

# GP GS-F

## User's Guide 8th Edition



**SINTEF**  
**DELAB**



**NORSIGD**

Norsk  
samarbeid  
innen  
grafisk  
databehandling

Norwegian  
Association  
for  
Computer  
Graphics

## Notice

---

SINTEF DELAB / NORSIGD retain all ownership rights to the GPGS-F software and its documentation.

The information in this document is subject to change without notice. SINTEF DELAB / NORSIGD assume no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license. SINTEF DELAB / NORSIGD are in no way responsible for consequential damages and/or costs from the use of this software.

FrameMaker is a registered trademark of Frame Technology Corporation.

PostScript is a registered trademark of Adobe Systems Incorporated.

X Window System is a trademark of the Massachusetts Institute of Technology.

This document was prepared using FrameMaker publishing software. All figures illustrating program examples were generated by using the GPGS-F device driver for PostScript, and subsequently included into the document.

# Table of Contents

	Page
Notice . . . . .	ii
Table of Contents . . . . .	iii
Preface . . . . .	xi
Font Conventions Used in the Manual . . . . .	xiii
Argument Naming Conventions . . . . .	xiii
Manual Distribution and Revisions . . . . .	xiv
Modification Dates . . . . .	xv

## Chapter 1

### *Graphic Devices*

1.1	System Initialization . . . . .	1-1
1.2	Device Control . . . . .	1-2
1.2.1	Device Options. . . . .	1-4
1.2.2	Inquiring Available Device Drivers . . . . .	1-5
1.3	Synchronizing GPGS-F Output with Other I/O Operations . . . . .	1-5
1.4	GPGS-F Version Numbering . . . . .	1-6

## Chapter 2

### *Windows, Viewports and Clipping*

2.1	Window - Viewport Mapping . . . . .	2-1
2.1.1	Window Definition . . . . .	2-1
2.1.2	Viewport Definition . . . . .	2-2
2.1.3	Default Values . . . . .	2-3
2.1.4	Window and Viewport Dimensions . . . . .	2-3
2.2	Clipping . . . . .	2-4
2.3	Coordinate Processing . . . . .	2-4

## Chapter 3

### *Introduction to Picture Segments*

## Chapter 4

### *Basic Graphic Primitives*

4.1	Linetypes . . . . .	4-1
4.1.1	Linepattern Length . . . . .	4-2
4.2	Drawing Single Lines . . . . .	4-2
4.3	Circular Arcs. . . . .	4-7
4.3.1	Software / Hardware Generation . . . . .	4-10
4.3.2	Circle Smoothness. . . . .	4-10
4.4	Elliptic Arcs . . . . .	4-11
4.5	Markers . . . . .	4-12

## Chapter 5

### *Drawing in True Scale*

## Chapter 6

### *Transformations*

6.1	System Transformation Matrix . . . . .	6-1
6.2	Basic Transformation Routines . . . . .	6-2
6.2.1	Translation . . . . .	6-3
6.2.2	Scaling . . . . .	6-4
6.2.3	Rotation . . . . .	6-4
6.2.4	Shearing . . . . .	6-6
6.2.5	Vanishing Point . . . . .	6-7
6.3	Combining Basic Transformations . . . . .	6-8
6.4	Transformation Matrix Manipulation . . . . .	6-9
6.4.1	Internal Matrix Stack . . . . .	6-9
6.4.2	Direct User Manipulation . . . . .	6-10
6.5	Viewing Routines . . . . .	6-12
6.5.1	Focal Point . . . . .	6-14
6.5.2	Vanishing Point . . . . .	6-16
6.6	Transformation Mode . . . . .	6-16

## Chapter 7

### *Character Strings*

7.1	Drawing Text Strings . . . . .	7-1
7.1.1	Format Control. . . . .	7-2
7.2	Drawing Integer and Real Numbers. . . . .	7-3
7.3	Character Size . . . . .	7-4
7.4	Character Transformations . . . . .	7-5
7.4.1	Shearing . . . . .	7-5
7.4.2	Rotation . . . . .	7-6
7.5	Software / Hardware Text Generation . . . . .	7-6

7.6	Text Alignment . . . . .	7-7
7.7	Text Fonts. . . . .	7-8
7.8	Character Encoding. . . . .	7-10
7.8.1	National Character Sets . . . . .	7-10
7.9	Proportional Spacing . . . . .	7-12
7.10	Inquiring Text Extent . . . . .	7-13

