPE, IDTYPE, PTRTYPE);

TYPE POINTER = @BOOLEAN;

TYPE ARGTYPE = RECORD
                 CASE TAG: ARGTAG OF
                   NILTYPE, BOOLTYPE: (BOOL: BOOLEAN);
                   INTTYPE: (INT: INTEGER);
                   IDTYPE: (ID: IDENTIFIER);
                   PTRTYPE: (PTR: POINTER)
               END;

CONST MAXARG = 10;
TYPE ARGLIST = ARRAY (.1..MAXARG.) OF ARGTYPE;

TYPE ARGSEQ = (INP, OUT);

TYPE PROGRESULT =
  (TERMINATED, OVERFLOW, POINTERERROR, RANGEERROR, VARIANTERROR,
   HEAPLIMIT, STACKLIMIT, CODELIMIT, TIMELIMIT, CALLERROR);

PROCEDURE READ(VAR C: CHAR);
PROCEDURE WRITE(C: CHAR);

PROCEDURE OPEN(F: FILE; ID: IDENTIFIER; VAR FOUND: BOOLEAN);
PROCEDURE CLOSE(F: FILE);
PROCEDURE GET(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);
PROCEDURE PUT(F: FILE; P: INTEGER; VAR BLOCK: UNIV PAGE);
FUNCTION LENGTH(F: FILE): INTEGER;

PROCEDURE MARK(VAR TOP: INTEGER);
PROCEDURE RELEASE(TOP: INTEGER);

PROCEDURE IDENTIFY(HEADER: LINE);
PROCEDURE ACCEPT(VAR C: CHAR);
PROCEDURE DISPLAY(C: CHAR);

PROCEDURE READPAGE(VAR BLOCK: UNIV PAGE; VAR EOF: BOOLEAN);
PROCEDURE WRITEPAGE(BLOCK: UNIV PAGE; EOF: BOOLEAN);
PROCEDURE READLINE(VAR TEXT: UNIV LINE);
PROCEDURE WRITELINE(TEXT: UNIV LINE);
PROCEDURE READARG(S: ARGSEQ; VAR ARG: ARGTYPE);
PROCEDURE WRITEARG(S: ARGSEQ; ARG: ARGTYPE);

PROCEDURE LOOKUP(ID: IDENTIFIER; VAR ATTR: FILEATTR; VAR FOUND: BOOLEAN);

PROCEDURE IOTRANSFER
  (DEVICE: IODEVICE; VAR PARAM: IOPARAM; VAR BLOCK: UNIV PAGE);

PROCEDURE IOMOVE(DEVICE: IODEVICE; VAR PARAM: IOPARAM);

FUNCTION TASK: TASKKIND;

PROCEDURE RUN(ID: IDENTIFIER; VAR PARAM: ARGLIST;
              VAR LINE: INTEGER; VAR RESULT: PROGRESULT);


```

```
PROGRAM P(VAR PARAM: ARGLIST);

CONST
  HOME_ERASE = '(:12:)';
  ERASE_STATUS = '(:14:)';
  WRITE_STATUS = '(:15:)';
  LINE_FEED = '(:10:)';
  BACKSPACE = '(:8:)';
  FOUR_BYTE_ABS = '(:28:)';
  AT_SIGN = '(:64:)';
  NULL_BYTE = '(:0:)';

"INPUT COMMANDS"

  ERASE = 0;
  WRITE = 1;
  MOVE = 2;
  VECTOR = 3;
  HTEXT = 4;
  VTEXT = 5;
  CLEAR = 6;
  EOT = 7;
  DEBUG = 8;

TYPE

  PACKED_LINE = ARRAY [1..66] OF INTEGER;
  PACKED_REAL = ARRAY [1..4] OF INTEGER;
  PAGE_INDEX = 0..256;
  SEND_INDEX = 0..512;
  TEXT_INDEX = 0..45;

  CH_HWD = ARRAY[1..2] OF CHAR;

VAR

  HEX : ARRAY [0..15] OF CHAR;

  DISPLAY_PAGE : ARRAY [1..256] OF INTEGER;
  OUTPAGE : PAGE;
  PAGE_CNTR : PAGE_INDEX;
  SEND_CNTR : SEND_INDEX;
  TEXT_LINE : ARRAY [1..44] OF CHAR;
  EOF,CHAR_MODE,VISIBLE,DEBUG_FLAG : BOOLEAN;

