CGTM No. 166
April 1975

AN INTRODUCTION TO THE SLAC UNIFIED GRAPHICS SYSTEM

ROBERT C. BEACH
COMPUTATION RESEARCH GROUP
STANFORD LINEAR ACCELERATOR CENTER
STANFORD, CALIFORNIA 94305

TABLE OF CONTENTS

The textual part of this document was produced using the FORMAT Text Processing Program. The figure was drawn on the 10 inch CALCOMP Drum Plotter using the SLAC Unified Graphics System and this drawing was then photographically reduced.

The SLAC Unified Graphics System is a collection of PL/1 procedures and FORTRAN subroutines which may be used to program for the graphic devices which can be driven from the central computing facility at SLAC.

This document is a basic introduction to the system and is intended primarily for those people who have no previous programming experience with the SLAC Unified Graphics System. This manual contains informal descriptions of the FORTRAN subroutines with the differences between the FORTRAN version and the PL/1 version noted. Before a programmer will be able to actually prepare programs using this system, he will also have to study parts of the basic reference manuals [BEA-1, BEA-2] for this system. In addition to these reference manuals, another document [BEA-3] is also available which describes how the system works. A programmer will not normally need this last document.

SLAC has always had a number of graphic devices accessible from the central computing facility. In general, the programmer has had a set of subroutines available for using each of these devices. One of the problems has been that the subroutines available for one device have been totally different from those of another device. Thus, changing from one graphic device to another required a programmer to learn a new set of subroutines in addition to re-programming the problem. Part of the problem has always been that two different graphic devices often have very little in common.

The SLAC Unified Graphics System is an attempt to eliminate many of these difficulties for a large class of graphic applications. The programmer will use exactly the same set of subroutines to create a picture for one graphic device as for another. Thus, if a programmer is using more than one graphic device, the number of different subroutines that must be kept in mind is considerably reduced. In addition, the problem of changing a program from one graphic device to another is greatly simplified. At the same time the programmer will be able to take advantage of any of the special features of a specific display device. In taking advantage of these special features the programmer, of course, will have limited the possibilities of changing to another graphic device at some future time.

SECTION 1.1: THE OPTIONS ARGUMENT

The first argument in almost all subroutines is a character string which is used to specify information to the system which may be optional or device-dependent. The character string may

contain any number of items separated by commas. Each item may be either (1) a simple string of characters or (2) a string of characters terminated by an equals sign, followed by a number, character string, or bit string. In FORTRAN, this options argument must be terminated by an asterisk. Thus the line:

'SMAL,ANGLE=5.3,DDNAME=PQRS,LITES=01010*'

is an example of the kinds of strings which are valid. If specific items of information are not supplied, default values will be assumed; if invalid information is supplied, it will be ignored.

SECTION 1.2: THE ERROR PROCESSING SUB-SYSTEM

This system contains a flexible scheme to report errors to the programmer. Any errors detected by these subroutines are classified into one of four severity levels. The default actions corresponding to these levels are:

1. Set error indicators.
2. Set error indicators and print a message.
3. Print a message and terminate the program without a memory dump.
4. Print a message and terminate the program with a memory dump.

The error indicators, for non-terminal errors, may be checked by the programmer at execution time to determine what the error was, and possibly correct the problem. The error message contains the name of the subroutine detecting the error and an index number. These index numbers are described in the basic reference manuals.

In addition, a programmer can also supply an error processing subroutine which will obtain control before the default action occurs. This error processor can do almost anything to try to recover from the error, including calling other subroutines in this system. The use of an error processing subroutine is also described in the basic reference system.

This section will give a basic description of how a programmer creates pictures in this system. Pictures are created in exactly the same manner for all of the graphic devices supported by this system.

SECTION 2.1: GRAPHIC ELEMENTS

In this system, a programmer creates descriptions of pictures in the following manner:

1. An array of integers is defined in the program.
2. The array is initialized by calling subroutine UGEINT.
3. Other subroutines, UGEPNT, UGELIN, UGETXT, etc., are called to pack the descriptions of points, lines, and text material into the array.

The information which has been packed into an array in this manner is called a "graphic element" in this system. A later section will describe a subroutine, UGEPUT, which can be used to transmit a graphic element to a graphic device. A complete picture may consist of many graphic elements. In essence, a graphic element is a device-independent description of a partial picture.