## Chapter 8

### *Interaction Facilities*

8.1	Basic Interactive Programming . . . . .	8-2
8.2	Interaction Modes . . . . .	8-4
8.2.1	Sample Mode Input . . . . .	8-4
8.2.2	Event Mode Input . . . . .	8-5
8.3	Echo Control. . . . .	8-8
8.3.1	Echo Specification. . . . .	8-8
8.4	Methods for Text Output . . . . .	8-11
8.5	Interaction With a Second Device . . . . .	8-11
8.6	Reading Additional Input Data . . . . .	8-12
8.7	Compatibility With Previous Versions . . . . .	8-13
8.8	Coordinate Conversion Routines . . . . .	8-14

## Chapter 9

### *Defining Line Patterns and Representation*

9.1	Defining Line Patterns . . . . .	9-2
9.2	Defining Line Representation . . . . .	9-3
9.2.1	Line Representation Parameters . . . . .	9-5

## Chapter 10

### *Polylines and Curves*

10.1	Polylines . . . . .	10-1
10.1.1	Automatic Value or Index Increment . . . . .	10-2
10.2	Parameterized Curves . . . . .	10-4
10.2.1	Automatic Value Increment . . . . .	10-5

## Chapter 11

### *Colour Specification*

11.1	Colour Index Selection. . . . .	11-2
11.2	Colour Models . . . . .	11-2
11.2.1	RGB Colour Model . . . . .	11-3
11.2.2	The HLS Colour Model . . . . .	11-4
11.2.3	The HSV Colour Model . . . . .	11-5
11.3	Monochrome Devices . . . . .	11-6

## Chapter 12

### *Raster Graphics*

12.1	Raster Graphics Programming . . . . .	12-1
12.2	Polygons . . . . .	12-2
12.2.1	Polygon Drawing . . . . .	12-3
12.2.2	Interior Style . . . . .	12-4
12.2.3	Perimeter Drawing . . . . .	12-4
12.2.4	Texture Rendering. . . . .	12-5
12.2.4.1	Texture Quality. . . . .	12-5
12.2.4.2	Pattern and Hatch Style Tables . . . . .	12-7
12.2.4.3	Global Texture Attributes . . . . .	12-8
12.2.4.4	Applying Texture to 3D Polygons . . . . .	12-11
12.3	Pixel Arrays . . . . .	12-12
12.3.1	Software / Hardware Generation . . . . .	12-12
12.3.2	Inquiring Pixel Values From the Display. . . . .	12-15

## Chapter 13

### *Picture Element Attributes*

13.1	Linewidth. . . . .	13-1
13.2	Depth Modulation . . . . .	13-2
13.3	Blinking . . . . .	13-2

## Chapter 14

### *Picture Segment Storing*

14.1	Segment Classes. . . . .	14-1
14.1.1	Pseudo Picture Segments . . . . .	14-1
14.1.2	Retained Picture Segments . . . . .	14-2
14.2	Picture Segment Identifiers . . . . .	14-2
14.3	Defining Picture Storage . . . . .	14-2
14.3.1	Primary Buffers . . . . .	14-3
14.3.1.1	Programming Guidelines. . . . .	14-4
14.3.2	Picture Libraries . . . . .	14-4
14.4	Copying Segments . . . . .	14-6
14.5	Deleting Segments . . . . .	14-7

## Chapter 15

### *Pseudo Picture Segments*

15.1	Inserting Pseudo Segments . . . . .	15-1
15.1.1	Colour of Inserted Primitives . . . . .	15-2
15.1.2	Areas of Application . . . . .	15-4
15.2	Clipping . . . . .	15-4
15.3	Pseudo Segment Reference . . . . .	15-4

## Chapter 16

### *Retained Picture Segments*

16.1	Storage Mode . . . . .	16-1
16.2	Deferral Mode . . . . .	16-2
16.2.1	Compatibility Routines . . . . .	16-3
16.3	Redrawing . . . . .	16-4
16.4	Deleting Segments . . . . .	16-4