PROCEDURE BUMP_PAGE_CNTR;

"INCREMENT INDEX OF INPUT PAGE
 WHEN ONE PAGE IS EXHAUSTED GET A NEW PAGE
   AND INITIALIZE THE PAGE INDEX"

  BEGIN
    IF PAGE_CNTR = 256 THEN BEGIN
      READPAGE(DISPLAY_PAGE,EOF);
      PAGE_CNTR := 0;
    END;
    PAGE_CNTR := SUCC(PAGE_CNTR);
  END;

PROCEDURE GET_CHARPAIR(VAR INTCHAR: UNIV INTEGER);
```

```

"UNPACK TWO CHARACTERS OF THE CHARACTER LINE
PARAMETER FOR HTEXT AND VTEXT COMMANDS"

  BEGIN
    BUMP_PAGE_CNTR;
    INTCHAR := DISPLAY_PAGE[PAGE_CNTR];
  END;

PROCEDURE GET_POINT(VAR X,Y: UNIV PACKED_REAL);

"UNPACK POINT PARAMETER FOR MOVE
 OR VECTOR COMMANDS"

  VAR I:INTEGER;
  BEGIN
    FOR I := 1 TO 4 DO BEGIN
      BUMP_PAGE_CNTR;
      X[I] := DISPLAY_PAGE[PAGE_CNTR];
    END;
    FOR I := 1 TO 4 DO BEGIN
      BUMP_PAGE_CNTR;
      Y[I] := DISPLAY_PAGE[PAGE_CNTR];
    END;
  END;

PROCEDURE GET_TERMINAL_POINT(VAR XINT,YINT:INTEGER);

"COMPUTE COMPUTEK POINT COORDINATE
 EQUIVALENTS OF REAL WORLD POINT COORDINATES"

  VAR X,Y:REAL;
  BEGIN
    GET_POINT(X,Y);
    XINT := TRUNC(X) MOD 256;
    YINT := TRUNC(Y) MOD 256;
  END;

PROCEDURE GETBYTES(VAR HIGH,LOW,HALFWD: UNIV CH_HWD);
BEGIN
  HIGH[1] := '(:0:)';
  HIGH[2] := HALFWD[1];
  LOW[1] := '(:0:)';
  LOW[2] := HALFWD[2];
END;

PROCEDURE PRINTABS(ARG:UNIV INTEGER);

"DEBUGGING PROCEDURE"

"CALCULATE AND DISPLAY HEX EQUIVALENTS OF
 ASCII CHARACTERS TO BE TRANSMITTED TO
 COMPUTEK TERMINAL"

  VAR T:ARRAY[1..4] OF CHAR;
      LOW,HIGH,REM,DIGIT,I:INTEGER;
  BEGIN
  REM := ARG;
  GETBYTES(HIGH,LOW,REM);
  T[1]:= HEX[HIGH DIV 16];
  T[2] := HEX[HIGH MOD 16];
  T[3] := HEX[LOW DIV 16];
  T[4] := HEX[LOW MOD 16];
  DISPLAY(T[1]); DISPLAY(T[2]); DISPLAY(T[3]); DISPLAY(T[4]); DISPLAY(' ');
END;

PROCEDURE SEND;

"TRANSMIT ASCII CONTROL CHARACTER PAGE
 TO COMPUTEK TERMINAL"

  VAR ARG:IOPARAM; CH:CHAR; I : INTEGER;
  BEGIN
    FOR I := 1 TO SEND_CNTR DO BEGIN
      CH := OUTPAGE[I];
      IF DEBUG_FLAG THEN BEGIN
        PRINTABS(CH);
        IF I MOD 10 = 0 THEN DISPLAY(NL);
      END;
    END;
    IF DEBUG_FLAG THEN DISPLAY(NL);
    WITH ARG DO BEGIN
      OPERATION := OUTPUT;
      COUNT := SEND_CNTR;
    END;
    IOTRANSFER(COMPUTEK,ARG,OUTPAGE);
    "IF IORESULT <> COMPLETE THEN BOMB;"
  END;

PROCEDURE BUMP_SEND_CNTR;