Thus the creation of a picture in this system is basically a two step process. In the first step, the programmer packs a device-independent description of a partial picture into an array. The second step comes when the information in this array is converted to device orders and transmitted to the device.

SECTION 2.2: THE PROGRAMMER COORDINATE SYSTEM

The programmer specifies the positions of points, end points of lines, and centers of characters by giving floating point X and Y coordinates. The default drawing space (the "programmer coordinate system") for each graphic device is a square area with coordinates of (0.0,0.0) at its lower left hand corner and coordinates of (1.0,1.0) at its upper right hand corner. The picture is scissored at the outer boundaries of the display. A subroutine is provided which allows the programmer to change these scaling and scissoring limits but it is usually better, especially for the beginning user, to use the default values. Other default values in the system assume that these values have not been changed.

SECTION 2.3: PICTURE GENERATION

An example of FORTRAN statements which define and initialize a graphic element are:

```
      INTEGER*4 ELEMNT(500)
      Call UGEINT('CLEAR*',ELEMNT,500)
```

The first argument of UGEINT may be used to perform other operations which will not be described here. The third argument must specify the dimension of the array. The PL/1 version of the system does not require this third argument because the dimension of an array is passed with the array.

To add the description of a point at (0.1,0.2) to the graphic element, the programmer may write:

```
      CALL UGEPNT('BRIT*',0.1,0.2,ELEMNT)
```

When the element is finally transmitted to a graphic device, the point will be in the BRIT mode if the device has intensity level control; if the device does not have intensity level control, this item is ignored. This system supports the four intensity levels, VDIM, DIMM, BRIT, and VBRT which stand for "very dim", "dim", "bright", and "very bright".

A simple way to specify lines is to give the end points, one at a time, to UGELIN along with a "blanking bit" which tells the device whether or not to draw when moving to the point. For example, to draw a line segment from (0.3,0.4) to (0.5,0.6), the programmer should write:

```
      CALL UGELIN('*',0.3,0.4,0,ELEMNT)
      CALL UGELIN('*',0.5,0.6,1,ELEMNT)
```

The fourth argument is the blanking bit; a zero means a blank movement and a one means a drawn line. In PL/1 the blanking bit argument is given as a bit string of length one. If, immediately after these two lines of code, the programmer writes:

```
      CALL UGELIN('*',0.7,0.8,1,ELEMNT)
```

then another line segment will be drawn from (0.5,0.6) to (0.7,0.8).

A programmer may add the description of character data to a graphic element by calling subroutine UGETXT. The specification of text data is more complex than that of lines and points because of the great diversity of graphic devices in this area. Some graphic devices do not have hardware character generators, while some others have character generators of great versatility. Consider the following statement:

```
      CALL UGETXT('SMAL*',0.1,0.2,'A',1,ELEMENT)
```

In this statement, the first argument says that the characters are to be drawn by the hardware character generator in the small size. If the graphic device does not have a character generator, this data will not be added to the picture. This system supports four hardware character sizes, VSML, SMAL, LARG, and VLRG, which stand for "very small", "small", "large", and "very large". The second and third argument give the X and Y coordinates of the center of the first of the characters which are to be displayed. The fourth and fifth arguments give the character string and its length. In the PL/1 version, the length is not passed as a separate argument. Now consider the statements:

```
      CALL UGETXT('SPACING=0.02*',0.3,0.4,'AB',2,ELEMNT)
      CALL UGETXT('XSPACING=0.02*',0.5,0.6,'ABC',3,ELEMNT)
```

In the first statement, the SPACING item says that the characters are to be produced by drawing the characters with short line segments (called "strokes"). The value, 0.02 in this case, is the spacing between the centers of consecutive characters. In the second statement, the XSPACING item gives a suggested character spacing. This system will use the hardware character generator if the device has one and if it can match the spacing value reasonably close; otherwise, the stroke generator is used. The first argument can also be used to specify other options like intensity levels and the orientation of the characters. The orientation item is ANGLE=<VALUE> where the value is the angle, in degrees, that the characters make with the horizontal in a counter-clockwise direction.

Multiple versions of the character stroke generator are available. The default version produces all of the characters on the IBM 029 keypunch. An alternate version includes upper and lower case Roman and Greek letters, a large number of special characters, and a flexible subscripting and superscripting ability.