## Chapter 17

### *Retained Segment Attributes*

17.1	Visibility . . . . .	17-1
17.2	Highlighting . . . . .	17-3
17.3	Priority . . . . .	17-3

## Chapter 18

### *Image Transformations*

## Chapter 19

### *Background Device*

19.1	Background Viewport . . . . .	19-2
19.2	Limitations . . . . .	19-2

## Chapter 20

### *Pick Input*

20.1	Element Namestack. . . . .	20-1
20.2	Element Detectability . . . . .	20-3
20.3	Segment Detectability . . . . .	20-4
20.4	Scanning for Hit. . . . .	20-5
20.5	Using Pseudo Segments . . . . .	20-7

## Chapter 21

### *Multi Window Devices*

21.1	Window to Viewport Mapping . . . . .	21-2
21.2	Window Management . . . . .	21-2
21.3	Window Operations . . . . .	21-6
21.4	Window Numbers . . . . .	21-9
21.5	Retained Segments . . . . .	21-10
21.6	Updating Window Contents . . . . .	21-10
21.7	Requesting Window Size . . . . .	21-12
21.8	Interaction . . . . .	21-13
21.9	Background Device. . . . .	21-13

## Chapter 22

### *Hidden Lines and Surfaces Removal*

22.1	HLHS Module Control. . . . .	22-2
22.2	Inserting the Result . . . . .	22-3
22.3	Using the Dummy Device. . . . .	22-6
22.4	Front- and Back-Facing Polygons . . . . .	22-7
22.5	Polygon Attributes . . . . .	22-8
22.6	Limitations . . . . .	22-8

## Chapter 23

### *Fetching System Status Data*

## Chapter 24

### *Errors and Messages*

24.1	GPGS-F Error Vector . . . . .	24-1
24.2	Default Error Handling . . . . .	24-2
24.3	Error File . . . . .	24-3
24.4	Application Supplied Error Routine . . . . .	24-4
24.4.1	Closing Down GPGS-F . . . . .	24-5



## Appendices

---

---

### Appendix A

#### *Installation Dependent Parameters*

### Appendix B

#### *Software Character Fonts*

### Appendix C

#### *Additional GPGS-F Products*

C.1 MICRO-GPGS-F . . . . .	C-1
C.1.1 Main Limitations Compared to GPGS-F . . . . .	C-2
C.2 GRAPHISTO . . . . .	C-2
C.3 SURRENDER . . . . .	C-4

### Appendix D

#### *Machine Dependencies*

D.1 File / Communication Channel Numbers . . . . .	D-1
--	-----

### Appendix E

#### *Device Driver Descriptions*

### Appendix F

#### *Routine Name Index*

### Appendix G

#### *Routine Number Index*

G.1 GPGS-F Routines. . . . .	G-1
G.2 GRAPHISTO Routines . . . . .	G-3
G.3 SURRENDER Routines. . . . .	G-4

### Appendix H

#### *C Language Interface*

### Appendix I

#### *Error Messages*

### Keyword Index



## **Preface to the first edition**

---

This manual is intended to be used as a guide when writing programs in GPGS-F. It may also be used as a text book or for self study. Moreover, there is an index of all the routines in **Appendix F** and it may thus be used for reference.

The GPGS-F system was originally designed by Rekencentrum Delft University of Technology, The Netherlands; Science Faculty, Catholic University Nijmegen, The Netherlands.

A version of the system written in standard Fortran has been developed by NORSIGD (Norwegian Association for Computer Graphics) at the Computing Centre at the University of Trondheim (RUNIT, now DELAB).

As GPGS-F is under continuous development, additions and corrections of this manual will be produced.

**NORSIGD**  
**September 1975**

## **Preface to the eighth edition**

---

This edition of the User's Guide describes version 9503 of the GPGS-F system.

The most important extension compared to the previous version, is the inclusion of routines for 'multi-window' devices.

Several extensions to the existing routines have been included. These include extending the allowable number of retained picture segments, removing the limitation on how many segments may be stored in picture libraries, and extending the GPGS-F text routines to handle 8 bit character codes.

In general, the modifications to the GPGS-F system during the past years have been affected by the evolving changes in use of computer graphics devices. Thus, GPGS-F is now very well suited for use with raster graphics workstations running some window system, and with colour and monochrome raster printers/plotters. Still, the system is fully backwards compatible, allowing old application to be run without modification with new kinds of devices.

**NORSIGD / SINTEF DELAB**  
**April 1995**  
**Magnar Granhaug**



## Font Conventions Used in the Manual

---

To ease readability, different fonts are used throughout the manual.

