THE BOLOGNA GRAPHICS LIBRARY

BGL USER'S MANUAL

A. Ficarra, M. Nanni
Istituto di Radioastronomia - CNR

L. Federici, A. Messina
Dipartimento di Astronomia - Bologna University

IRA 89/86 BGL 2/1985

1. INTRODUCTION.

The Bologna Graphics Library (BGL), Version 1.0, is a software environment, whose major component is a library of FORTRAN-callable basic graphics subroutines running only under the VAX/VMS operating system. Its main purpose is to provide users with application programs that generate and manipulate pictures on any graphic peripheral.

The background and the main characteristics of the BGL system are described in the "Presentation" paper (BGL 1/84). We suggest that the user reads that paper before reading this manual, that explaines which functions the BGL system performs and how they are executed. In the manual BGL 3/85 users can find reference informations for the use of the BGL subroutines, whereas in the manual BGLA 4/86 informations can be found on a BGL advanced (BGLA) library which uses the BGL subroutines either to create a graph with axis lines, labels, scale annotations and tick marks or to perform more specific activities as recalling a graphical output previously saved, or drawing an histogram or contour lines.

The main components of the BGL are:

a) the main graphics library;

b) the command language;

c) the basic Calcomp software interface;

d) the terminal management library;

e) the miscellaneous routine library.

Except the Calcomp interface subroutines, that keep their traditional names, all the BGL routines have names composed of

two parts: the first one is the string "BGL_", common to all the BGL routine names, to prevent any ambiguity with the names of other user or local system library routines; the second one is a set of characters either not more than six in the graphics library (usually an acronym obtained from the expression which describes the function performed), or not more than seven in the terminal control library (the first two being "VT").

In the Appendix A, instruction can be found to install the BGL system on a VAX/VMS system.

1.1. THE MAIN GRAPHICS LIBRARY.

The subroutines of the main library perform graphics "primitives", that is, basic graphical functions which can be summarized as follow.

Draw "picture objets", like vectors, any type of polygons, boxes, arcs and circles.

"Shade" each object to a defined horizontal shade line with a uniform gray or colour intensity.

Fill or erase selected areas on devices equipped with "raster" addressability.

Define the portion of the picture that it will be seen ("window") and generate a picture "clipped" within the window.

Define the portion of the graphic device surface on which the picture will be drawn ("viewport")

Change the user coordinate system ("world coordinate"), by providing a new unit of measure ("scaling"), or redefining the system origin ("translation"), or giving the new coordinates of the boundaries of a selected rectangular area ("mapping").

Represent alphanumeric texts and numbers in graph mode.

Manipulate colours and colour look-up tables.

Perform input operations by means of a position "locator", such as a cursor or a crossbar moved by a joistick.

Save a sequence of routine calls (in a coded format) and the associated input argument values into a file on the user reserved disk area.

Execute the graphic operations previously saved in a file through a single call.

Select the physical device where to display the pictures.

Print the image of a terminal screen on paper, if the selected peripheral is equipped with a "hard-copy" device.

Moreover, the main graphics library subroutines allow alphanumeric actions such as those performed by the following subroutines:

BGL_APLDEF : define an alphanumeric area ("scroll"),

BGL_ALPCLR : clear an alphanumeric area,

BGL_CLRASC : clear all the alphanumeric texts on the screen.

These routines can be used on any terminal with any graphic action, since their characteristics are known to the graphics library.

In Sect. 8., a detailed examination of the graphic functions will be done; whereas a list of the names and related functions of the routines of this library can be found in Appendix B.

1.2. THE BGL COMMAND LANGUAGE.

In addition to the library of graphics subroutines, the BGL

system is equipped with a simple Command Language, whose purpose is to allow the user to operate on the system in an easy and quick way, or to execute some tasks involving the use of graphic subroutines.

The access to the Command Language is obtained entering at the user terminal the string "BGL" followed by one (or more) sub-commands and the related parameters invoking specific functions. As the subroutine library, also the Command Language can be easily expanded and could allow the execution of an ever increasing number of graphics utilities.

At the present, the Command Language performs two sets of functions, which are described in the following.

i) System management.

These commands are reserved to privileged users and allow:

$ BGL INSTALL	:	the installation of the BGL;
$ BGL CONFIG	:	the definition of the local system configuration (or reconfiguration) with respect to the peripherals;
$ BGL UPGRADE	:	the BGL software updating.

ii) Procedure execution.

These commands can be entered by any user and set on procedures which allow:

$ BGL LINK	:	the "linking" of the application program to the BGL;
$ BGL ASSIGN	:	the selection of the out-of-line device to be used;
$ BGL RELEASE	:	the management of the graphic output sent

to assigned peripherals;

$ BGL SHOW : the request of informations on the local system configuration;

$ BGL HARD : the out-of-line hard-copy of the device screen on a graphical printer;

An on-line help is also provided, to let the user know action and required parameters of any command. It can be activated entering "?" after the string "BGL". The Fig. 1 visualizes the structure of the commands, an asterisk marks the commands reserved to privileged users.

1.3. THE BASIC CALCOMP SOFTWARE INTERFACE.

The BGL system contains also a library of subroutines with the same names, purposes and calling sequencies as the main subroutines of the basic Calcomp-Versaplot library. These subroutines call the BGL routines and actually work as a software interface between the application program and the BGL system. Therefore, the old user programs containing calls to the basic Calcomp routines can still be runned in the BGL system. Moreover, by means of this interface, also the high-level Calcomp graphics packages that call basic Calcomp routines can be runned in the BGL environment. The Appendix C lists the names and relative functions of the routines contained in this library.

1.4. THE TERMINAL MANAGEMENT LIBRARY.

This library consists of two groups of subroutines. The

BGL COMMANDS (* => SYSTEM MANAGER ONLY)

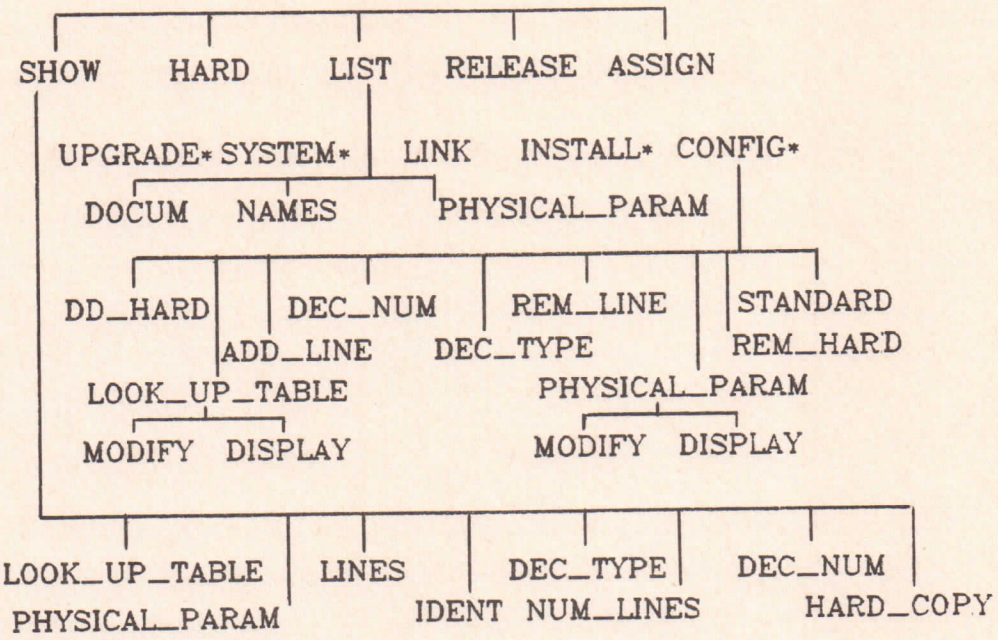

Fig. 1

first group is formed by routines that allow to operate in "full-screen mode" on video terminals of the DEC VT100 family, whereas they do not produce any effect when called at terminals of any other type. With this set of routines it is possible to perform interactively operations such as the splitting of the screen into several areas, or the moving of the alphanumeric cursor to locate the wanted function from a "menu" displayed on the screen, or the writing of "blinking" texts or in "reverse-video" mode etc...

If this set of routines is used to manage the terminal screen in alphanumeric mode, caution is necessary, when a graphic session is performed, since the parameters used are not known to the main graphics library.

The second group consists of routines working on any type of terminal and allows to set or to modify some terminal line characteristics (for instance, it is possible to disable the terminal from receiving broadcast messages).

The Appendix D lists the names and relative functions of the routines of this library.

1.5. THE MISCELLANEOUS ROUTINE LIBRARY.

These routines are mainly used by the graphics library for internal management but they can also be accessed by the user for his applications. Three different functions can be performed, which are the following:

i) Creation and management of files.

The routines of this set manage files mapped in the process virtual memory. Their functions are:

BGL_OPEN_SE	:	generate a new file or access to an existing file;
BGL_CLOSE_SE	:	close a file;
BGL_CLEAR_SE	:	clear the content of a file;
BGL_READ_SE	:	read data from a file;
BGL_WRITE_SE	:	write data in a file.

ii) Management of call sequences.

The routines of this second set operate on the argument list of a call (written in Assembler). Their functions are:

BGL_DEFARG	:	report if an optional argument has been specified or not (it is a logical function);
BGL_TRADDR	:	transfer the control to a routine specified as an argument;
BGL_TRANSF	:	transfer the call sequence (included the optional arguments) from a subroutine to another one specified as an argument.

iii) Various purposes.

The routines of this last set call System Service procedures. Their functions are:

BGL_CPUPAG	:	return the CPU time and the number of page faults;
BGL_TYPE	:	perform in a FORTRAN program the DCL command $ TYPE;
BGL_WAIT	:	set the process in "WAIT" for the indicated time.