Now consider the following statements:

```
      INTEGER*4 ELEMNT(150)
      CALL UGEINT('CLEAR*',ELEMNT,150)
      DO 102 I=1,6
        DO 101 J=1,6
          X=0.05*FLOAT(I)+0.6
          Y=0.05*FLOAT(J)+0.6
          CALL UGEPNT('*',X,Y,ELEMNT)
 101    CONTINUE
 102  CONTINUE
      DO 103 I=1,11
        A=0.01745329*(9.0*FLOAT(I-1))
        X=0.5*COS(A)+0.1
        Y=0.5*SIN(A)+0.1
        CALL UGELIN('*',0.1,0.1,0,ELEMNT)
        CALL UGELIN('*',X,Y,1,ELEMNT)
 103  CONTINUE
      CALL UGETXT('SPACING=0.1*',0.2,0.8,'TEST',4,ELEMNT)
      CALL UGETXT('SPACING=0.1,ANGLE=45*',0.7,0.2,
     X  'TILT',4,ELEMNT)
```

Figure 2.3.1 shows the picture which will be produced by these statements.

In addition to subroutines UGEPNT and UGELIN which supply a single point or single line end point at a time, there are subroutines which supply a group of points or line end points at once. Suppose the arrays XARRAY and YARRAY contain the X and Y coordinates of points. Then the statement:

```
      CALL UGEPTS('*',XARRAY,YARRAY,NPTS,ELEMENT)
```

will add the NPTS points in the arrays to the element. The statement:

```
      CALL UGEPTS('*',XARRAY,YARRAY,NPTS,1,1,ELEMENT)
```

will draw lines from the first point to the second point, from the second point to the third, etc. The fifth and sixth

arguments in the example, force line segment to be drawn between each consecutive pair of points. Other combinations are possible which can, for example, cause every third line to be blanked. In the PL/1 version of the system, these two arguments are combined into a single bit string argument. The PL/1 argument corresponding to the fifth and sixth argument in the example is '1'B.

The SLAC Unified Graphics System supplies a number of additional subroutines for describing pictures. There is, for example, a subroutine called UGAXIS which generates an axis with tic marks and labels with a single call. These additional subroutines will not be described in this document.

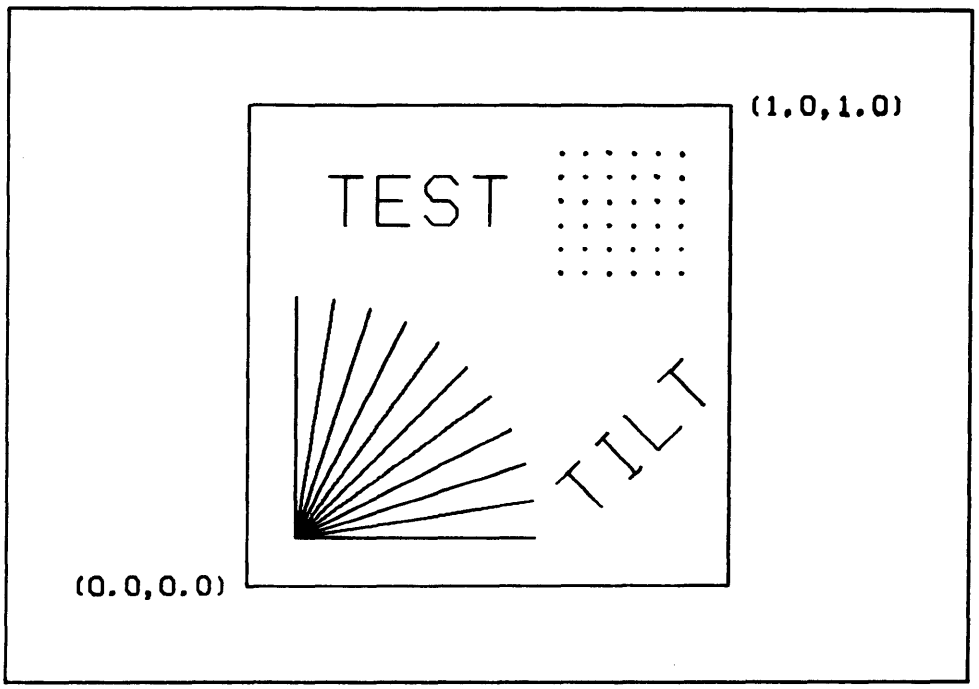

Figure 2.3.1: Examples of UGEPNT, UGELIN,and UGETXT.