Subroutine definitions are presented as

**CALL REQHIT (Itool, Maxnam, Namarr(1), Lennam)**

Argument names are written as **Itool** in the description that follows the definition. The same font is also used when later referring to subroutine arguments.

The notation **Namarr(1)** is used to mark that an argument is an array.

When an argument is used to return value(s) to the application program, this is marked by underlining the argument, as Lennam above.

With the exception of the definition, subroutine names appearing in the text are written as *REQHIT*.

Examples of program code will appear as

```
CALL GPGS
CALL NITDEV (IDEV)
.
```

When an example shows a complete program, this is marked with the comment

```
C  COMPLETE WORKING EXAMPLE
```

Some examples include calls to external routines that are not part of GPGS-F. The routine names are then written in italics, as

```
CALL HOUSE
```

*Italics* and **Boldface** elsewhere in the text is used to emphasize certain words or phrases, such as cross-references and some important terms when first mentioned.

## Argument Naming Conventions

---

In program examples and subroutine definitions, Fortran implicit declaration of variables is assumed, unless explicitly stated otherwise.

That is, argument names starting with **I - N** are integer variables, others are floating point variables, except argument names starting with **C**, which are character variables (must be declared).

## Manual Distribution and Revisions

---

Starting with the 8th edition of the GPGS-F User's Guide, the manual is available on the World Wide Web, in PostScript format.

To get to the manual, just follow the **GPGS** link from NORSIGD's home page,  
**<http://www.oslo.sintef.no/norsigd/>**

This allows for a more dynamic change of the manual. Whenever changes are necessary, either to correct errors or to describe new features, these will be inserted in the manual without delay, and the changes will be announced through NORSIGD and/or WWW.

There will be a list available, giving the latest modification date of each chapter and appendix of the manual. At the bottom right of each page, this date is given. Thus, readers will easily see what chapters of their manual are revised, and may download (or order) these individual chapters.

The modification dates are given per chapter/appendix, even if only a single page is changed, with one exception. Because of its size, **Appendix E** is divided into 3 parts, with the first one being an introduction, the second part describes drivers 0 to 57, and the last part describes the drivers from 58 and up.

The list of modification dates is also given on the next page. Whenever a chapter/appendix is changed, this list is also updated, i.e. readers should always download this list when downloading a new update of a chapter/appendix.

## Modification Dates

Chapter/ Appendix	Last changed
Notice	Apr 7, 1995
Table of Contents	May 10, 1995
Preface	Apr 7, 1995
About the Manual	May 30, 1996
Modification Dates	May 30, 1996
<b>1</b> Graphic Devices	Apr 7, 1995
<b>2</b> Windows, Viewports and Clipping	Apr 7, 1995
<b>3</b> Introduction to Picture Segments	Apr 7, 1995
<b>4</b> Basic Graphic Primitives	Aug 7, 1995
<b>5</b> Drawing in True Scale	Apr 7, 1995
<b>6</b> Transformations	Apr 7, 1995
<b>7</b> Character Strings	May 10, 1995
<b>8</b> Interaction Facilities	Apr 7, 1995
<b>9</b> Defining Line Patterns and Representation	Apr 7, 1995
<b>10</b> Polylines and Curves	Apr 7, 1995
<b>11</b> Colour Specification	Apr 7, 1995
<b>12</b> Raster Graphics	Aug 9, 1995
<b>13</b> Picture Element Attributes	Apr 7, 1995
<b>14</b> Picture Segment Storing	May 10, 1995
<b>15</b> Pseudo Picture Segments	Apr 7, 1995
<b>16</b> Retained Picture Segments	May 10, 1995
<b>17</b> Retained Segment Attributes	May 10, 1995
<b>18</b> Image Transformations	Apr 7, 1995
<b>19</b> Background Device	Jan 18, 1996
<b>20</b> Pick Input	Apr 7, 1995
<b>21</b> Multi Window Devices	Mar 14, 1996