"INCREMENT INDEX OF THE OUTPUT PAGE,
 WHEN PAGE IS FULL TRANSMIT AND
 BEGIN NEW OUTPUT PAGE"

  BEGIN
    IF SEND_CNTR = 512 THEN BEGIN
      SEND;
      SEND_CNTR := 0;
    END;
    SEND_CNTR := SUCC(SEND_CNTR);
  END;

PROCEDURE SEND_BYTE(CH:CHAR);

"ADD A SINGLE ASCII CHARACTER TO THE OUTPUT PAGE"

  BEGIN
    BUMP_SEND_CNTR;
    OUTPAGE [SEND_CNTR] := CH;
  END;

PROCEDURE SEND_AT_SIGN;

"ADD THE ASCII CHARACTERS TO THE OUTPUT PAGE
 WHICH WILL CHANGE THE COMPUTEK TERMINAL
 FROM FOUR BYTE ABSOLUTE MODE TO
 CHARACTER MODE"

  VAR I:INTEGER;
  BEGIN
    SEND_BYTE(AT_SIGN);
    FOR I := 1 TO 3 DO SEND_BYTE(NULL_BYTE);
  END;

PROCEDURE SEND_4_BYTE(X,Y:INTEGER; VISIBLE:BOOLEAN);

"COMPUTE THE FOUR ASCII CHARACTERS THAT
 EXECUTE A MOVE OR VECTOR COMMAND,
 X AND Y ARE INTEGER COORDINATES OF
 THE TARGET POINT.  VISIBLE = TRUE FOR
VECTOR,  VISIBLE = FALSE FOR MOVE"

  VAR ACCUM:INTEGER;
  BEGIN
    ACCUM := 2;
    IF (Y MOD 2) = 1 THEN ACCUM := ACCUM + 16;
    IF ((Y DIV 2) MOD 2) = 1 THEN ACCUM := ACCUM + 32
    ELSE ACCUM := ACCUM + 64;
    IF VISIBLE THEN ACCUM := ACCUM + 1;
    SEND_BYTE(CHR(ACCUM));
    ACCUM := Y DIV 4;
    IF ((ACCUM DIV 32) MOD 2) = 0 THEN ACCUM := ACCUM + 64;
    SEND_BYTE(CHR(ACCUM));
    ACCUM := 0;
    IF (X MOD 2) = 1 THEN ACCUM := ACCUM + 16;
    IF ((X DIV 2) MOD 2) = 1 THEN ACCUM := ACCUM + 32
    ELSE ACCUM := ACCUM + 64;
    SEND_BYTE(CHR(ACCUM));
    ACCUM := X DIV 4;
    IF ((ACCUM DIV 32) MOD 2) = 0 THEN ACCUM := ACCUM + 64;
    SEND_BYTE(CHR(ACCUM));
  END;

PROCEDURE PROCESS_DRAW(KEY:INTEGER);

"PUT COMPUTEK IN FOUR BYTE ABSOLUTE MODE,
 UNPACK AND TRANSLATE TO COMPUTEK
 COORDINATES THE TARGET POINT,
 COMPUTE AND TRANSMIT A FOUR BYTE ABSOLUTE COMMAND"

  VAR X,Y:INTEGER;
  BEGIN
    IF CHAR_MODE THEN BEGIN
      SEND_BYTE(FOUR_BYTE_ABS);
      CHAR_MODE := FALSE;
    END;
    GET_TERMINAL_POINT(X,Y);
    IF KEY = VECTOR THEN VISIBLE := TRUE ELSE VISIBLE := FALSE;
    SEND_4_BYTE(X,Y,VISIBLE);
  END;

PROCEDURE PROCESS_TEXT(KEY:INTEGER);

"PUT COMPUTEK IN CHARACTER MODE.
 UNPACK TEXT PARAMETER.  TRANSMIT TEXT
 CHARACTERS.  FOR VERTICAL TEXT,
 TRANSMIT LINEFEED AND BACKSPACE
 BETWEEN TEXT CHARACTERS"

```