The principal advantage of having the programmer provide an array to hold the picture description data is that it provides an explicit receptacle for the device independent form of the picture. On first reading, it may seem that this scheme has introduced one more problem for the programmer, namely assuring that the array does not overflow. However, the programmer can use the error processing sub-system to identify this problem and cause the picture data to be transmitted to a graphic device whenever the array fills up. This is described in the basic reference manuals.

SECTION 2.4: EDGE OF SCREEN PROBLEMS

There are no problems with drawing points, lines, or characters produced by the stroke generator near the screen boundary. These items will always be correctly positioned and will always be correctly scissored if they extend off screen. However, some problems can occur at the screen edges with the hardware character generators.

For example, suppose a programmer tries to plot a character at the point (0.0,0.0). On the IBM 2250, the hardware can plot a character centered on any addressable point. Thus this character will appear in the picture and will partially extend outside the area where lines and points may be plotted. However on the TEKTRONIX 4013, the hardware can plot a character only if its lower left corner is at an addressable point. In this case, the character will not appear in the picture.

Because of these differences in hardware character generators, it is suggested that the programmer not try to plot hardware generated characters too close to the edge of the screen. No character should be placed within one-half character height of the top or bottom of the screen or within one-half character width of the sides.

The non-interactive graphic devices which are now supported by this system are:

1. The CALCOMP Drum Plotters (Models 564/565)... These are off-line devices which can only plot lines and points; no hardware character generator is provided. One device uses 10 inch paper and the other uses 29 inch paper.
2. The CALCOMP Microfilm Plotter (Model 1675)... An off-line device which can produce 16mm unsprocketed film or microfiche.
3. The VERSATEC Electrostatic Plotter (Model 1100A)... This device is connected to a NOVA computer at SLAC. This system uses it as an off-line device. In the graphics mode, no hardware generator is available.
4. The TEKTRONIX 4013 Display Terminal... When used in this mode, the output from a program is a partitioned data set with each picture as a separate member. Under the WYLBUR Text Editing System, a picture may be brought into the active file and drawn on the screen. Some of these devices are connected to hard-copy units.
5. The DTC-300 Typewriter Terminal... These devices are used in the same manner that the TEKTRONIX 4013 units are used.
6. A pseudo-device is also available which provides a convenient way to save pictures in a device-independent form on a disk.

This section will describe the basic subroutines which enable a programmer to control these devices. However, some references to interactive devices will be made in this section.

SECTION 3.1: INITIALIZATION AND TERMINATION

Subroutine UGOPEN is used to initialize a graphic device. It should be called before any other subroutine in this package is called. It is when the program calls UGOPEN that this system first knows what device the program intends to use. The call to UGOPEN, therefore, causes the device dependent portion of the system to be brought into memory. For example, the statement:

```
CALL UGOPEN('CAL/10D*',98)
```

specifies that the graphic device is the 10 inch CALCOMP Drum Plotter. In this case, the program will assume that the JCL contains a DD card with a DDNAME of PLOTTAPE and the plotter commands will be written to this data set. The second argument in UGOPEN is the device identification; it is only important when a program is using more than one graphic device. The use of multiple graphic devices will not be discussed in this document. The options list can also be used to supply a number of other

parameters, some of them device dependent, to the system. Consider the statement:

```
      CALL UGOPEN('IBM2250,DDNAME=GDEV,MAXENUM=50*',99)
```

This statement says that the graphic device to be used is an IBM 2250; the DD card which specifies which specific unit is to be used has a DDNAME of GDEV (instead of the default SCOPE50); and the maximum number of elements which can be contained in any one picture is 50 (instead of the default 25). This limit on the number of elements in a picture applies only to refresh display devices.

Generally, one of the final things a program should do is to call subroutine UGCLOS. Calling this subroutine signals the system that the program intends to make no more use of the graphic device. A typical call is:

```
      CALL UGCLOS(99)
```

The argument is the identification of the graphic device to be terminated.

SECTION 3.2: DISPLAY DEVICE CONTROL

Subroutine UGPICT is used to control the picture on a graphic device. The only operation which applies to non-interactive devices is the picture clearing operation:

```
      CALL UGPICT('CLEAR*',0)
```

This operation always signals that a new picture is being started; on a mechanical device like the CALCOMP Drum Plotters, the pen moves over to a fresh drawing area; on a device like an IBM 2250, the screen is cleared.