Chapter/ Appendix	Last changed
<b>22</b> Hidden Lines and Surfaces Removal	Apr 7, 1995
<b>23</b> Fetching System Status Data	Jan 18, 1996
<b>24</b> Errors and Messages	Apr 7, 1995
<b>A</b> Installation Dependent Parameters	Apr 7, 1995
<b>B</b> Software Character Fonts	May 10, 1995
<b>C</b> Additional GPGS-F Products	Apr 7, 1995
<b>D</b> Machine Dependencies	Apr 7, 1995
<b>E</b> Device Driver Descriptions (Introduction)	Jan 18, 1996
<b>E</b> Device Driver Descriptions (drivers 0 to 57)	Jan 18, 1996
<b>E</b> Device Driver Descriptions (drivers 58 and up)	Apr 9, 1996
<b>F</b> Routine Name Index	Apr 7, 1995
<b>G</b> Routine Number Index	Apr 7, 1995
<b>H</b> C Language Interface	Apr 7, 1995
<b>I</b> Error Messages	Apr 7, 1995
Keyword Index	Aug 16, 1995





# Chapter 1

## Graphic Devices

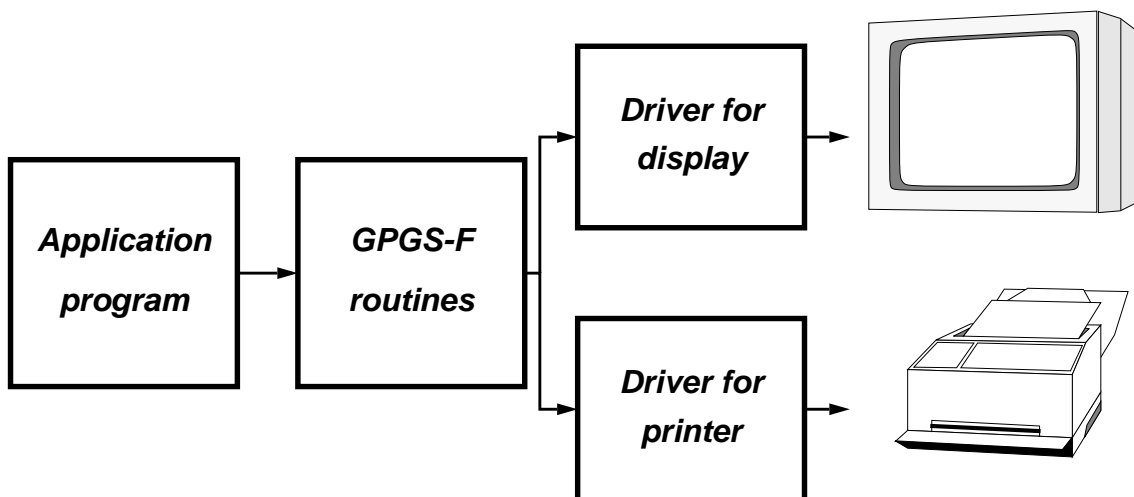
---

GPGS-F has been designed to support several graphic devices from the same application program. The device independence of GPGS-F is realized by placing all code dependent on the physical characteristics of each graphic device in a separate module called a *device driver* (the available drivers are listed in **Appendix E**).

Adapting the system, and hence application programs, to new kinds of graphic hardware is thus achieved by adding new device drivers. No changes in the main part of GPGS-F is necessary, except for a simple table update.

In most cases, a device driver generates code for a specific output device. There are however some drivers that are common to several devices, such as the **HPGL** driver, which may generate output for a large number of different Hewlett-Packard plotters.

**Figure 1.1** *Picture generation on display and printer.*



### 1.1 System Initialization

In any GPGS-F application program, the first call to a GPGS-F routine must be

**CALL GPGS**

This initializes the system and ensures that all internal data is set to its initial state.

## 1.2 Device Control

Before generating any graphic output, a device to receive the output must be specified. First, the driver for the device must be initialized by

**CALL NITDEV (ldev)**

where **ldev** is the GPGS-F unit number for the device driver. This will both initialize the driver software and, if necessary, prepare the physical device for receiving output. A device driver that has been initialized by *NITDEV*, is said to be *open*.

It is possible to have up to four devices open at any time. If more than four are wanted, one of the first four must be released before initializing the fifth. Although several devices may be open at a time, GPGS-F is able to generate graphic output for just one device at a time, using the current *active* device driver.

Note that *NITDEV* just specifies what device to generate output for, it does not say where the output should be sent. By default, the output is directed to the terminal running the program, but this may be changed by the *DEVOPT* routine described on **page 1-4**.

By default, the last driver initialized by *NITDEV* will be active. Selecting one of the other open drivers to be active is done by

**CALL SELDEV (ldev)**

where **ldev** is defined as for *NITDEV*.

