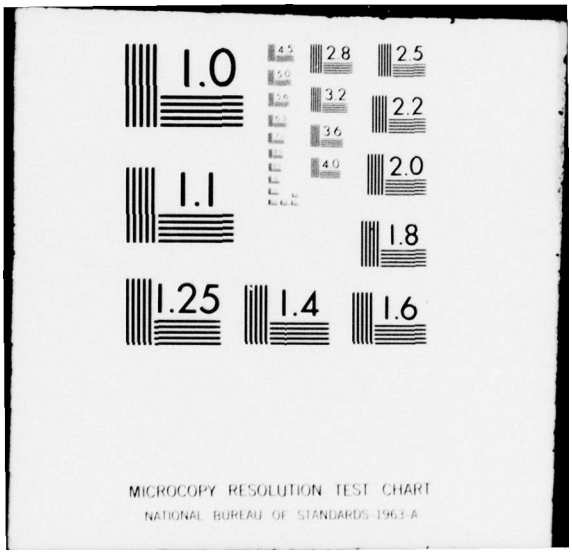
AD-A059 903 ARMY ENGINEER WATERWAYS EXPERIMENT STATION VICKSBURG MISS F/G 9/2
THREE-DIMENSIONAL GRAPHICS COMPATIBILITY SYSTEM, VERSION 2.0. H--ETC(U)
JUL 78

UNCLASSIFIED NL

1 OF 1
AD A059903

END
DATE FILMED
12-78
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

LEVEL II

WATERWAYS EXPERIMENT STATION

Vicksburg, Miss.

HOST-COMPUTER IMPLEMENTATION GUIDELINES

for the WES Automatic Data Processing Center

(6) THREE-DIMENSIONAL

GRAPHICS COMPATIBILITY SYSTEM,

Version 2.0.

HOST-COMPUTER IMPLEMENTATION GUIDELINES,

(11) 5 Jul 78 (12) 34 p.

July 5, 1978

038 100

3-D GCS Version 2.0 Host-computer Implementation Guidelines

I. Introduction.

The following paragraphs develop guidelines for implementing GCS on a host computer system. Included will be the definition of computer-dependent Graphics Status Area (GSA) initialization values, functional specifications for each of the 3D GCS computer-dependent routines, a discussion of factors involved in supporting 3D GCS on a computer system, and a description of a phased sequence for bringing up GCS on a new computer system.

II. Graphics Status Area Initialization.

In this section, each of the computer-dependent elements of the Graphics Status Area will be listed along with the procedures required to calculate the value, if appropriate.

KTERM GCS string terminator character. Default value is the internal computer character set representation of an ASCII backslash ("\") character.

KUCASE Upper case shift character. Default value is the internal computer character set representation of an ASCII less-than ("<") character.

KLCASE Lower case shift character. Default value is the internal computer character set representation of an ASCII greater-than (">") character.

KSUBCH Subscript escape character. Default value is the internal computer character set representation of an ASCII underscore ("_") character.

KSUPCH Superscript escape character. Default value is the internal computer character set representation of an ASCII pound sign ("#") character.

KWRTFL Standard alphanumeric output file. Default value is the computer system default alphanumeric output Fortran file number.

KRDFL Standard alphanumeric input file. Default value is the computer system default alphanumeric input Fortran file number.

KOUTFL Standard graphics output file. Default value is the computer system default graphics output Fortran file number if one exists.

III. 3D GCS Computer-dependent Routines.

This section describes the functions embodied in each of the 3D GCS computer-dependent routines. This discussion will include interface specifications and functional specifications. Since these functions are computer-dependent, the requirement to implement these routines in ANS Fortran is relaxed. However, use of a higher-level language for implementation is recommended for ease of maintenance. The functions are grouped into four categories: packing, job control, character set conversion, and I/O. Within these categories, the routines will be listed alphabetically.