The subroutine which adds a graphic element to the current picture is UGEPUT. The subroutine call:

```
      CALL UGEPUT('*',0,ELEMNT)
```

will take the graphic data in the array ELEMNT, transform it to device orders and transmit it to the device. After subroutine UGEPUT has been called, the array ELEMENT may be re-used.

The second arguments in both of these subroutines has meaning only for interactive devices. For non-interactive devices, they should be zero.

The interactive graphic devices which are now supported by this system are:

1. The IBM 2250 Display Console... These units consist of a refresh display scope, keyboard, and light pen. There are three of them at SLAC.
2. The GIF-IDIIOM Display Console... This device consists of a refresh display scope, keyboard, light pen, function button array, and a number of other experimental devices.
3. The TEKTRONIX 4013 Display Terminal... These units consist of a storage scope and a keyboard. There are 20 of them at SLAC.

This section will describe the additional subroutines which are necessary to control interactive devices.

SECTION 4.1: DISPLAY DEVICE CONTROL (REVISITED)

Display device control is more complex on interactive devices, especially for refresh display devices. On such devices, each element may be manipulated individually. For example, the statement:

```
CALL UGEPUT('LPEN*',7,ELEMNT)
```

gives the element an identification of 7 and says that the element is to be detectable with the light pen. This numeric identification is the name by which the programmer will refer to the element when the element is to be manipulated. If a second element with the identification of 7 is transmitted and the device is a refresh display device, then it will replace the first. Elements transmitted with zero identifications are considered to be unidentified. A picture can contain many unidentified elements but they cannot be manipulated individually.

The use of subroutine UGPICT has been considerably extended for interactive devices also. For instance, the statement:

```
CALL UGPICT('NDET,OMIT*',7)
```

will change element 7 so that it is no longer light pen detectable, and put it into the "omit" state. An element in the omit state will not appear on the display device. It can be put into the "include" state by the statement:

```
CALL UGPICT('INCL*',7)
```

This statement causes the element to reappear on the screen. Thus, this include-omit switching can be used to temporarily blank out an element and then restore it without re-generating it with subroutines UGEPNT, etc. and re-transmitting it with UGEPUT. Finally, the statement:

```
CALL UGPICT('CLEAR*',7)
```

will delete element 7 from the picture.

The element manipulation that has been described here only applies to refresh display scopes; most storage scopes have an inherent hardware limitation which prevents individual elements from being deleted from a picture. On such storage scopes, as on non-interactive devices, the only valid operation is the clear screen operation. On storage devices, the element identification is usually ignored.

Another subroutine, UGCTRL, is provided for interactive devices to perform a number of special operations on the display device. Most of the operations are device dependent but a common operation is:

```
CALL UGCTRL('BEEP*',IARRAY,XARRAY)
```

which causes the audible alarm on the device to sound. IARRAY is a 10 word INTEGER*4 array, and XARRAY is a 2 word REAL*4 array which is used for input and output of parameters for some of the other operations. In PL/1, the second and third arguments are combined into a single structure argument whose declaration is:

```
DECLARE 1 IODATA,
          2 IARRAY(10) FIXED BINARY(15,0),
          2 XARRAY(2) FLOAT BINARY(21);
```

SECTION 4.2: ATTENTION CONTROL

This section describes subroutines which allow the programmer to control attentions from an interactive display device. It is by means of these subroutines that the programmer synchronizes the execution of the program with the actions of the console operator. Some of the items which can generate attentions are the keyboard (on all of the interactive devices) or the light pen (on some devices). If an attention is "enabled", then the console operator can generate this attention and the program can determine that the attention has been generated. If an attention is "disabled", the actions of the console operator will not be communicated to the program. Initially, all attentions are disabled.

To enable the keyboard and the light pen, the programmer may write the statement:

```
CALL UGEATN('KBRD,LPEN*')
```

Keyboard attentions are generated by a variety of keys: on the IBM 2250, the KBRD attention is generated by holding the key labeled ALT down and depressing the key labeled END; on the TEKTRONIX 4013, it is generated by the key labeled RETURN. On the IBM 2250, the LPEN attention is generated by pointing the light pen at a detectable element and depressing the foot pedal.