```
  VAR VERTICAL:BOOLEAN; I:TEXT_INDEX;
      CHAR_PAIR:ARRAY [1..2] OF CHAR;
  BEGIN
    IF NOT CHAR_MODE THEN BEGIN
      SEND_AT_SIGN;
      CHAR_MODE := TRUE;
    END;
    IF KEY = VTEXT THEN VERTICAL := TRUE ELSE VERTICAL := FALSE;
    I := 1;
    REPEAT
      GET_CHARPAIR(CHAR_PAIR);
      TEXT_LINE[I] := CHAR_PAIR[1];
      TEXT_LINE[SUCC(I)] := CHAR_PAIR[2];
      I := I + 2;
    UNTIL CHAR_PAIR[2] = EM;
    I := 1;
    REPEAT
      SEND_BYTE(TEXT_LINE[I]);
      I := SUCC(I);
      IF VERTICAL THEN BEGIN
        SEND_BYTE(LINE_FEED);
        SEND_BYTE(BACKSPACE);
      END;
    UNTIL TEXT_LINE[I] = EM;
  END;

PROCEDURE PROCESS_PAGE;

"INITIALIZE COUNTERS AND FLAGS.
 PROCESS COMMANDS UNTIL EOT,
 LEAVE TERMINAL IN CHARACTER MODE
 FOR NEXT TRANSMISSION"

  VAR EOTRANSFER:BOOLEAN; KEY:INTEGER;
  BEGIN
    SEND_CNTR := 0;
    PAGE_CNTR := 0;
    EOTRANSFER := FALSE; CHAR_MODE := TRUE;
    REPEAT
      BUMP_PAGE_CNTR;
      KEY := DISPLAY_PAGE[PAGE_CNTR];
      CASE KEY OF
        MOVE,VECTOR: PROCESS_DRAW(KEY);
        CLEAR,WRITE,ERASE:BEGIN
          IF NOT CHAR_MODE THEN SEND_AT_SIGN;
          CASE KEY OF
            CLEAR: SEND_BYTE(HOME_ERASE);
            WRITE: SEND_BYTE(WRITE_STATUS);
            ERASE: SEND_BYTE(ERASE_STATUS)
          END; "CASE"
        END;
        HTEXT,VTEXT: PROCESS_TEXT(KEY);
        DEBUG: DEBUG_FLAG := TRUE;
        EOT: EOTRANSFER := TRUE
      END; "CASE"
    UNTIL EOTRANSFER;
    IF NOT CHAR_MODE THEN SEND_AT_SIGN;
    SEND;
  END;

```

```
BEGIN
  "INITIALIZE;"
  HEX[0] := '0';
  HEX[1] := '1';
  HEX[2] := '2';
  HEX[3] := '3';
  HEX[4] := '4';
  HEX[5] := '5';
  HEX[6] := '6';
  HEX[7] := '7';
  HEX[8] := '8';
  HEX[9] := '9';
  HEX[10]:= 'A';
  HEX[11]:= 'B';
  HEX[12]:= 'C';
  HEX[13]:= 'D';
  HEX[14]:= 'E';
  HEX[15]:= 'F';


"INITIALIZE COMPUTEK TERMINAL BY
 CLEARING SCREEN AND HOMING THE CURSOR.
 READ INPUT COMMAND PAGES AND PROCESS
 UNTIL PAGE MARKED EOF IS RECEIVED.
 SEND NORMAL TERMINATION MESSAGE TO
 INITIATING PROCESS"