Packing Routines

The packing routines provide for packing and unpacking arbitrary length bit strings (GCSBIT, GCSBTR), for packing and unpacking single characters (GCS1CH, GCS1PK), and for extracting the left-most n-character wide field from a character string (GCSSTD). Two former computer-dependent routines, GCSPCK and GCSUPK, have been rewritten to call GCS1CH and GCS1PK and are now device and computer system independent.

GCSBIT(NEXT,IWIDTH,IWORD,IBITLC,IBUF)

NEXT is a word containing the bits to be packed right-justified, zero-fill.

IWIDTH is the number of bits to be packed (i. e., the number of bits in NEXT).

IWORD is the word in the buffer in which packing will occur.

IBITLC is the number of bits currently used within IWORD.

IBUF is an array in which the bit string is accumulated. The calling routine should provide a buffer one word larger than the actual buffer size since GCSBIT will leave overflow bits in this word.

Comments: This routine accumulates a packed bit string of arbitrary length within a buffer provided by the calling program. This allows the formation of strings of bits in which the storage of the bits ignores word boundaries.

GCSBTR(NEXT,IWIDTH,IWORD,IBITLC,IBUF)

NEXT is a word in which the bits retrieved will be placed right-justified, zero-fill.

IWIDTH is the number of bits to be retrieved (i. e., placed in NEXT).

IWORD is the word in the buffer from which bits are currently being retrieved.

IBITLC is the number of bits already retrieved from IWORD.

IBUF is an array containing the bit string from which bits will be retrieved. The calling routine should provide a word of zero bits immediately following the IBUF array.

Comments: This routine retrieves string of bits (up to one word) from the packed bit string provided by the calling program (i. e., the buffer IBUF). Word boundaries will be ignored during retrieval.

GCS1PK(N,INCHAR,IBUFR)

N is the character position within the output character string into which the KBYTEL-width character is to be packed.

INCHAR is a word containing one KBYTEL-width character right-justified, zero-fill.

IBUFR is the output character string word into which the KBYTEL-widht character will be inserted.

Comments: This routine inserts the KBYTEL-width character contained in INCHAR into the N^{th} KBYTEL-width character position of the output character string IBUFR. Character positions are numbered left to right. Note that KBYTEL is not a fixed quantity and may vary during execution.

GCS1CH(N,ISTRNG,NUCHAR)

N the character position within the input character string from which the KBYTEL-width character is to be extracted.

ISTRNG is the input character string (maximum length is one word).

NUCHAR is a word in which the KBYTEL-width character is placed.

Comments: This routine extracts the N^{th} KBYTEL-width character from the input character string word and places it in the output word (NUCHAR) right-justified, zero-fill. Note that KBYTEL is not a fixed quantity and may vary during execution.

GCSSTD(N,ISTRNG,NUSTRG)

N is the number of left-most characters from the input character string to comprise the output character string.

ISTRNG is the input character string.

NUSTRG is the left-justified output character string padded with zero bits if necessary to fill the word.

Comments: This routine places the left-most N characters from the input character string in the output character string word padding, if necessary, with zero bits. Both the input and output character strings may be of maximum length of four characters or one word whichever is greater.

Job Control Routines

Only two job control routines currently are required, GCSJOB and GCSTBK. These routines are intended to access operating system facilities for services or to obtain information.

GCSJOB(JOBID,IDUSER,IROUTE,ICLASS,IDATE,ITIME)

JOBID is a word in which the job ID of the run will be returned.

IDUSER is a word in which the ID of the person executing the program will be returned.

IROUTE is the routing address to which the output should be sent.

ICLASS is the job security classification.

IDATE is the date of the run.

ITIME is the time of the call to GCSJOB.

Comment: The GCS terminator character is appended to the end of each field of information being returned to the calling program. The calling program has the responsibility of insuring that the current terminator is the default terminator.

A maximum of twelve characters including the GCS terminator character are allowed to be returned for each field. The only exception is that the security classification may be up to eight words long.

If a parameter is not defined on a particular operating system, a valid GCS text string consisting of only the default GCS text string terminator must be provided.

****
****** This routine is still under development. ******
****** The description above is subject to ******
****** change. Providing this routine is opt- ******
****** ional until development is completed. ******
****

GCSTBK(IERROR)

IERROR contains the GCS error number.