Subroutine UGRATN may be used to determine if an attention has occurred. The subroutine call:

```
CALL UGRATN(0.0,ATCODE,IARRAY,XARRAY)
```

will check to see if an interrupt has occurred and will set the last three arguments accordingly. ATCODE is an INTEGER*4 variable which will be set to 'NONE', 'KBRD', or 'LPEN' to indicate what has occurred. IARRAY is an INTEGER*4 array with a dimension of 5. The first word will be set to 0, 1, or 2 corresponding to the above three possibilities for ATCODE. For the KBRD attention, no other information is available. However, more information is available for a LPEN attention. IARRAY(2) gives the identification of the element that the console operator was pointing at, while XARRAY(1) and XARRAY(2) give the X and Y coordinates of the item that was pointed to.

Subroutine UGRATN can also be used to wait for a specific length of time for an attention or it can wait indefinitely for an attention. The statements

```
      TIME=30.0
      CALL UGRATN(TIME,ATCODE,IARRAY,XARRAY)
```

will wait until 30 seconds has passed, or until an attention occurs. The unexpired time will be returned in TIME. Finally, the statement

```
      CALL UGRATN(-1.0,ATCODE,IARRAY,XARRAY)
```

will cause the program to wait indefinitely for an attention.

In PL/1, the last three arguments are combined into a structure whose declaration is:

```
      DECLARE 1 ATTND,
                2 ATCODE CHARACTER(4),
                2 IARRAY(5) FIXED BINARY(15,0),
                2 XARRAY(2) FLOAT BINARY(21);
```

The final subroutine for attention control may be used to disable attentions. For example, to disable the LPEN attention, the user should write:

```
      CALL UGDATN('LPEN*')
```

SECTION 4.3: KEYBOARD INPUT CONTROL

A "keyboard input buffer" is a special kind of text element which may be displayed on an interactive device and modified by the console operator. When a keyboard input buffer is displayed, a cursor will appear in it. The console operator may type on the keyboard to insert characters into the buffer at the position marked by the cursor.

A keyboard input buffer is added to the display by calling subroutine UGKPUT. A typical call is:

```
      CALL UGKPUT('SMAL*',0.1,0.2,5,'ABCD',4)
```

This statement adds a keyboard input buffer with an identification of 5 to the display. This buffer may be manipulated by UGPICT, just like any other element. In this example, the cursor will be positioned at the character at the point (0.1,0.2). On refresh devices like the IBM 2250, the characters ABCD will also appear on the screen and the console

operator may change these characters or retype them completely. On this type of device, the programmer may put many input buffers on the screen at one time. On the IBM 2250, the JUMP key will move the cursor from one input buffer to another. Subroutine UGKPUT works a little differently on a storage scope like a TEKTRONIX 4013. On this device, the characters ABCD of the example would not be written on the screen and only one input buffer can be on the screen at one time. In PL/1, the final argument which gives the length of the string is not required.

A keyboard input buffer is read from the screen by subroutine UGKGET. Consider the example:

```
INTEGER*4 STRING(5)
CALL UGKEGT(5,STRING,20,NSTR,ICUR)
```

This example will read the keyboard input buffer with an identification of 5 into the array STRING. The third argument gives the maximum number of characters which can be put in STRING. UGKGET puts the actual number of characters read into NSTR and sets ICUR to the index of the character which contained the cursor. On an IBM 2250, the entire buffer, as given by UGKPUT, is read back, while on the TEKTRONIX 4013, only those characters actually typed will be returned. In PL/1, the second, third, and fourth arguments are combined into a varying length character string.

To summarize this section and the preceeding section, simple interaction between the program and the console operator can be done as follows:

1. The program puts a picture on the screen, adds a keyboard input buffer, and assures that the keyboard is enabled.
2. The program waits for an attention. At this time, the console operator can enter text into the input buffer. When the text has been entered, the console operator generates a KBRD attention.
3. Upon receiving the attention, the program can read the keyboard input buffer and process the data it contains.

Interaction with a light pen would be done in an analogous manner.

This section contains a list of all of the items that have been referenced in this document.

[BEA-1] R. C. Beach, CGTN #142, The SLAC Unified Graphics System: PL/1 Version, Stanford Linear Accelerator Center, Stanford CA 94305

[BEA-2] R. C. Beach, CGTN #143, The SLAC Unified Graphics System: FORTRAN Version, Stanford Linear Accelerator Center, Stanford CA 94305

[BEA-3] R. C. Beach, CGTN #163, The Internal Operation of the SLAC Unified Graphics System, Stanford Linear Accelerator Center, Stanford CA 94305