2. HOW TO LINK A FORTRAN PROGRAM TO THE BGL.

An application program calling BGL routines needs to be linked to the BGL in order to become executable. All the user addressable BGL routines are in the form of "shareable images" and reside in an apposite "library of shareable images". The full name of the file containing this library is installation dependent. If this name is known (for instance, BGL_DISK:[BGL.EXE]BGLSHR.EXE), the user has simply to include it as an input to the LINK command, with the /LIB qualifier.

Assume, for instance, that the user compiled main program (MYPROG.OBJ) calls some routines, besides BGL, residing in the user file MYSUB.OBJ and in the user object-module library MYLIB.OLB; in this case the string command to be entered is

```
$ LINK MYPROG,MYSUB,MYLIB/LIB,-
$_BGL_DISK:[BGL.EXE]BGLSHR/LIB
```

Otherwise, the BGL Command Language can be used, typing the following command to VAX/VMS:

```
$ BGL LINK MYPROG,MYSUB,MYLIB/LIB
```

Whichever the linking procedure used, the executable image (MYPROG.EXE) can be runned by typing the usual command

```
$ RUN MYPROG
```

However, residing the BGL routines as shareable images, the image program obtained presents two important peculiarities.

First, the content of the shareable images is not copied in

MYPROG.EXE; this file contains only the pointers to the BGL shareable images, and this, of course, saves disk storage space.

Second, the image file MYPROG.EXE accesses the BGL subroutines through "transfer vectors" and is automatically updated at run time, when the BGL software has been updated or when a new graphic device has been made available by a new device driver. Therefore, software improvement and new installed graphic devices are immediately available to the users without any need of relinking their programs.

When many users run different graphics programs, this is a very powerful feature.

3. ON THE GRAPHIC DEVICE MANAGEMENT.

Here and in the following, it is defined as "Graphic Task" (GT) a set of graphics operations performed on a selected device, the so-called "Current Graphic Device" (CGD).

The CGD cannot be changed during the execution of a GT, but it can be redefined when a GT is closed and a new GT is started, if desired. Actually, several GT's can be performed in sequence, when running a single application program. Moreover, it is possible to change the CGD at the start of a new GT, without exiting from the program.

This is the basic feature which characterizes the claimed "device independence" of the BGL: not only an application program can run unchanged on any device, but it is also possible to select or modify the CGD dynamically, according to the results of computations, or to the answers given during the execution of an interactive program.

As an example, assume that a program designed to perform iterative processes gives a plot on the user terminal screen, as a result of each iteration. With the BGL, the program can be structured in such a way that the user controls interactively whether the iteration process have to continue and, when the desired results are obtained, he changes the CGD, in order to produce a final graphic output on a plotter.

The CGD can be changed either, out of an applicative program, by means of the BGL Command Language, in particular with the command

```
$ BGL ASSIGN devnam
```

or, inside a program, by a call to the BGL routine, which is

```
BGL_NEWDEV ('devnam')
```

Of course, the second type of assignment (the dynamical one) prevails on the first one, but is effective only as long as the program is running. On the contrary, the first type of assignment is active either untill the user log-in session is current or untill a new command $ BGL ASSIGN devnam is issued.

The user can ask to the system the actual CGD inserting in a program a call to the routine

```
BGL_DEVINQ (DEVNAM,LENG)
```

which returns the string of the device name (DEVNAM) and its length (LENG).

3.1. THE DEVICE SPECIFICATION.

At the beginning of a running session, the user can learn the names of all the device types whose software interfaces are available, by entering the command

```
$ BGL LIST NAMES
```

Each physical device is identified by a name, that usually should be easy to memorize (as VT125, VERSATEC etc...), and by an order number associated to each installed peripheral of the same type.

The order number is optional when entering a call to BGL

routines and its default value is 1. To know the number of the devices of the same type the user can enter the command

```
$ BGL SHOW devnam LINES
```

which returns a name list of the physical lines connecting the device devnam to the computer.

The user CGD is identified, by default, by the special string "TT", whichever the actual name and the order number of the terminal are. Therefore, the user has not to define a CGD when he wants to perform graphics operations interactively on a terminal screen. Only when a graphic output is requested on a different device, the user must define a new CGD. To return to the interactive terminal the CGD must be reassigned either inserting the string "TT" as a parameter of the BGL ASSIGN command, i.e.

```
$ BGL ASSIGN TT
```

or giving the same string as the argument of the BGL_NEWDEV routine calling sequence, i.e.

```
BGL_NEWDEV ('TT')
```

In this case, note that the assignment of the actual name of the interactive terminal, instead of the string "TT", produces a completely different effect. Namely, the graphic output is considered "off-line" (cfr. Sect. 4.2. for "off-line" GT), and could be obtained on the screen of the interactive terminal only when the LOGOFF command is given.

In a FORTRAN program informations on the physical device can

be obtained by a call to the following routines:

BGL_DEVINQ : the name of the current device;

BGL_DSCINQ : the coordinates of the boundary of the physical viewport (in "standard units", cfr. Sect. 6.);

BGL_DSSINQ : the coordinates of the boundary of the physical viewport (in "screen normalized units", cfr. Sect. 6.).

3.2. THE "NULL" DEVICE.

Up to now we have supposed that a CGD is always assigned to a graphic device. This is not true when it occurs one of the following events:

No CGD was explicitly assigned and the user is not working in an interactive session (a batch job, for example).

No CGD was explicitly assigned and the user terminal is not recognized as a graphic peripheral (all the available graphic devices are declared at the installation or reconfiguration time).

The user has tried to assign a new CGD, but has mispelt the device name, so that the system cannot identify the entered name among those of the installed graphic devices.

The user has intentionally entered a meaningless device name when assigning a new CGD.

In all the above listed cases, the CGD is assigned to the so-called "null" device. If the "null" device has been selected (intentionally or not), the user is notified of this event either by a message displayed at the terminal (when the BGL ASSIGN command has been entered), or by a return code set to a specific value (when the BGL_NEWDEV routine has been called).

If the "null" device is selected, application programs using

BGL routines can be still runned, but no graphic output is actually produced, even if it is still effective all the remainder of graphics operations not directly intended to draw pictures or to perform interactive operations on a physical device.

A possible application of this particular device is illustrated in the following example. If the user is not interested in the immediate examination of a picture, he can select the "null" device to save the graphic output in a file. In this way, the user can draw later the saved picture as many time as he wants, select parts of it, use different scale factors, change the output device etc..., all without repeated and separate runs of the applicative program.

The names and relative functions of the BGL routines which allow the writing on and the reading from User Graphic Files (hereafter UGF) are the following:

BGL_SVINIT : Initialize the specific User Graphic File (UGF). All the subsequent graphic operations will be saved in the UGF.

BGL_SVCLOS : Close the UGF opened with a call to BGL_SVINIT.

BGL_SVMIMA : Compute boundary coordinates of a picture previously saved in a UGF.

BGL_SVEXEC : Display pictures previously saved in a UGF.

It is worth to stress at this point that, if the terminal cannot work at the same time in graphic and alphanumeric modes, a call to the routine BGL_STAMOD must be done to Set the Terminal to Alphanumeric MODe, before any action such as, for instance, a

FORTRAN READ or WRITE instruction. The reverse setting (from alphanumeric to graphic) is performed automatically at any call of a graphics library routine.

4. ON THE "INTERACTIVE" AND "OFF-LINE" GRAPHIC TASKS.

Graphic Tasks operating on a "null" device are very unusual; mostly GT's work on actual physical peripherals, known to the system (namely, their names were entered at the installation or at the reconfiguration time). In these cases the GT's fall into two classes: "interactive" GT's and "off-line" GT's. The type of GT can be tested in the program by a call to

```
BGL_STAMOD (ICOD)
```

which returns different code numbers (ICOD) for each GT type.

4.1. THE "INTERACTIVE" GT.

A GT is defined "interactive" when all the following conditions are satisfied at a time

> No CGD was explicitly assigned or a CGD has been reassigned with the string "TT".
>
> The image running the GT has been activated by an interactive process (namely, the user started the process by entering Username and Password on a terminal), and hence the image owner process is not a batch-job or a detached process.

With an interactive GT the user can perform graphic operations at the terminal in real time: control a picture while it is being build up, exchange information with the program, modify a picture according to the information obtained from the picture itself, etc...

Moreover, the user can perform some specific interactive

operations, such as to move a cursor (cfr. Sect. 7.), to measure the position of some points on the screen (cfr. Sect. 7.), or to define interactively the boundary of a screen area (cfr. Sect. 6. and Sect. 7.), ect...

4.2. THE "OFF-LINE" GT.

A GT is defined as "off-line" if the CGD has been explicitly assigned and its name (different from "TT") matches the name of the graphic devices declared at the installation or reconfiguration time. An "off-line" GT does not display the graphic output immediately, but stores it into an intermediate file, using a symbolic code to indicate the calls to the device-dependent routines and the values of the associated arguments.

All the files created by "off-line" GT's (here and in the following "Off-line Graphic File", OGF) are put into a dedicated directory named

BGL_DISK:[BGL.BAT]

with a name

ABC123456.USR

where ABC is the name of the physical device without the first letter, the number gives hour (12), minutes (34), seconds (56) of the generation time, and USR reports the first three letters of the Username.

The OGF are read by a system process, named BGLGEST, which is

started by the start up procedure and is always running. However, it does not waste any of the system computational resources, since it is normally "hibernate".

At the end of a GT, a mailbox is sent by the "close" routine of the device-independent library and wakes up the BGLGEST process by providing it with the name of the OGF just created. Then, BGLGEST generates a "detached" process to which the OGF is assigned as primary input, with a name derived from the OGF name, i.e.