Comments: This routine invokes the computer system error traceback routine. It is implemented so that tracebacks can occur when GCS errors have been identified. It is important that the traceback not abort execution of the program. If no traceback function exists or the traceback aborts execution of the program, this routine should be null.

Character Set Conversion Routines

These routines provide for converting between the GCS internal character set ASCII and the host computer internal character set. Note that even if the host computer character set can support both upper and lower case characters, the mechanism must still be provided to support the GCS case shifting function. This may require that duplicate conversion tables be included whose only difference is that all upper case characters be mapped to lower case when lower case has been specified. If the host computer character set does not support the entire ASCII special character graphics, then some special characters may also differ between upper and lower cases. If this occurs, however, the special characters (,), +, -, =, *, /, <, >, and \ must be supported in both cases. Only the printable characters need be converted; control characters and character indecies with no graphic symbol associated with them need not be handled unless desired. It should be emphasized that every attempt should be made to accommodate the standard mapping between ASCII and the host-computer character set defined by the host-computer manufacturer for the operating system being utilized.

GCSCVT(IN,IOUT)

IN is the charcter to be converted in host-computer internal character set right-justified, zero-fill.

IOUT is a word in which the character will be placed after being converted to ASCII right-justified, zero-fill.

Comments: This routine converts from host-computer internal character set to ASCII. The case-shifting constant will have been added to the host-computer character before this routine is called. Therefore, the case shifting should be recognized by the magnitude of the character bit value.

GCSRVT(IN,IOUT)

IN is the character to be converted in ASCII right-justified, zero-fill.

IOUT is a word in which the character will be placed after conversion to host-computer internal character set.

Comments: This routine converts from ASCII to host-computer internal character set. Since ASCII supports both upper and lower case, it may be necessary to map lower case characters into upper case characters if the host-computer character set supports only upper case.

GCSOFS(IFILCD)

IFILCD is the Fortran file number of the sequential file to be opened.

Comments: This routine opens a file for sequential access.

Input/Output Routines

Three classes of I/O routines are included in this category: sequential file I/O, random file I/O, and telecommunications interface routines.

The sequential file I/O routines GCSOFS, GCSRFS, GCSWFS, and GCSCFS are required for use by devices which place plot output on sequential files. Use of these routines is required since Fortran I/O may place undesirable control information on a file when using unformatted I/O (Formatted I/O cannot be used since it is line-oriented with line length restrictions on some computers).

The random file I/O routines GCSOFR, GCSRFR, GCSWFR, and GCSCFR are required for supporting GCS segmentation and structure facilities. These routines are required since ANS Fortran does not have random file I/O capability.

The telecommunication routines TINPUT and TOUTPT are required for communication between GCS device-dependent routines and devices connected to the host-computer via telecommunications lines. These are required since the interface may connect directly to the operating system telecommunications service routines.

GCSRFR(IFILCD,INDEX,IRECNR,IRECLN,IRECRD)

IFILCD is the Fortran file number of the random file.

INDEX is an array in which the random file index may be maintained.

IRECNR is the record number of the record to be retrieved.

IRECLN is the length of the record to be retrieved.

IRECRD is an array into which the record will be read.

Comments: This routine performs a random (direct) read of the desired record from the specified file. If the record is larger than the buffer array IRECRD, only the first IRECLN words will be placed in the buffer area. If the record is smaller than the buffer area, the remaining words of the buffer will be unchanged. If the record does not yet exist, the buffer array will be zero-filled.

GCSRFS(IFILCD,ILENG,IBUFFR,ISTAT)

IFILCD is a Fortran file number.

ILENG specifies the length of the input buffer.

IBUFFR is an array of size ILENG words into which the data will be read.

ISTAT is a status indicator. Valid values are:

- 0 = End of file
- >0 = Actual number of words read
- <0 = Error occured during the read operation. Absolute value is actual number of words read.