  SEND_CNTR := 0;
  CHAR_MODE := TRUE;
  DEBUG_FLAG := FALSE;
  SEND_AT_SIGN;
  SEND_BYTE(HOME_ERASE);
  SEND;
  REPEAT
    READPAGE(DISPLAY_PAGE,EOF);
    IF NOT EOF THEN PROCESS_PAGE;
  UNTIL EOF;
  "WRAPUP;"
  PARAM[1] .BOOL := TRUE;
END.
```

APPENDIX C

FORTRAN/SPASCAL CSGP MODULES

FUNCTIONAL GROUP	FORTRAN MODULES	SPASCAL MODULES
Image	START(7)	START(7)
Construction	MOVE(10)	MOVE(18)
	VEC(10)	VECTOR(18)
	CLEAR(7)	CLEAR(12)
	WMODE(8)	WMODE(12)
	EMODE(8)	EMODE(12)
	SENDPDF(7)	SEND_PDF(12)
	HTEXT(11)	HTEXT(13)
	VTEXT(10)	VTEXT(13)
	DTEXT(16)	PUTTEXTINPDF(32)
	{1}SEND(8)	{2}SET_INDEX(6)
		" VAL_INDEX(4)
		" SET_INDEX(5)
		" DEBUG_CMPTK_DRVR(12)
PDF Display	COMPIL(36)	COMPIL(40)
	{3}CMPUTK(145)	LOAD_CMPTK_DRVR(6)
	" TTYIO(78)	{5}CSGPCOMPUTEK(235)
		PUT_DISPPAGE(17)
		GET_TEXT(16)
		PACK_CHAR(14)
		PACK_REAL(8)
PDF/Error	DUMPPD(43)	DUMPPDF_PRINTER(55)
Output	{4}HCOPY(190)	DUMPPDF_CONSOLE(53)
	" YSORT(34)	ERROR(15)
	" CLIPH(64)	DISPSTRNG(13)
	" GETC(8)	LOAD_PRINTER_PROG(6)
	" PUTC(8)	{5}PRINTER
		PUT_PRNTPAGE(23)
		GET_TEXT_OUT(12)
		INT_TO_STR(23)
		REAL_TO_STR(53)
Image	INIT(11)	INIT_T_MATRIX(12)
Transformation	MATMUL(14)	MATMUL(17)
	DAPPLY(18)	APPLY_T_MATRIX(27)
	TRANS(22)	TRANS(24)
	SCALE(22)	SCALE(18)
	REFLEC(24)	REFLEC(18)
	ROTATE(37)	ROTATE(39)

	{1}GENROT(40)	TRIG(44)
	{4}PERSP(17)	
	" CLIP(54)	
	" CLIP2(62)	
	" CODE2(8)	
	" CL2(44)	
	" AND(10)	
	" DSAVE(21)	
	" DCOPY(21)	
Computek Terminal Input	{4}RPOINT(10)	
	" VMODE(?)	
	" PTCNVT(21)	
	FORTRAN	SPASCAL
	TOTALS (1164)*	TOTALS (964)*
	(544)**	(884)**

Notation:

( ) Lines of code (does not include comments).

* Total number of lines of code.

** Total of functional equivalent lines of code.

{1} FORTRAN modules not required for the SPASCAL-INTERDATA 8/32 implementation.

{2} New capabilities added to the SPASCAL CSGP.

{3} IBM/370 assembler subroutines.

{4} FORTRAN modules left for future conversion efforts.

{5} SPASCAL programs that execute in the SOLO output process partition. CSGPCOMPUTEK is the name of the text (ASCII) file containing the revised version of M. Neal's Computek driver program. PRINTER is the name of the file containing the object code (SEQCODE) of a system program that runs the line printer.

CONVERSION OF A GRAPHICS PACKAGE TO SEQUENTIAL PASCAL

by

DANIEL THOMAS SNYDER

B. S., Ohio State University, 1973

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1978

ABSTRACT

The objective of the project was to port the Computer Science Graphics Package (CSGP), written in FORTRAN for a large computer environment (IBM/370), to a minicomputer environment (INTERDATA 8/32), coded in sequential Pascal (SPASCAL). Both implementations interact with the Computek 300 GT display terminal.

The CSGP allows interactive communication between a Computek terminal user and a remote computer which executes programs to construct, transform, and display three-dimensional straight-line pictures. The graphics package software consists of a set of routines which builds and manipulates a data structure image representation, called the Pseudo Display File (PDF), and translates the PDF into code suitable for display at the terminal.

There were basically two reasons behind the desire to port the CSGP. It was intended that moving to a minicomputer environment would remove the package from a time sharing system to a system with a faster response mode, and converting from FORTRAN to SPASCAL would gain better programming features.

The porting of the CSGP was more than just a line-by-line conversion of FORTRAN code to SPASCAL. The change in computing environments necessitated complete revision of

the computer-to-terminal interface software and additional I/O support routines. The change to SPASCAL caused major changes to the PDF data structures and the modularity of the CSGP software in order to take advantage of the powerful structures and structured programming features provided by SPASCAL.

The original CSGP implementation is approximately 550 lines of FORTRAN and IBM/370 assembler code compared to 880 functional equivalent lines of SPASCAL that comprise the new version. The manhours expended during each phase of the project were as follows: Organization/Design, 130; Coding, 20; Test/Debug, 100; Documentation, 80; and Total, 330.