USRABC12345678

where the symbol have the same meaning of the previous example of OGF name, except the last two digits (78), assigned by an automatic counter which spans the number field from 0 to 99, in sequence.

The "detached" process opens the OGF and reads first a header containing general information, such as the name of the assigned device (CGD). If the CGD is not allocated to another user, all the remainder of the OGF is read, the coded device-dependent routines are performed, and the graphic output is produced on the CGD. Finally, the OGF is purged. Otherwise, if the CGD is allocated to another user, the detached process ends without performing any graphic operation, but the related OGF is not purged.

To assure a sort of "spooling" in this last case, BGLGEST is scheduled to wake up every three minutes, to scan the directory BGL_DISK:[BGL.BAT] and, for every OGF found, to perform the same procedure outlined before, in the case of a wake up by a mailbox.

Therefore, at the end of a GT, a graphic output not immediately produced is queued and searched for every three minutes, to be sent on the requested CGD as soon as possible.

If the BGLGEST process individuates the CGD as a device classified as not automatic at the installation time (for instance, some tipe of plotter requiring an external feeding between a graphic output and the other), it puts the OGF in a HOLD state. In this case, it is the user who commands the release of the output entering

```
$ BGL RELEASE devnam
```

when the device devnam is ready. Furthermore, the user can also stop its own "detached" process, created by BGLGEST, entering, for instance

```
$ STOP USRABC12345678
```

If this happens, the user should also recall to purge the corresponding OGF (ABC123456.USR, in the example given).

5. WHY THE BGL IS "USER FRIENDLY".

The "user friendliness" of the BGL systems consists mainly in letting the system do as many things as possible, and in requiring the user to perform only actions strictly connected with their application problems. Therefore, many functions that the system needs to set up in order to operate properly, but that do not concern directly the user, are performed automatically. In the following all the operations and parameter values which are given by default and a particular system of error control related to the "return code number" are commented.

5.1. THE INITIALIZATION PROCEDURE.

The initialization of a GT is performed automatically as the first BGL routine call is executed in a running program, even if a previous GT in the same program has been closed. The initialization procedure performs mainly the following function, in sequence.

Set the system parameters to their initial values (system state-defining parameters to internal values, graphic attribute and coordinate transformation coefficients to initial default values, etc...).

Identify the GT type ("off-line", "interactive") and the CGD.

If the CGD is not the "null" device, select the appropriate device-dependent library and built up a table of address to allow the immediate translation of any subsequent call to a device-independent routine into the corresponding device-dependent call.

If the GT is "interactive", save the normal working characteristic of the terminal, in order to restore them

when the GT is closed; set all the device graphic initial conditions; set the terminal in graphic mode.

If the GT is "off-line", generate and open an OGF; write a header with general information about the GT and the selected graphic device.

5.2. THE CLOSURE PROCEDURE.

The closure procedure too is automatically performed when the program is either at the end or aborted, as a consequence of an error or of a CNTRL Y display command. In these cases, the following operations are performed.

Whichever the GT type is, check if a user file for storing graphic output is open and, if it is, save the last graphic data and close the file.

If the GT is "interactive", complete the last graphic operation; reset the working characteristics of the terminal to their normal condition.

If the GT is "off-line", write the last data, if any, on the OGF; close the OGF; send a mailbox to the BGLGEST system process (cfr. Sect. 4.2.).

If necessary, the user can control the closure procedure by a call to routines which are:

BGL_CLOSE : close the GT leaving the device unchanged;

BGL_NEWDEV : close the GT assigning a new device for the new GT.

5.3. THE DEFAULT SETTINGS.

All the BGL routines involve a large set of parameters which describe the operating state of the system (coordinates, storing

files, graphics attribute, etc...) and which have not to be changed at any time a GT is performed. Therefore, all these parameters are set by the system to consistent standard values, i.e. the default values or defaults, concisely. These defaults are of two types, "initial" defaults and "routine argument" defaults.

The "initial" defaults are the parameter values which are set by the initialization procedure (cfr. Sect. 5.1.) and which remain in effect untill explicitly changed by the user. These values are either device independent (f.i., the initial units of the application coordinate system, cfr. Sect. 6.), or device dependent (f.i., the colour to be used in drawing objects).

The default values changed by the user with a call to BGL routines become "current" default values and are effective for all the subsequent calls to the function using that parameter.

The "routine defaults" are the values assumed by optional argument of some BGL routine. If not otherwise specified, the assumed values coincide with the "current" default values defined by the last call to the corresponding default setting routine, or with the "initial" defaults, if the user has not modified it in the program. The following list makes clear how to default an argument value in a subroutine.

```
CALL BGL_SUB (A,B,C)          no argument defaulted
CALL BGL_SUB (,B,C)            argument A defaulted
CALL BGL_SUB (A,,C)            argument B defaulted
CALL BGL_SUB (A)         arguments B and C defaulted
CALL BGL_SUB                  all arguments defaulted
```

Since VAX/VMS FORTRAN does not allow to default string arguments, the user should pass the "blank string" (' '), to obtain string default values.

5.4 THE USER ERRORS AND THE RETURN CODE NUMBERS.

A great care has been applied to test all the BGL software components in order to make the system error free. Of course, it cannot be excluded that errors may still occur and that changes may be needed to improve both the software structure and the functional efficiency of the system, but it is a task of the system manager to correct the system software updating BGL, if necessary.

The most frequent source of errors is the user indeed, who usually supplies wrong argument values to the BGL routines. In this case, it was decided to avoid a complete "error handling" facility, as in some more complex graphics systems. Therefore, except in a few desperate cases, the error condition is not notified to the user and no error message is returned. Rather, the system itself either ignores the call, setting initial default values, or takes some corrective actions to avoid a fatal error with a subsequent abort of the program.

If the user, for example, has tried to set a screen area which is partially outside the physical available surface of the device, the system corrects, in the internal BGL memory space, the values given by the user and sets the area properly.

The user is made aware of the result of any call to BGL routines by a system of "return code numbers", which are

arguments of the BGL routines, always optional and always the last in the sequence of the subroutine arguments. The list of the possible "return code numbers" and their meaning is reported in the reference manual BGL 3/85. The "return code number" is omitted only when it is really impossible to make an error.

The same "return code number" system is also used to report the state of the system, if required.

6. ON THE COORDINATE SYSTEMS AND TRANSFORMATIONS.

The length units, the scale factors and the mapping of the user graphics onto a device screen are the main problems to face, when the development of a graphics software is undertaken. In the BGL system these problems have been solved taking into account the advantages of different main graphics softwares actually in use and the development of modern video terminals, which have determined the introduction of the so-called "virtual graphics".

To reach this result the three following worlds have been individuated.

The science universe, where the user problem is set ("application" plane), with its own cartesian and orthogonal coordinate system ("application" system) and its own length units ("application" units).

The wonderland universe, where the graphic output is generated ("virtual" plane), with its own cartesian and orthogonal coordinate system ("virtual" system), with its own length units in centimetre ("standard" units), and unlimited in extension.

The physical universe, where the graphic output is visualized ("device" plane), with its own cartesian and orthogonal coordinate system ("device" system), and with its own length units either in centimetre ("standard" units) or in Screen Normalized units ("SN" units).

Moreover the following two linear trasformations have been defined.

The transformation from the "application" system to the "virtual" system (Application TRansformation or ATR).

The transformation from the "virtual" system to the "device" system (Device TRansformation or DTR).

With these premises, two distinct steps correspond to the production of a graphic output: a "generation" phase, with the graphic output on the "virtual" plane, and a "visualization" phase, with the graphic output on the "device" plane. Usually, the second step follows automatically the first and the user cannot detect any difference between the two. Only in two cases it is the distinction real:

when the graphic output generated in a GT is saved on a UGF from a "null" device;

when a graphic output is obtained starting from a previous UGF.

In the first case, only the "generation" phase is present, in the second one only the "visualization" phase is performed.

6.1. THE "GENERATION" PHASE.

It is assumed that the user has defined his "application" system and units (luminosity versus time, adimensional units versus adimensional units, etc...). The point coordinates that the user enters as arguments of the BGL subroutines are almost always in "application" units.

The BGL routines perform an ATR; namely, transform linearly the "application" system with "application" units in the "virtual" system with "standard" coordinates. The coefficients of the transformation are memorized in an internal area and can be used, on request. With this transformation the graphic output is generated on the unlimited "virtual" plane, and all the

graphic objects rapresented in it have dimensions in "standard" units (centimetre).

At the start of a GT, the default ATR is unitary, i.e. the "application" system coincides with the "virtual" system. However, the user can define an ATR in different ways by means of the following routines:

BGL_ATRSET : set a new ATR;

BGL_ATRMAP : set new coordinates of a rectangular area in the "application" system, starting from the current "application" coordinates;

BGL_ATRDEF : return to the default "virtual" system;

BGL_ATRINQ : request to the system the current parameters of the actual ATR (f.i., in order to save the current transformation coefficient and to restore them later, via BGL_ATRSET);

It is also possible to define a "clipping" area (i.e., an area outside of which no graphic output is generated):

BGL_CLPSET : define an area in the "virtual" plane outside of which no graphic output is generated ("clipping" area);

BGL_CLPOFF : clear a defined "clipping" area.

The "clipping" area is not a "window" and does not coincide with thw area defined to set an ATR.

It should be noted that the "clipping" area is not defined at the start of a GT and that it is the graphic output in the "virtual" plane which is saved om the UGF, on request.

6.2. THE "VISUALIZATION" PHASE.

Once the graphic output is assumed as generated on the "virtual" plane, it is possible to visualize the operations performed by the BGL system, figuring all the device screen, or part of it, as a lens through which the "virtual" plane could be scanned.

It is now easy to understand that for the "visualization" phase it is necessary to individuate:

the area of the "virtual" plane to be visualized ("window"),

the area of the "device" plane where to visualize the graphic output ("viewport"),

the coefficient of the DTR.