When a device is no longer to be used, it must be released by

**CALL RLSDEV (ldev)**

This will ensure that the device is closed in a proper way, and if retained segments are stored for the device (see **Chapter 14** and **Chapter 16**), these will be deleted from incore buffers.

When drawing more than one picture using the same device, the display surface must be cleared between each drawing by

**CALL CLRDEV (ldev, lopt)**

Most device drivers will ignore the value of **lopt**. For a few drivers, the value may however be used to control some device dependant features (described in **Appendix E**).

The actual action performed by *CLRDEV* will obviously depend on the type of device in use. For terminals, the previous picture will be erased, for plotters/printers, a new piece of paper will be loaded, or in case of roll paper, the paper will be advanced. If automatic paper change is not available, paper must be changed manually by the user (which devices this apply to is also shown in **Appendix E**).

Note that *NITDEV* implies a call to *CLRDEV*, i.e. there is no need to call *CLRDEV* before the first drawing. Nor is it necessary to call *CLRDEV* after the last drawing, as *RLSDEV* will perform the necessary paper advance for the last page.

Note that *CLRDEV* will, as *RLSDEV*, delete any retained segments from incore buffers.

A common use of multiple devices, is to have a hard copy or plotting device initialized at the same time as an interactive device. Thus an interactive user may request a copy of his current display by merely switching devices and executing the same sequence of GPGS-F calls to the plotting device as previously issued to the interactive display (GPGS-F does however provide a method for getting hardcopies by specifying the plotting device as a *background device*, see **Chapter 19**).

### **Example 1.1**     *Device switching.*

```

C
C   Initialize GPGS-F and plotter device.
C
C       CALL GPGS
C       CALL NITDEV(IDEV1)
C
C   Initialize graphic terminal.
C
C       CALL NITDEV(IDEV2)
C
C   The last initialized device is active.
C   Now generate a drawing according to arguments
C   (that the user can change interactively?).
C   'DSPLAY' is a user routine.
C
C   1000 CONTINUE
C       CALL DSPLAY(ARGUMENTS)
C       IF ('hardcopy') THEN
C
C           Switch to plotter device and make a plot using the same
C           drawing routine with the same argument values.
C
C               CALL SELDEV(IDEV1)
C               CALL DSPLAY(ARGUMENTS)
C           ENDIF
C
C   If more plots wanted, clear the screen,
C   prepare new paper on the plotter.
C
C       IF ('one_more_plot') THEN
C           CALL CLRDEV(IDEV1, 0)
C           CALL CLRDEV(IDEV2, 0)
C           CALL SELDEV(IDEV2)
C           GO TO 1000
C       ENDIF
C
C   Release devices, end program.
C
C       CALL RLSDEV(IDEV1)
C       CALL RLSDEV(IDEV2)
C       END

```

## 1.2.1 Device Options

When a device driver is initialized by *NITDEV*, various default conditions are set, both in the driver software and device firmware.

In some cases these defaults are not what the user wants, and there is a need for some user control of the driver start-up conditions. A routine is therefore available for setting some device driver options to be used at driver initialization.

This routine is defined as

**CALL DEVOPT (larr(1), llthi, Rarr(1), llthr, Carr(1), llthc)**

where options are supplied through **larr** (integers), **Rarr** (real numbers) and **Carr** (text strings). **llthi**, **llthr** and **llthc** gives the length of each of the three arrays.

The driver descriptions in **Appendix E** show how many options are recognized by the different device drivers, and what device specific features the options control.

The first integer option (**larr(1)**) is however defined the same way for all devices. This gives the unit number of the communication channel (or file number) that is to be used for driver output. With some computers the unit number is chosen by the application programmer, with others the number must be obtained from GPGS-F (see **Appendix D** for details).

By definition, setting an option value to 0 (zero), or an empty string for text string options, means that the default value of the option is to be used. The same applies to options not supplied.

A single option may be set, without affecting options already set, by specifying the negative option number in place of the array length (see example below).