Comments: This routine reads the next physical record from the file into the buffer. If the number of words read is less than the buffer size, unused buffer locations will not be modified. If the number of words read is greater that the buffer size, only the first ILENG words of the record will be placed in the buffer. Excess words are discarded. Note that in all cases, the number of words read will be indicated in ISTAT. If an end-of-file is encountered, ISTAT is set to zero.

GCSOFR(IFILCD,INDEX,INDXSZ)

IFILCD is the Fortran file number for the random file.

INDEX is an array in which the index for the random file can be maintained if not maintained by the operating system.

INDXSZ is the number of words in the INDEX array.

Comments: This routine opens a file for random access and initializes the index for the file if necessary.

GCSWFS(IFILCD,LENGTH,IBUFFR,ISTAT)

IFILCD is a Fortran file number.

ILENG specifies the number of words of data to be written.

IBUFFR is an array containing the ILENG words of data to be written.

ISTAT is a status indicator. Valid values are:

0 = No error during processing.
1 = Error during processing.

Comments: This routine writes the ILENG words of data in the buffer to the file. No other information may be placed on the file. This means that it may not be possilbe to use unformatted Fortran I/O since control information is frequently written to the file along with the data. Also, it may not be possible to tuse formatted Fortran I/O since the number of characters which can be written is often limited to one line and end-of-line symbols may be inserted within the data stream. Since files written using GCSWFS are often read by tape drives attached to other than the host-computer, it is important that the information written to the file contain no computer-system dependent control information.

GCSCFS(IFILCD,IEOF)

IFILCD is the Fortran file number of the sequential file to be closed.

IEOF is an end-of-file request flag. If set to one, an end-of-file mark is written on the end of the file. All other values inhibit writing of an end-of-file.

Comments: This routine closes the file specified in IFILCD if required by the computer system. An end-of-file mark must be written if IEOF has value 1. The close operation takes place with no rewind.

GCSWFR(IFILCD,INDEX,IRECNR,IRECLN,IRECRD)

IFILCD is the Fortran file number of the random file to be written upon.

INDEX is an array which may be used to maintain the index for the random file.

IRECNR is the number of the record to be written.

IRECLN indicates the number of words in the record to be written.

IRECRD is an array of IRECLN words containing the data to be written.

Comments: This routine writes or rewrites the record indicated. If the record did not exist before, the record will be written. If the record already exists and is being modified, the record should be rewritten in place.

GCSCFR(IFILCD)

IFILCD is a Fortran file number.

Comments: This routine closes the random file indicated by IFILCD. Clsoing a random file may require invocation of an operating system support routine to write the index on to the file.

TINPUT(ICOUNT,IBUFFR,IPRMPT)

ICOUNT is the number of ASCII characters to be accepted from the terminal.

IBUFFR is a buffer into which the ASCII characters will be placed one character per word, right-justified, zero-fill. The buffer is considered to contain ICOUNT words.

IPRMPT is an array containing a prompt sequence which is used to initiate the input operation. The first word of the array contains a count of the number of characters in the prompt sequence and is also the number of words in the IPRMPT array minus 1. The prompt characters are in the same format as would be passed to TOUTPT.

Comments: This routine initiates an input operation which asks for ICOUNT ASCII characters to be read from the terminal. If less than ICOUNT characters are received, the remaining buffer positions are zero-filled. If more than ICOUNT characters are received, excess characters are ignored. The input operation is prefixed by sending the prompt sequence to the terminal (if IPRMPT(1) is not equal to zero). Preferably, this would occur as part of the input request but, on some systems, it may be necessary to call TOUTPT to send the prompt to the terminal. The important thing is to make the delay between the time the prompt is apparent to the terminal operator and when input can be accepted by the operating system indetectable to the terminal operator.

Note that input characters placed in IBUFFR must be ASCII. If ASCII originates at the terminal, the characters must not be modified before being placed in the buffer. It is also important that any characters being buffered within TOUTPT or the operating system be sent to the terminal prior to the prompt sequence.

TOUTPT(ICOUNT,IBUFFR)

ICOUNT is the number of characters to be transmitted to the terminal.