Given the "window", it is sufficient to give any one of the other two items in order to fix the third one.

The BGL system assigns to "window", "viewport" and DTR default parameter values, which are determined at the installation or reconfiguration time, for any declared device. For instance, the default "viewport" is defined, in "SN" units, by the point coordinates (0,0) and (1,1), the default DTR is defined by the scale factors for x and y axes introduced at the device installation or reconfiguration time, the default "window" is determined from the default "viewport" via the default DTR. It should be clear from this example that the DTR is not necessarily unitary , since the scale factor is usually 1 for plotter type devices and 0.5 for videoterminals; therefore the initial "window" has a double area with respect to the device screen, in terminals of the video type.

It should be noted that in standard videoterminals the

default viewport coincides with the full screen while in devices of the plotter type it is possible to define a viewport larger than the default one, up to a maximum size defined at the installation or reconfiguration time. The user can know the maximum allowed viewport size entering the subroutine BGL_DSZINQ.

To set a "window", the user can enter the following routines which perform the specified actions:

BGL_WINSET : set coordinate values (in "application" units) for a new "window";

BGL_WINDEF : reset the default "window", which is coincident with an area of the physical device and has the origin coincident with the bottom left corner of the physical device;

BGL_WININQ : return the current "window" coordinates (in "application" units).

To preserve the BGL user friendly, the coordinates for the selection of the "window" on the "virtual" plane are entered in "application" units.

To set a "viewport", the user can enter the following routines which perform the specified actions:

BGL_VPCSET : set coordinate values (in "standard" units) for a new "viewport";

BGL_VPSSET : set coordinate values (in "screen normalized" units) for a new "viewport";

BGL_VPTDEF : reset the default viewport.

BGL_VPCINQ : return the current "viewport" coordinates (in "standard" units);

BGL_VPSINQ : return the current "viewport" coordinates (in "screen normalized" units);

The double unit choice for the "device" system is in function of the user purposes. However, since the correspondence between "SN" and "standard" units in the "device" system is device-dependent, the routine BGL_DSZINQ is provided, which returns the physical device size (in cm) corresponding to one SN unit.

If required, the user can enter, instead of the "viewport" defining routines, the following routines, which define a "device" transformation and hence set the "viewport" coordinates, given a "window":

BGL_DTRSET : set the specified transformation coefficients and define the new viewport by applying the new coefficients on the current window;

BGL_DTRDEF : reset the default transformation coefficients and define the new viewport by applying the new coefficents on the current window.

BGL_DTRINQ : return the coefficients of the current "device" transformation.

Moreover, three further peculiar performances leaving unchanged the current DTR are worth mentioning, which are the following:

BGL_WINWCT : set new "window" and determine new "viewport" with the old DTR;

BGL_VPSWCT : set new "viewport" (in "SN" units) and determine new "window" with the old DTR;

BGL_VPCWCT : set new "viewport" (in "standard" units) and determine new "window" with the old DTR;

Whichever the choice is, the user has a further possibility: he can decide whether the "window" has to be mapped onto the

"viewport", with device clipping area coincident with the mapped area, or if, on the contrary, only the origin of the "window" has to correspond to the origin of the "viewport". The user has simply to enter coordinate values of two opposite corners or of one only corner in the "window" and/or "viewport" setting routine, to select mapping or translation, respectively.

In both cases, the user can request a return code number (cfr. Sect. 5.4), to know whether the defined viewport is partially or completely out of the device screen. Of course, the clipping is performed only on the device screen surface which intersects the defined or computed "viewport".

Fig. 2, Fig. 3 and Fig. 4 summarize the system performances in the "visualization" phase. Downward and upward arrows indicate "set" and "inquire" statements, respectively.

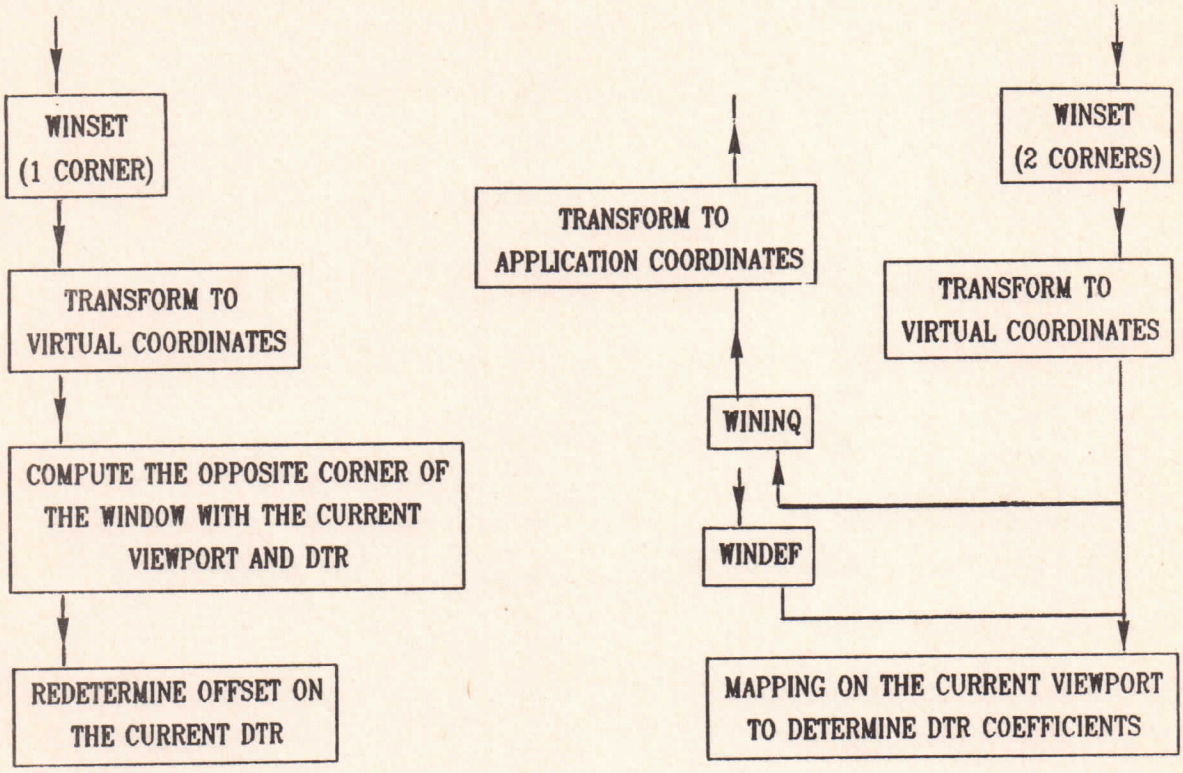
WINSET
(1 CORNER)
TRANSFORM TO
VIRTUAL COORDINATES
COMPUTE THE OPPOSITE CORNER OF
THE WINDOW WITH THE CURRENT
VIEWPORT AND DTR
REDETERMINE OFFSET ON
THE CURRENT DTR
TRANSFORM TO
APPLICATION COORDINATES
WININQ
WINDEF
WINSET
(2 CORNERS)
TRANSFORM TO
VIRTUAL COORDINATES
MAPPING ON THE CURRENT VIEWPORT
TO DETERMINE DTR COEFFICIENTS

Fig. 2

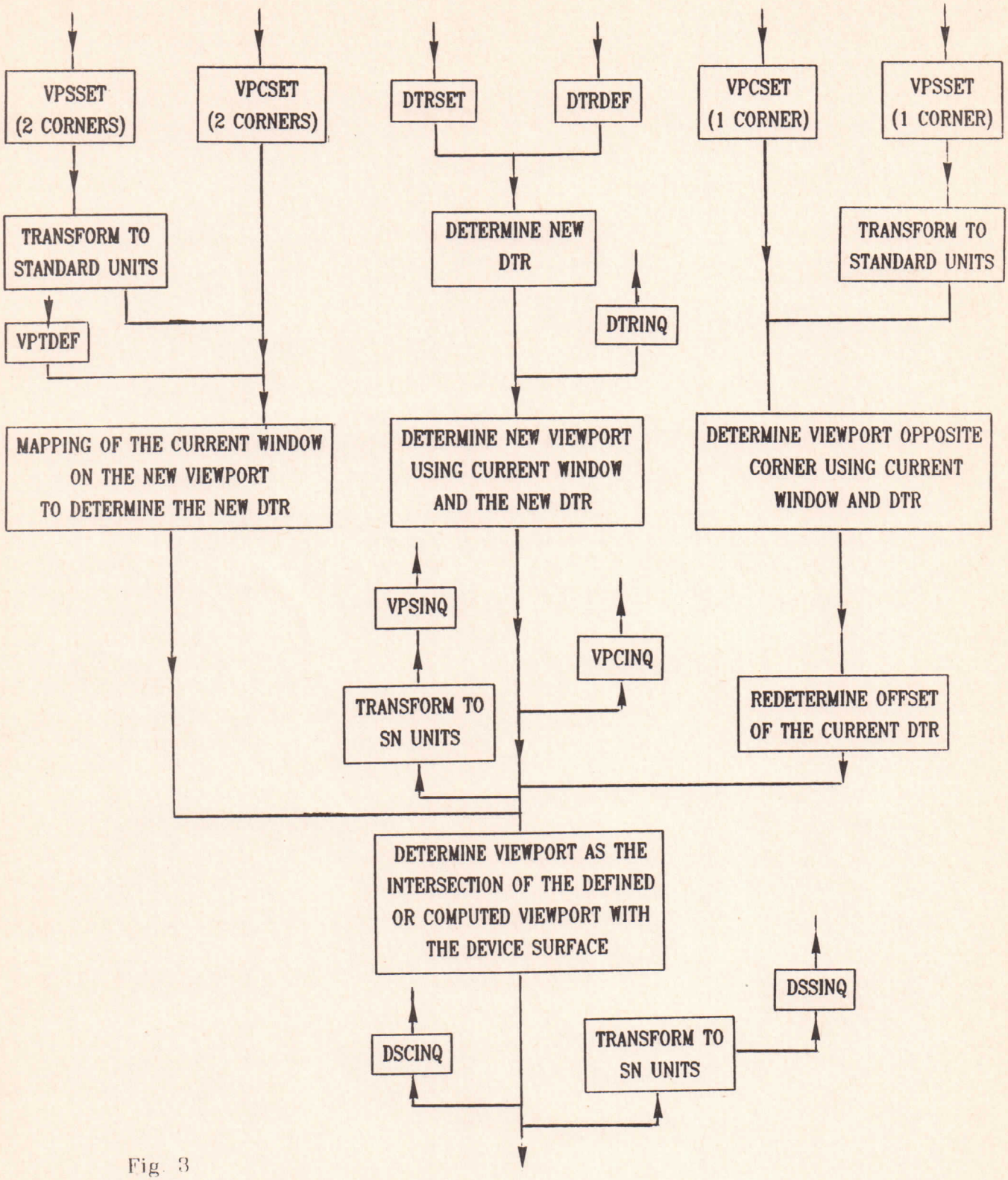
VPSSET
(2 CORNERS)
VPCSET
(2 CORNERS)
DTRSET
DTRDEF
VPCSET
(1 CORNER)
VPSSET
(1 CORNER)
TRANSFORM TO
STANDARD UNITS
DETERMINE NEW
DTR
TRANSFORM TO
STANDARD UNITS
VPTDEF
DTRINQ
MAPPING OF THE CURRENT WINDOW
ON THE NEW VIEWPORT
TO DETERMINE THE NEW DTR
DETERMINE NEW VIEWPORT
USING CURRENT WINDOW
AND THE NEW DTR
DETERMINE VIEWPORT OPPOSITE
CORNER USING CURRENT
WINDOW AND DTR
VPSINQ
VPCINQ
TRANSFORM TO
SN UNITS
REDETERMINE OFFSET
OF THE CURRENT DTR
DETERMINE VIEWPORT AS THE
INTERSECTION OF THE DEFINED
OR COMPUTED VIEWPORT WITH
THE DEVICE SURFACE
DSSINQ
DSCINQ
TRANSFORM TO
SN UNITS