### **Example 1.2**     *Setting device options.*

```

C
C  Declare an integer option array with fixed values.
      DIMENSION IOPTS(3)
      DATA IOPTS/33, 1, 3/
C
C  Initialize GPGS-F, set fixed options.
      CALL GPGS
      CALL DEVOPT(IOPTS, 3, RDUM, 0, ' ', 0)
C
C  Get the value to be used for real option 4 (user routine)
      CALL Getop4(RVAL4)
C
C  Set real option 4, without changing options already set.
      CALL DEVOPT(0, 0, RVAL4, -4, ' ', 0)
C
C  Then initialize the device.
      CALL NITDEV(IDEV)

```

The options set by *DEVOPT* are sent to the *next* driver that is initialized. That is, if several drivers are initialized, *DEVOPT* must be called before each call to *NITDEV*. Also note that *DEVOPT* may be used to set options for individual device windows when using multi window devices (see **Chapter 21**).

For compatibility with previous versions of GPGS-F, integer options may be set by

**CALL NITOPT (larr(1), llth)**

where **larr** and **llth** are defined as for *DEVOPT*. Note however that the maximum value of **llth** is 10 when using *NITOPT*.

### 1.2.2 Inquiring Available Device Drivers

In most cases, the application programmer will know what device drivers his program is to be used with. If this is not the case, it is possible to ask GPGS-F what drivers are linked with the program.

The availability of a given device driver is returned by

**CALL INQDRV (ldev, lstat)**

where **ldev** is the GPGS-F driver number. **lstat** will return 1 if the driver is linked, -1 if not. If **ldev** is not a legal GPGS-F device number, **lstat** -2 will be returned.

## 1.3 Synchronizing GPGS-F Output with Other I/O Operations

When using GPGS-F with 'traditional' graphic terminals, and plotters connected between the computer and a terminal, the same communication line is used for graphic output from GPGS-F and other I/O operations for the application program. In most cases, such devices will have to be switched to some sort of *graphic mode* to be able to interpret input as graphic commands instead of ordinary text. If the application program then sends some text to the device, this may not be recognized.

GPGS-F will switch the current device to graphic mode when a *picture segment* is opened, and switch back to the previous mode when the picture segment is closed (picture segments are introduced in **Chapter 3**). Thus, to be sure no text is interpreted as graphic commands, the application should avoid generating any I/O outside GPGS-F while a picture segment is open.

If, for some reason, an application program needs to perform I/O while a segment is open, the device may be forced to leave graphic mode by

**CALL UPDAT (0)**

This will also ensure that graphic commands buffered within GPGS-F are flushed (the *UPDAT* routine is described in detail in **Chapter 16**).

## 1.4 GPGS-F Version Numbering

It is possible to find what version of GPGS-F that is linked with an application by

**CALL INQGPV (lbasv, lsubv)**

where **lbasv** will return the version number of the common base version that is the base for **lsubv**, which is the version number of the actual version (VMS, Unix etc.) the application is running. Both version numbers are given as a 4-digit number **yymm**, where **yy** is the year, and **mm** is the month (e.g. **9309** means the version was released in September 1993).

# Chapter 2

## Windows, Viewports and Clipping

---

This chapter describes the coordinate systems used in GPGS-F:

- **user coordinate system**, the coordinate system the user defines his drawing in.
- **window coordinate system**, transformed user coordinates.
- **normalized device coordinates (NDC)**, device independent coordinates for referring to the display surface of a device.

Transformations are mentioned in this chapter, but are described in detail in **Chapter 6**.

### 2.1 Window - Viewport Mapping

The major purpose of designing a device-independent graphic system, is to allow the application programmer to define his drawing in a coordinate system that is independent of the actual graphic device that is to be used.

#### 2.1.1 Window Definition

The coordinates used to define a GPGS-F drawing are called **user coordinates**. There are no limitations on the values that may be used, as long as they may be represented as single precision real numbers, which is the storage type used to store coordinate values in GPGS-F.

As the user coordinates are received by GPGS-F, they are transformed by the system transformation matrix (see **Chapter 6**) into **window coordinates**. If no transformations are specified, user and window coordinates will be identical.

The window coordinates are then transferred to the device driver in use. To make the driver able to actually display the primitives described, it must know how to map the received coordinates onto to display surface. This mapping is set by defining a **window**, a rectangular area (or a box if 3D) within the window coordinate system, and a corresponding area/box on the display surface where this window is to appear.