IBUFFR is an array of ICOUNT words. Each word contains one ASCII character, right-justified, zero-fill.

Comments: This routine transmits the ASCII characters within IBUFFR to the terminal unmodified with no additional characters inserted. If desirable, the characters may be packed into buffers before sending to the terminal. In this case, the buffer should be sent when full or when TOUTPT is called with ICOUNT equal to zero.

IV. 3D GCS Implementation Factors.

3D GCS makes few assumptions about the capabilities of the computer and operating system in whose environments it will function. Essentially, 3D GCS (or even 2D GCS) can be successfully and easily supported on any computer system which has the following characteristics:

a) A word size of 32 bits or larger.

b) A loader which allows several libraries to be searched during the linkage edit of the user program.

c) A capability for reading and writing direct access (random access) files.

d) A capability for sending arbitrary length strings of ASCII characters to a display device (required for interactive devices only).

With these capabilities, it is a relatively straight-forward task to bring up 3D GCS. Section V contains a phased sequence for accomplishing this.

Circumventing the lack of any of these capabilities or characteristics can be a laborious undertaking and, in the case of direct file I/O, may not even be possible. The following paragraphs are designed to assist the implementor in modifying 3D GCS to accommodate limitations in the host-computer system.

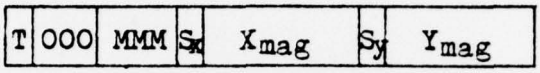
Word Size

GCS was originally developed for computers whose words consist of at least 32 bits and in which may be stored at least four host-computer characters. There are only three areas in 3D GCS where this limitation is critical.

Wherever GCS mode and option names are specified, only the first four characters of each mode or option name are significant. 3D GCS assumes these four characters occupy positions in the same word. The processing of these modes and options takes place in many GCS routines. Besides USET and UPSET, included in these routines are UCOLOR, UDOIT, UFORMT, UQUERY, all user-callable segmentation routines, and all user-callable structure routines. These routines all call the computer-dependent routine GCSSTD to extract the first four characters of the option. GCSSTD can be written to process multiple-word character strings. However, the comparisons which take place in USET, UPSET, UQUERY, and the others will require code modification to cause correct matching. An easier solution, if available, is to use double word integers.

There are several areas within GCS in which character string constants are define and initialized by DATA statements (e. g., UTAXIS). These will have to be located and modified as necessary. They can frequently be found by looking for the character strings "/4H" and ",4H".

The third area in which GCS requires this restriction is in the storage of character descriptors for the GCS character set. Each stroke of a character is stored in a 24-bit field in the lower portion of a computer word as follows:



This 24-bit field is split into the three 8-bit subfields shown above. The X and Y bytes represent percentages of a character enclosing box in sign/magnitude format (S=1 represents negative). The MMM field has value 0 for a visible stroke and has value 5 for a move. The T field flags the last stroke in the character when set to 1. While it is possible to store the character stroke in a three-byte field, unless a computer is byte-addressable, such a packing scheme is too inefficient to be acceptable. Once again, use of double-word integers is indicated, if available. If not, each stroke may be stored in two-word table entries with the appropriate changes made to the referencing code in GCSSIM and GCSSYM. The recommended split is to place the opcode (MMM), termination (T), and X fields in one word and the Y field in another. The termination field should be moved down so that it does not make the contents of the word negative when referenced arithmetically.

Cyclic Library Searches

GCS is organized into device-independent routines, computer-dependent routines (which are device-independent), and device-dependent routines. Normally, all device-independent routines are stored in one device-independent library, and each set of device-dependent routines are stored in a separate device-dependent library. The appropriate device-dependent library is selected at load time for the device to be used. A linkage editor or loader which can search several libraries as if one (actually, logically concatenate the libraries) is a great aid in using GCS on a computer system. Such an editor will perform a cyclic search to satisfy all external references. Some linkage editors and loaders cannot search a library a second time. Frequently, this effect can be achieved by listing the device-independent and device-dependent libraries twice causing each to be searched twice. A more practical approach might be to store the computer-dependent routines and GCSSIM in each of the device-dependent libraries.