Fig. 3

DEFINE	USE	COMPUTE
WINDOW (WINSET,WINDEF)	VIEWPORT	DTR (MAPPING)
VIEWPORT (VPCSET,VPSSET,VPTDEF)	WINDOW	DTR (MAPPING)
DTR (DTRSET,DTRDEF)	WINDOW	VIEWPORT
WINDOW (WINWCT)	DTR	VIEWPORT
VIEWPORT (VPCWCT,VPSWCT)	DTR	WINDOW

Fig. 4

7. ON THE GRAPHIC CURSOR AND CURRENT POINT.

When the system identifies a GT as "interactive" (cfr. Sect. 4.1.), all the functions that allow the use of a graphic cursor (GC) are abilitated. Of course, the system does not control the GC, but it is the user who decides when and where to set active the GC, where to move the GC by hardware action, and when to return the control to the program.

Once the GC has been switched on, the control can always be returned to the programm entering any key from the display; it is optional to return also the ASCII code of the returned key.

The following input and output functions can be performed in a program:

BGL_GCUPGA : return the coordinates (in "application" units) of the GC position;

BGL_GCUPGC : return the coordinates (in "standard" units) of the GC position;

BGL_GCUPGS : return the coordinates (in "SN" units) of the GC position;

BGL_GCUPSA : set the GC at the point of given coordinates (in "application" units);

BGL_GCUPSC : set the GC at the point of given coordinates (in "standard" units);

BGL_GCUPSS : set the GC at the point of given coordinates (in "SN" units);

The first three functions can be used to set interactively the "window" (in "application" units) and the "viewport" (in any of the possible units of the "device" system). Moreover, it should be noted that if the interactive device has no hardware

for the GC, a simulation can be performed by the software.

A completely different nature has the Current Point (CP), which indicates where it is the graphic pen, that is moved in any graphic output generation. In the BGL reference manual (BGL 3/85) any routine specification reports also the position of the CP at the end of the routine execution.

The knowledge of the CP position can reduce remarkably the number of parameters to be specified in subsequent calls of the BGL routines, and is also useful when character strings have to be drawn, in sequence, on the same graphic area. Also for the CP there are input and output functions which are:

BGL_MOVE : move the CP to a point of given coordinates (in "application" units);

BGL_WHERE : return the CP coordinates (in "application" units);

8. ON THE GRAPHIC OUTPUT.

As just outlined in Sect. 1.1., the main graphics library performs only elementary graphic functions. Higher level functions can be found in the BGLA library (BGLA 4/86). The graphic output routines can be grouped as it follows, according to their functions:

F1 function: generate linear elements (vectors and arcs);
F2 function: fill an area;
F3 function: perform mixed F1 and F2 type functions;
F4 function: clear an area;
F5 function: draw markers;
F6 function: generate text;
F7 function: perform special device-dependent functions.

All the previous functions require coordinates in "application" units, and their execution is controlled by means of parameters ("attributes"), which are optional, i.e. are set initially to default values, but can be modified by the user, when required. Each function has its own attributes, which can be in common with other functions.

8.1. THE FUNCTION F1.

The BGL routines which perform this type of function can:

BGL_LINE : draw a segment from the CP (cfr. Sect. 7.) to a given point;

BGL_ARC : draw an arc of a circumference, given the center coordinates (default value is CP), the starting point (default value is CP), and the angle of the center-to-starting-point direction to the horizontal line passing through the center (default value is 360 degrees);

BGL_CLPBOX : draw a box around the clipping area or, if the clipping area has not been defined, ignore the call and return the proper "return code number".

8.2. THE FUNCTION F2.

This function can be performed only on those devices which are of "raster" type (e.g., VERSATEC or LA100). Other devices of "vectorial" type can perform this function only if supported by specific hardware functions. Therefore, in the following, the action of a routine is individuated either by "shade" or by "fill", to distinguish between "shading" hardware functions (e.g., VT125 or VT240) and "filling" hardware functions (e.g. TEKTRONIX 4107).

With respect to other sofisticated graphics softwares, which allow "patterns" of different types for the filling of an area, the BGL system allows a filling with uniform gray intensity (if possible, with uniform colour intensity) and a variation of the uniform intensity.

Specific performances of routines in this group are:

BGL_FIALSH : "shade" an area generated by the projection of a given segment (from CP to a given point) on a given horizontal line;

BGL_FIAASH : "shade" an area generated by the projection of a given arc (center, starting point and radius given) on a given horizontal line;

BGL_FIAREC : "shade" a box, given two opposite corners;

BGL_FIAPOL : "fill" a polygon defined by a given point sequence.

Moreover, on some devices, such as RAMTEK 6211, the following routine can be executed:

BGL_FIAHRD : fill a polygon starting from a given internal point.

8.3. THE FUNCTION F3.

Specific performances of the routines in this group are:

BGL_BOX : draw a box, given a point and assuming as opposite corner the CP, and "shade" it, if required;

BGL_CIRCLE : draw a circle, given the center point and the radius (if the x and y axes have different scales, the radius is taken on the x scale), and "shade" it, if required.

8.4. THE FUNCTION F4.

Specific performances of the routines in this group are:

BGL_CLRGAR : clear all the graphic output in a box, given two opposite corners of the box;

BGL_CLPCLR : clear all the graphic output in the clipping area, or, if the clipping area has not been defined, ignore the call and return the proper "return code number";

BGL_CLRGSC : clear all the graphic on the device screen, and, if the current device is of "raster" type, feed paper, if required.

8.5. THE FUNCTION F5.

Specific performances of the routines in this group are:

BGL_MARKER : draw a marker centered on a given point (default value is CP).

All the symbol actually available can be found in the BGL reference manual (BGL 3/85).

8.6. THE FUNCTION F6.

For uniformity reasons, the alphanumeric texts are always software constructed, even if the device has the relative hardware functions. Therefore, any character is generated as a sequence of lines, taking generation tables ("fonts") as refence.

The "fonts" actually available on the BGL system are: the "normal font", the "roman font", the "italic font", the "script font". Any "font" has its corresponding version in "greek font". Moreover, exponents, suffixes and some mathematical symbols can be generated.

Examples of the various "fonts" and how to select any of them or how to perform a specific action can be found in an appendix of the BGL reference manual (BGL 3/85). The default "font" is the "normal" one. However, once a font has been selected by the user, it is it which is assumed as a default "font". Specific performances of the routines in this group are:

BGL_TXTSTR : generate on a graphic area an alphanumeric text, given as a character string, or select

the font to be used henceforth ('\fx'="x..... font", '\g'="greek" in the font selected);

BGL_TXTCIN : convert an integer number in a numeric string and write it in an internal area;

BGL_TXTCRE : convert a real number in a numeric string and write it in an internal area;

BGL_TXTNUM : generate on a graphic area a numeric string previously built from a real or integer number;

BGL_TXTLEN : return the length of a given string in "character box" units.

It should be noted that the exact positioning of a string in a graphic area is made possible in the BGL system by the introduction of two "displacement" parameters for x and y coordinates. Once the length of a string is known in "character box" units, it is possible to let start the string exactly where desired, passing as routine arguments the two necessary displacements (in "character box" units), with respect to the given starting point.

8.7. THE FUNCTION F7.

At the moment one only routine belongs to this group, i.e. BGL_HARD. It performs the hard-copy of the graphic area of a screen device on an associated printer. The devices with an hard-copy facility are declared at the installation or reconfiguration time; the user can enter a command of the BGL Command Language (cfr. Sect. 1.2.), i.e.

```
$ BGL SHOW devnam HARD
```

to know the order number which characterizes the printer associated to the device DEVNAM.

8.8. THE FUNCTION ATTRIBUTES.

As anticipated in Sect. 8., the graphic output functions require "attributes", which assume system default values, if not specified by the user in the subroutine arguments. If the BGL routines write a graphic output on a user graphic file, all the "attributes", defaulted or not, are written on the file, so that later the graphic functions could be executed. However, it is also possible to change, in the "visualization" phase (cfr. Sect. 6.2.), some of the "attributes" by means of specific routines. In the following, it is given a list of the "attributes" and of specific actions performed on them by the BGL routines.

i) The "line-type" attribute.

This attribute is shared by the functions of F1, F3, F5 and F6 type. Its initial default value is device dependent and is assigned at the installation or reconfiguration time. For example, it is associated to gray levels in a VT125 terminal or to the thickness of the line in a VERSATEC printer.

The BGL routines can perform the following actions:

BGL_LININD : define a new default value for the "line-type" attribute;

BGL_LININQ : return the current default value and the maximum number of possible values for the "line-type" attribute;

BGL_LINPNT : modify the assigned value of the "line-type" attribute in the "visualization" phase.

ii) The "gray-level" attribute.

This attribute is used only by routines with the function F2. It fixes the uniform gray (or colour) intensity for the filling of an area. The default value is device dependent and is assigned at the installation or reconfiguration time.

The BGL routines can perform the following actions:

BGL_FIAIND : define a new default value for the "gray-level" attribute;

BGL_FIAINQ : return the current default value and the maximum number of possible values for the "gray-level" attribute;

BGL_FIAPNT : modify the assigned value of the "gray-level" attribute in the "visualization" phase.

iii) The "marker-symbol", "marker-orientation", and "marker-size" attributes.

All these attributes are associated to the function F5 and define, respectively:

The symbol to be used to generate a marker at a given point. Its initial default value is 3, which corresponds to a cross-hair symbol (+).

The rotation angle (in degrees) of the symbol with respect to the positive x direction. Its initial default value is zero.

The dimension of the symbol to be drawn, measured in a dimension unit which corresponds to about 0.5 cm on the "virtual" plane. Its initial default value is 1.

A BGL routines can perform the following actions:

BGL_MARATT : modify the current default value of some of these attributes, or of them all.

iv) The "text-orientation" and "text-size" attributes.

All these attributes are associated to the function F6 and define, respectively:

The rotation angle (in degrees) of a string with respect to the positive x direction. Its initial default value is zero.

The dimension of the character to be drawn, measured in a dimension unit which corresponds to about 1.0 cm on the "virtual" plane. Its initial default value is 1.

A BGL routines can perform the following actions:

BGL_TXTATT : modify the current default value of one of these attributes, or of them all.

v) The "picture-identifier" attribute.

This attribute is related to all graphic output functions. Its value has a meaning only when its related output functions are written on a UGF. Actually, the routines which recall a UGF (BGL_SVEXEC and BGL_SVMIMA) could allow the selection of only those graphic outputs with the given "picture-identifier" attribute. Therefore, it is possible, in the "visualization" phase, to select parts of a graphic output, to merge two or more graphic outputs, to visualize the graphic output in an order reversed with respect to the generation one, etc... Its initial default value is zero.

A BGL routine can perform the following action:

BGL_PICIND : modify the current default value of the "picture-identifier" attribute.

The Fig. 5 rapresents the selection criterion for the visualization of a graphic output, with respect to different values of the "picture-identifier" attribute. In this figure, PD is the "picture-identifier" specified as an argument of BGL_SVEXEC and/or BGL_SVMIMA (default value is 0), whereas PR is the "picture-identifier" associated with the graphic output at the generation phase.

The Fig. 6 summarizes all the steps performed when a graphic output operation is executed.

	PR<0	PR=0	PR>0
PD<0	PD=PR	NO	NO
PD=0	NO	YES	YES
PD>0	NO	YES	PD=PR

Fig. 5

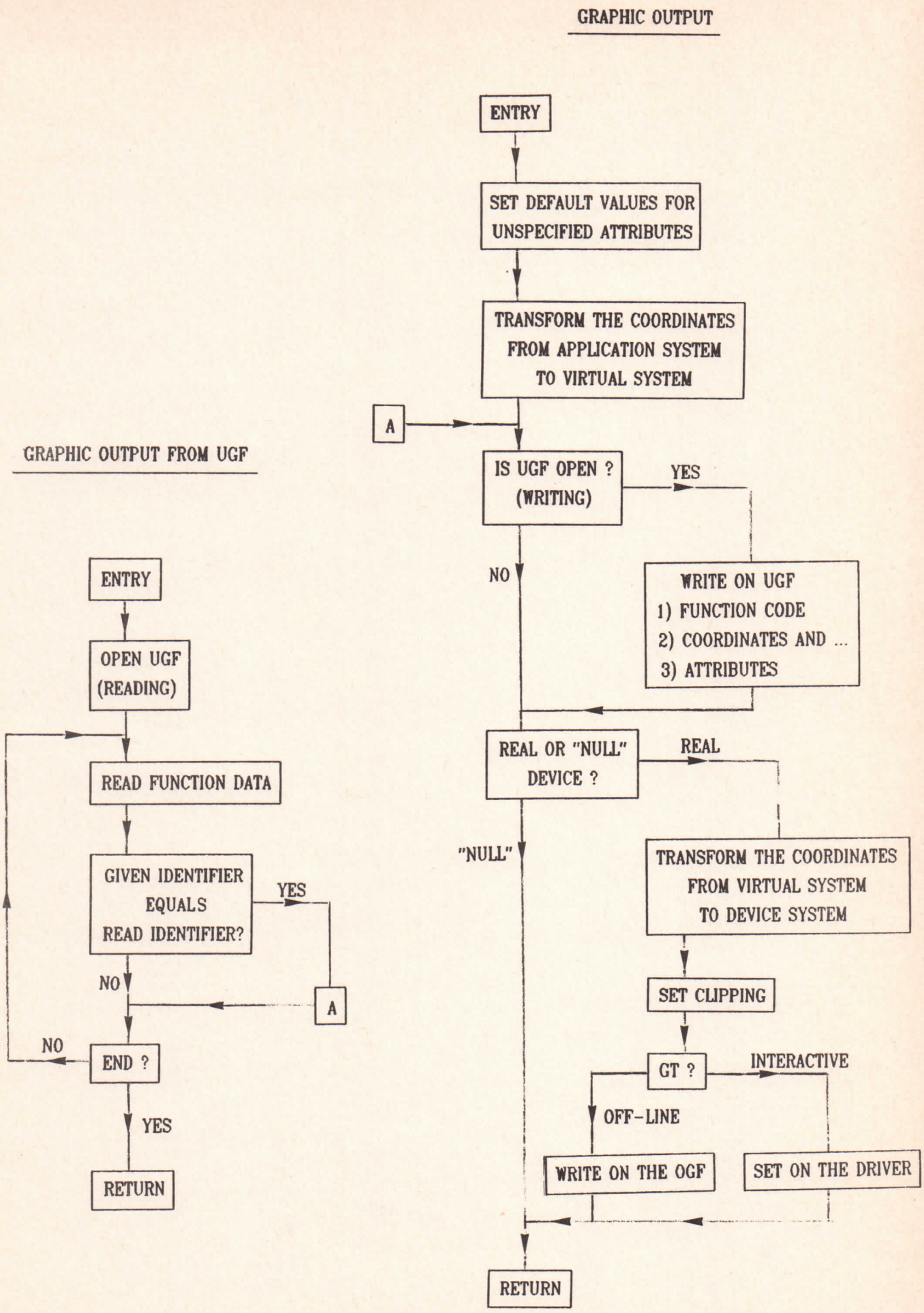
GRAPHIC OUTPUT
ENTRY
SET DEFAULT VALUES FOR UNSPECIFIED ATTRIBUTES
TRANSFORM THE COORDINATES FROM APPLICATION SYSTEM TO VIRTUAL SYSTEM
A
IS UGF OPEN ? (WRITING)
YES
NO
WRITE ON UGF
1) FUNCTION CODE
2) COORDINATES AND ...
3) ATTRIBUTES
REAL OR "NULL" DEVICE ?
REAL
"NULL"
TRANSFORM THE COORDINATES FROM VIRTUAL SYSTEM TO DEVICE SYSTEM
SET CLIPPING
GT ?
INTERACTIVE
OFF-LINE
WRITE ON THE OGF
SET ON THE DRIVER
RETURN
GRAPHIC OUTPUT FROM UGF
ENTRY
OPEN UGF (READING)
READ FUNCTION DATA
GIVEN IDENTIFIER EQUALS READ IDENTIFIER?
YES
NO
A
NO
END ?
YES
RETURN

Fig. 6

9. ON THE COLOUR TABLES.

In the devices with colours, the "line-type" and "gray-level" attributes (cfr. Sect. 8.8.), identify the selected colours for lines and areas, respectively. Of course, the colour number is device dependent and is identified by a number, which is called the color index. The number 0 is associated to the background, the other numbers, up to a fixed N, are foreground colours. The characteristics of any number-associated colour are defined by means of a table, called the look-up table (LUT).

The BGL system allows a device-dependent number of LUT's, the first of these LUT's, defined at the installation or reconfiguration time, is the initial default LUT. The user can utilize one of these LUT's or even define new LUT's up to the maximum number of 64, if the current device has the required resolution. Furthermore, the BGL routines allow also the passage from a RGB system (Red, Green, Blue) to a HLS system (Hue, Lightness, Saturation), when defining a LUT.