Direct Access I/O

It will be very difficult to support GCS on a computer system which does not support direct access file I/O. This is because both the GCS structure capability and the GCS segmentation capability require direct access files for storing the data. (It is also intended that the planned support for Hershey characters will require that these also be stored on a direct file). The best attempt to support GCS on such a computer system would be to simulate direct access files using sequential files. It should not be expected that GCS will function efficiently when using structures or segmentation in this case.

Telecommunications Protocol

When sending graphics commands to a display device connected via a telecommunications line, GCS assumes that any number of characters can be transmitted to the display device without sending some system-dependent end-of-line terminator character such as a carriage return character. At times, such lines may exceed the capacity of an operating system buffer. This is not critical provided the operating system does not insert such an end-of-file character arbitrarily within the logical stream of data. If it does, the correctness of the picture being displayed may be thwarted. Every attempt should be made to provide such a transparent flow of characters to the terminal. If it cannot be accomplished, the device-dependent routines within GCS must be modified to bracket the inserted end-of-line characters with commands that keep the end-of-line characters from effecting the visual output. Note: input operations do not require an unlimited line length.

V. Phased Implementation Sequence.

The following steps describe the sequence of actions necessary to bring 3D GCS up on a new computer system. It is assumed that 3D GCS source is available on tape in the form of card images.

1) Read the source tape on to disk in a form in which the source can be manipulated using the host-computer standard source program library maintenance procedures. Note that for proper organization later into object libraries, it may be necessary to store the device-independent source programs in one file and each set of device-dependent source programs in separate files. Other computer systems may allow all the source programs to be kept together. If an INCLUDE, COPY, or *CALL capability is available, the Graphics Status Area (GSA) should be placed in a module which can be included and the code in the source programs replaced with COPY's, INCLUDE's, etc.

2) Establish the computer-dependent initializations for the Graphics Status Area. Then edit these values into the appropriate places in the source code where the GSA initialization statements are located.

3) Implement the computer-dependent routines described in this document. These should be thoroughly tested before attempting to use them within GCS. Once debugged, the source code should be placed with the other device-independent source programs.

4) Compile all device-independent routines into one object library in a form which can be searched by the loader or linkage editor. If compilation errors are detected, they should be corrected and this step repeated until no compilation errors occur.

5) Compile all device-dependent routines for the device upon which development will occur into another object library in a form which can be searched by the loader or linkage editor. If compilation errors are detected, they should be corrected and this step repeated until no compilation errors occur. It is highly recommended that the development device selected be one for which a set of GCS device-dependent routines already exists even if this must be an alphanumeric device (alphanumeric terminal or line printer).

6) Write a test program which invokes the basic GCS functions. A typical such program might be as follows:

```
CALL USTART
CALL USET("PERCENT UNITS")
CALL UDAREA(0.,100.,0.,100.)
CALL UMOVE(0.,0.)
CALL UPEN(100.,100.)
CALL UPRINT(50.,50.,"TEXT ")
CALL UEND
STOP
END
```

This program will draw a diagonal line from the lower left corner of the display surface to the upper right corner of the display surface and will display the word "TEXT" at a location where the lower left corner of the first character position is at the center of the display surface with the characters extending to the right. Continue testing GCS until it is apparent that GCS is functioning satisfactorily for the development device. Note that to use GCS, it will be necessary to insure that the BLOCK DATA subroutine for the display device selected be loaded since it is the subroutine which initializes the GSA.

7) Repeat steps 5 & 6 for each display device to be supported. Read the 3D GCS Device-Dependent Implementation Guidelines if a device not already supported by GCS is to be used.

8) Design and implement a control card procedure or other mechanism which makes device selection convenient for the user. Typically, this can be accomplished by passing the name of the device to the control card procedure as a parameter. The control card procedure will then cause the appropriate control cards to be processed so that the appropriate device-dependent library will be used by the linkage editor or loader. While this step is optional, it is strongly recommended since it greatly facilitates using GCS.