To summarize, the BGL routines can:

BGL_HLSRGB	:	convert a given LUT from HLS system to RGB system;
BGL_RGBHLS	:	convert a given LUT from RGB system to HLS system;
BGL_LOOCNT	:	create a new LUT which has all the colour indexes set to zero;
BGL_LOOMTE	:	modify one by one all the colour indexes of an existing LUT.
BGL_LOORTE	:	return the intensity of the three fundamental colours corresponding to a given color index and to a given LUT.

BGL_LOOINQ : return the (fixed) number of foreground colours available on the current device and the current number of LUT's;

BGL_LOOSET : set on the screen a given LUT (a device hardware function is activated).

APPENDIX A - How To Install BGL On a VAX/VMS System.

1. Introduction

The following procedure will install BGL on your VAX/VMS system, starting from a "BACKUP/SAVE_SET" supplied file. The system manager privileges are required.

To start the procedure, you have to type some DCL commands and, then, load and execute a command procedure, that will "bootstrap" and control the remainder of the installation.

The main installation procedure, started automatically by the "bootstrapping" procedure, will compile and link all the BGL modules and will create all the image files needed by user in order to run the BGL routines.

Moreover, the procedure will ask you for information about your specific site configuration. You will be prompted about all the device types for which software interfaces are currently available. For each of these, you will be requested to enter information about the device characteristics and the names of the physical peripherals. A suitable and complete on-line "help" feature will guide you to answer correctly everything (in any case, in order to avoid mistakes, it may be useful for you to have prepared, before starting, a note containing the main information, such as the physical names of all the graphics peripherals, etc...).

When the installation procedure is finished, you will have to perform some final operations, in order to make the BGL system executable.

2. Defining the input backup medium

First of all it is necessary to define the backup input device name. The "SAVE_SET" file may reside either on magnetic tape or on any other Files-11 Level 2 structured device. In this last case it may also reside on a volume of a remote node. You must create a logical name, BCK_DEV, by typing

ASSIGN name BCK_DEV

If the backup medium is a magnetic tape, "name" specifies only the device name; otherwise, if the SAVE_SET file resides on a disk, "name" specifies the disk drive name and the directory name; if the disk is in a remote node, the disk name must include the node name.

Examples :

```
ASSIGN MT:                      BCK_DEV  ( magnetic tape )
ASSIGN DRA2:[DIRNAM]            BCK_DEV  ( local disk )
ASSIGN NORAD::DRA2:[DIRNAM]     BCK_DEV  ( disk of the remote
                                           node NORAD )
```

3. Defining the BGL disk name

Next, you have to create the logical name BGL_DISK in order to define the name of the disk on which all the BGL files will reside. Type

ASSIGN name BGL_DISK

EXAMPLE : ASSIGN DQA0: BGL_DISK

NOTE :

It is important that no directory named [BGL] already

exists on this disk.

4. Loading the "bootstrapping" procedure

Having defined the input and output device names, now you can load the first procedure, that will create the main directory and all the subdirectories for the BGL files. Moreover, this procedure will load the main installation procedure from the backup medium and will create some other necessary logical names and symbols.

If the backup medium is a magnetic tape you must mount the tape before invoking the BACKUP command. Type the following

```
MOUNT/FOREIGN  tape-drive-physical-name  DUMMY
BACKUP/REWIND BCK_DEV:BGL.BCK/SELECT=[BGLN.SERVICE]BOOT.COM
TEMP.COM
```

Otherwise, if the SAVE_SET files resides on disk, you must type

```
BACKUP  BCK_DEV:BGL.BCK/SAVE/SELECT=[BGLN.SERVICE]BOOT.COM
TEMP.COM
```

5. Running the "bootstrapping" procedure

You have just created a command file in your default directory, named TEMP.COM This is a temporary file and will be automatically deleted later. Now you must run this procedure by typing

```
@TEMP
```

This step takes about 3 minutes. If some error occurs during this procedure execution, either because the procedure attempts

to create some directory which already exists, or due to some mistake in typing the quoted logical names definitions, the best way is to correct the error, log-out of your session, log-in again and restart the installation procedure from the beginning.

6. Starting the main installation procedure

After the "boostrapping" procedure is finished, the main installation procedure is activated automatically, and starts by compiling and linking some basic general service modules. Each module name is displayed on the screen, so that, if an error occurs, it is easy to recognize the name of the module which has produced the error.

7. Configuring each device type in your system

The next step performed by the procedure is to get information about the graphics peripherals installed in your system, in order to tailor BGL to your own environment. The procedure reads an internal list which contains the names of all the device types for which software interfaces are currently available and, for each of these, displays a prompt string, like the following

BGL INSTALL devname›

where "devname" is a device type name, whose meaning is easily understandable (such as VT125, VERSATEC etc...).

If no peripheral of this device type is installed in your site, answer the prompt with a "carriage return" (‹CR›); this

answer causes the procedure to go ahead to ask information about the next device type.

Otherwise, if the prompted device type is present in your system configuration, you have to enter all the necessary information about it. To do this, consider that the procedure has built up, and has entered into, a "BGL command language", whose purpose is to allow an easy and efficient way of communication between user and system. The use of this BGL command language is fully explained elsewhere.

Moreover, you can give correctly any information by using a very suitable and complete on-line "help" feature: if you enter a question mark (?), you can get full explanation about the names and the meaning of the commands and parameters you are allowed to enter. To exit from "help", enter a ‹CR›.

The most important information you have to communicate is about the names of the lines connected to the graphics peripherals (such as TXA3:, LVA0: etc..). To do this, you must use the ADD_LINE command.

If the selected device type is a video terminal provided with a hard-copy device, enter the ADD_HARD command.

If you have made some errors in writing the line names, you can use the REM_LINE or REM_HARD commands in order to remove the wrong names.

The commands DEC_NUM and DEC_TYPE should be normally used only if the selected device type is an interactive video terminal and it is not a Digital standard peripheral.

The PHYSICAL_PARAM. command is called in order to modify some values of the physical parameters associated to the current

device type. This command must be followed by a sub-command, whose name may be DISPLAY or MODIFY: the DISPLAY sub-command is useful to read the standard and current values of each parameter and the associated parameter number, whereas the MODIFY sub-command is called to change parameters values. Normally standard values are adequate and you have not to modify anything. Only in very rare cases, especially when there are different models of the same device type, you need to modify some parameters in order to tailor them to your specific model (as an example, there are several models of Versatec printer-plotter, with different densities of "nibs" per inch). Moreover, some parameters are not modifiable, by definition; some other parameters have no meaning for the device type you are dealing with and can not be set or modified. In conclusion you have not to spend much time to process this part of the procedure!

However, remember that two parameters have to be set according to the needs of your computer centre:

1) OFFSET IN X-DIRECTION ...

   If this parameter is set to a value greater than zero (valid only for non-interactive plotter devices) a label with the username and the date will be written at the beginning of each plot. If the computer users are not many, this label may be not necessary

2) AUTOMATIC OUTPUT ON DEVICE

   This parameter controls whether a non-interactive plot is immediately drawn on the device, or it is hold on a disk file until it is released by means of a specific command. This choice may depend on several conditions; as an example, a

planar plotter which accepts only single sheets of paper, must be prepared everytime before receiving a plot etc...

The LOOK_UP_TABLE command is structured exactly as the PHYSICAL_PARAM. command, with the same subcommands. Of course it must be used only if the device is a colour video or plotter, provided with look-up tables. Remember that the VT125 Digital video terminal does have look-up tables and it is provided with RGB output channels for colour monitors.

When you think to have already given all the necessary information about the selected device, enter a ‹CR› to go ahead and configure the next device type. If you have made some error, or have entered a ‹CR› too early, it does not matter: when the installation procedure will be finished, you can correct the errors by using the BGL command language, in particular by calling the CONFIG command.

Before prompting you for the next device information, the procedure compile all the source modules of the related software interface routines. This operation is performed only if the selected device is configured in your system, that is, at least one connected line has been declared.

8. Creating the device-independent routine shareable image

A subdirectory of [BGL] contains the primitive device-independent routines. They are the only user callable routines of BGL and need to be linked to the device-dependent routines and to the service shareable images. When all the devices have been configured, the procedure performs this step,

which is also the last one, and creates the main BGL shareable image (to be linked by the user programs).

9. Final operations

The procedure execution takes about 30 minutes, longer on a busy VAX (if you do not think about the answers to the prompts for too long!).

When the procedure is finished, you have to do few conclusive operations:

1) Insert the statement

   @SYS$MANAGER:BGLSTART

   at the end of your system startup procedure. The file SYS$MANAGER:BGLSTART.COM is created (or modified) by the BGL system everytime something has been modified in the device or software configuration. It contains all the commands needed to define the system logical names used by the BGL routines and to set the graphics terminal characteristics.

2) Insert the statement

   BGL:==@BGL_DISK:[BGL.SERVICE]BGL

   into the system login procedure. This is the only symbol assignment needed in order to run the whole BGL system.

3) Finally, in order to make the BGL system operating, shut-down and then reboot the system.

APPENDIX B - The Main Graphics Library.

Subroutines which terminate the current Graphic Task.

BGL_CLOSE : Close the GT without changing the assigned device.

BGL_NEWDEV : Close the GT and assign a new device.

Subroutines which control the alphanumeric text.

BGL_STAMOD : Place the terminal in alphanumeric mode.

BGL_APLDEF : Define the area where to display alphanumeric text.

BGL_ALPCLR : Clear the alphanumeric text from the defined area.

BGL_CLRASC : Clear the alphanumeric text from the screen.

Subroutines which control the "application" transformations

(cfr. Sect. 6.).

BGL_ATRDEF : Set the default transformation coefficients.

BGL_ATRSET : Set the specified transformation coefficients.

BGL_ATRMAP : Define the mapping from the current transformation onto the wanted one.

Subroutines which control the "device" transformations

(cfr. Sect. 6.).

BGL_DTRDEF : Reset the default transformation coefficients and define the new viewport by applying the new coefficents on the current window.

BGL_DTRSET : Set the specified transformation coefficients and define the new viewport by applying the new coefficients on the current window.

BGL_WINDEF : Reset the default window and compute the new transformation coefficients by mapping the new window onto the current viewport.

BGL_WINSET : Set the specified window and compute the new transformation coefficients by mapping the new window onto the current viewport.

BGL_VPTDEF : Reset the default viewport and compute the new transformation coefficients by mapping the current window onto the new viewport.

BGL_VPCSET : Set the specified viewport (with coordinates in "standard units") and compute the new transformation coefficients by mapping the current window onto the new viewport.

BGL_VPSSET : Set the specified viewport (with coordinates in "screen normalized units") and compute the new transformation coefficients by mapping the current window onto the new viewport.

BGL_WINWCT : Set the specified "window" and determine the new "viewport" with the old DTR;

BGL_VPSWCT : Set the specified "viewport" (in "SN" units) and determine the new "window" with the old DTR;

BGL_VPCWCT : Set the specified "viewport" (in "standard" units) and determine the new "window" with the old DTR;

Subroutines which control the clipping rectangle.

BGL_CLPSET : Set a clipping area at the specified boundary.

BGL_CLPOFF : Remove the clipping area.

Subroutines which control the colour tables.

BGL_LOOCNT : Allocate a new look-up table.

BGL_LOOMTE : Modify an element of the specified look-up table.

BGL_LOORTE : Read an element of the specified look-up table.

BGL_LOOSET : Load the specified look-up table.

BGL_HLSRGB : Convert a colour specification from HLS to RGB.

BGL_RGBHLS : Convert a colour specification from RGB to HLS.

Subroutines which allow interactive use of the Graphic Cursor (GC).

BGL_GCUPGA : Return the GC point in "application unit" coordinates.

BGL_GCUPGC : Return the GC point in "standard unit" coordinates.

BGL_GCUPGS : Return the GC point in "screen normalized unit" coordinates.

BGL_GCUPSA : Move the GC to the point specified in "application unit" coordinates.

BGL_GCUPSC : Move the GC to the point specified in "standard unit" coordinates.

BGL_GCUPSS : Move the GC to the point specified in "screen normalized unit" coordinates.

Subroutines which control the current point (CP).

BGL_MOVE : Move the CP to the specified point.

BGL_WHERE : Return the coordinates of the CP point.

Subroutines which draw linear elements.

BGL_LINE : Draw a straight line.

BGL_ARC : Draw an arc of the specified length.

BGL_CLPBOX : Draw a box along the boundary of the given clipping area.

Subroutines which fill a selected area.

BGL_FIAASH : Shade an arc to the specified horizontal line.

BGL_FIAHRD : Flood polygons with a solid colour (device-dependent function).

BGL_FIALSH : Shade a line to the specified horizontal line.

BGL_FIAPOL : Fill any type of polygon defined by a sequence of vertex coordinates.

BGL_FIAREC : Fill a rectangular area.

Subroutines which draw closed lines and fill the internal area, if required.

BGL_BOX : Draw a box and fill it, if required.

BGL_CIRCLE : Draw a circle and fill it, if required.

Subroutines which clear graphic objects from selected areas.

BGL_CLPCLR : Clear all graphic output in the clipping area.

BGL_CLRGAR : Clear all graphic output in the specified area.

BGL_CLRGSC : Clear all graphic output on the screen, or advance paper, if the current device is a graphic printer).

Subroutines which draw a graphic text.

BGL_TXTSTR : Draw a text string.

BGL_TXTCIN : Convert an integer number to a decimal text string and save the string in an internal memory area.

BGL_TXTCRE : Convert a real (floating point) number to a decimal text string and save the string in an internal memory area.

BGL_TXTNUM : Draw the number previously converted to a text string.

BGL_MARKER : Draw a special symbol centered at the specified point.

BGL_TXTLEN : Return the length of a given string in "character box" units.

Subroutines which save or restores pictures.

BGL_SVINIT : Initialize the specific User Graphic File (UGF). All the subsequent graphic operations will be saved in the UGF.

BGL_SVCLOS : Close the UGF opened with a call to BGL_SVINIT.

BGL_SVMIMA : Compute boundary coordinates of a picture previously saved in a UGF.

BGL_SVEXEC : Display pictures previously saved in a UGF.

BGL_HARD : Print the screen image on the defined "hard-copy" printer (device dependent function).

Subroutines which control the function attribute (cfr. Sect. 8.6.).

BGL_LININD : Set the default LINE_TYPE attribute value.

BGL_LINPNT : Redefine a LINE_TYPE attribute value.

BGL_FIAIND : Set the default GRAY_TYPE attribute value.

BGL_FIAPNT : Redefine a GRAY_TYPE attribute value.

BGL_PICIND : Set the default PICTURE IDENTIFIER attribute value

BGL_MARATT : Set the default MARKER attribute value

BGL_TXTATT : Set the default TEXT attribute value.

Subroutines which inquire for...

BGL_ATRINQ : ...the coefficients of the current "application" transformation

BGL_DTRINQ : ...the coefficients of the current "device" transformation

BGL_DEVINQ : ...the name of the current device

BGL_DSCINQ : ...the coordinates of the boundary of the physical viewport (in "standard units")

BGL_DSSINQ : ...the coordinates of the boundary of the physical viewport (in "screen normalized units")

BGL_DSZINQ : ...the device coordinates in "standard units" corresponding to the value 1 in "screen normalized units"

BGL_LININQ : ...the current default of the LINE_TYPE attribute value and the maximum number of values allowed at the current device

BGL_FIAINQ : ...the current default of the GRAY_LEV attribute value and the maximum number of values allowed at the current device

BGL_LOOINQ : ...the current look-up table and the maximum number of defined look-up tables

BGL_VPCINQ : ...the coordinates of the boundary of the defined (or computed) viewport, in "standard units"

BGL_VPSINQ : ...the coordinates of the boundary of the defined (or computed) viewport, in "screen normalized units".

APPENDIX C - The Basic Calcomp Software Interface.

Subroutines which perform basic functions.

PLOTS	:	Initialize a new Graphic Task (GT).
PLOT	:	Draw a straight line, or move the current point, and/or redefine the axis origin, or close the current GT.
FACTOR	:	Enlarge or reduce the size of the subsequent picture object.
NEWPEN	:	Select a different line type.
NUMBER	:	Convert a floating-point number to the decimal equivalent string and draw the string.
SYMBOL	:	Draw a text string.
WHERE	:	Return the current point coordinates.

Subroutines which perform more complex functions.

SCALE	:	Compute scale factors to process unscaled data with LINE and AXIS.
AXIS	:	Draw a positioned axis line with labels, scale annotations and tick marks.
LINE	:	Plot points from coordinate data arrays.
CURVE	:	Draw a smooth curved line trough a set of user-defined coordinate data points.

APPENDIX D - The Terminal Management Library.

Subroutines which are effective on any type of terminal.

BGL_VTBELL : Ring the bell one/more time.

BGL_VTCLOSE : Restore initial conditions.

BGL_VTNOBRD : Set/reset the NOBROADCAST characteristics.

BGL_VTNOECH : Set/reset the NOECHO characteristics.

BGL_VTREAD : Read a character string from the terminal keyboard.

BGL_VTWIDTH : Define the character number on each imput/output line.

BGL_VTWNUM : Write an integer number.

BGL_VTWNUMB : Write a right justified integer number into a fixed length field.

BGL_VTWNUMZ : Write a right justified integer number into a fixed length field and fill the field with leading zeros.

BGL_VTWRAP : Set/reset the WRAPAROUND characteristics.

BGL_VTWRITE : Write a text string.

Subroutines which are effective only on terminals conform to ANSI standard, such as DEC VT100.

BGL_VTATTR : Turn on/off the REVERSE, BLINK, BOLD and UNDERSCOPE attributes.

BGL_VTVTCGET : Return the alphanumeric coordinates of the cursor location.

BGL_VTCHTM : Move the cursor along a line.

BGL_VTCSAVE : Save the cursor position and the character attributes in the video local memory.

BGL_VTCREST : Restore the cursor position and the character attributes saved in the video local memory.

BGL_VTCSET	:	Move the alphanumeric cursor to the specified location.
BGL_VTCVTM	:	Move the cursor along a column.
BGL_VTDOUBH	:	Write a double-height, double-width line.
BGL_VTDOUBW	:	Write a single-height, double-width line.
BGL_VTEDOUB	:	Restore the normal size of a line.
BGL_VTEAREA	:	Erase a defined area on the screen.
BGL_VTELINE	:	Erase a line partially or completely.
BGL_VTESCRN	:	Erase the screen partially or completely.
BGL_VTGETCH	:	Read a character and return its ASCII code.
BGL_VTGRAF	:	Select/remove the Special Graphics Set.
BGL_VTGRID	:	Draw a box with any number of internal segments.
BGL_VTGRINP	:	Enable the operator to move the cursor and return the cursor final location.
BGL_VTLEDS	:	Turn on/off the leds on the terminal keyboard.
BGL_VTM132	:	Set/reset the 132 COLUMNS mode.
BGL_VTMKEY	:	Set/reset the KEYPAD APPLICATION mode.
BGL_VTMREV	:	Set/reset the REVERSE VIDEO mode.
BGL_VTMWRAP	:	Set/reset the WRAPAROUND mode.
BGL_VTRNUM	:	Read an integer number from the terminal keyboard.
BGL_VTSCRLL	:	Define the video scrolling area.