A window is defined by

```
CALL WINDW (W2arr(1))
```

where **W2arr** is a 4 element array describing a 2-dimensional window, or by

```
CALL WINDW3 (W3arr(1))
```

where **W3arr** is a 6 element array describing a 3-dimensional window. Within the arrays, the window limits are given in the sequence [**Xlow**, **Xhigh**, **Ylow**, **Yhigh** (**Zlow**, **Zhigh**)].

The window definition may be changed at any time within an application. GPGS-F will however store the last definition only, i.e. if an application wants to switch between different window settings, the values must be kept by the application itself.

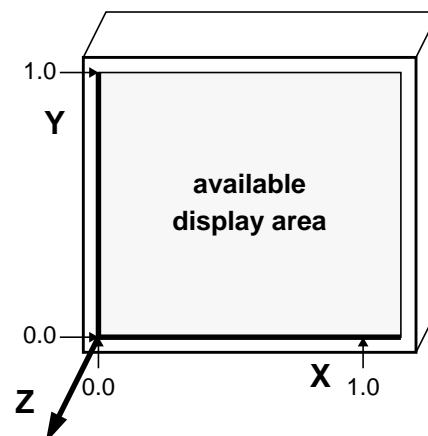
## 2.1.2 Viewport Definition

In order to address the display surface without referring to the physical device coordinates, **normalized device coordinates** are used. This coordinate system is defined by assigning the coordinate range 0.0 - 1.0 to the largest square area of the display. The position [0.0, 0.0] is defined to be at the lower left-hand corner of the display. Thus the point [0.0, 0.5] will be half way up the left edge of a square display area, and [0.5, 0.5] will be at the centre.

With non-square devices the largest possible square (or cubic volume) of the display surface is assigned the range 0.0 to 1.0. Whichever coordinate may be longer then use values greater than 1.0 to address the part of the display area outside the square.

The NDC system is a normal right-handed coordinate system, with the X axis running horizontally and the Y axis vertically. The Z axis points from the display surface towards the viewer, with 0.0 being at the display surface.

**Figure 2.1** *Normalized Device Coordinates.*





The area within the NDC space where the window is to be displayed, is called the **viewport**. This is defined by

**CALL VPORT (V2arr(1))**

for a 2-dimensional viewport, and

**CALL VPORT3 (V3arr(1))**

for a 3-dimensional viewport. **V2arr** is a 4 element array and **V3arr** is a 6 element array. The viewport limits are given in the same sequence as the window, i.e. [**Xlow**, **Xhigh**, **Ylow**, **Yhigh** (**Zlow**, **Zhigh**)].

As with the window, the viewport may be changed as many times as required, but *only when there is no picture segment open* (picture segments are introduced in **Chapter 3**).

Programmers should be careful that their windows and corresponding viewports have the same aspect ratio. If not, the mapping of coordinates onto the display will produce an unwanted (in most cases) scaling effect.

A complete program example using windows and viewports is shown on **page 4-6**. The example also shows how window settings are used to get a zooming effect.

### 2.1.3 Default Values

GPGS-F will define a default window and viewport, ranging from 0.0 to 1.0 in all three directions, when initialized. If no transformation is applied, user coordinates and normalized device coordinates are then identical.

### 2.1.4 Window and Viewport Dimensions

When 2-dimensional windows and viewports are defined, by *WINDOW* and *VPORT*, the Z axis range of either defaults to [0.0 - 1.0]. For most GPGS-F applications, this is of little significance. However, when clipping is enabled (see next page), the use of *WINDOW* indicates that only 2-dimensional clipping should take place. Whenever 3-dimensional clipping is wanted, *WINDOW3* must be used to specify the clipping limits along the Z axis.

The reason for *VPORT3* and 3-dimensional viewports, is to specify the mapping of user Z coordinates into the NDC space. The Z component of the normalized device coordinate is used for computing the intensity value when the device supports intensity depth modulation (**page 13-2**), and to make it possible to transform NDC values back to window and user coordinates (**page 8-14**).

## 2.2 Clipping

If a user coordinate outside the window limits is specified, this will be mapped to a point outside the viewport limits, and maybe also outside the display area of the output device. In most cases this is not wanted, and such coordinates should not be part of the display.

To ensure that only parts of the drawing that are inside the window will be part of the display, lines and other primitives must be **clipped**.

Clipping a line means that the coordinates of the endpoints are compared with the boundaries of the window, and the parts of the line that fall outside the window are not displayed.

Clipping of coordinates is controlled by

**CALL CLICTL (lswtch)**

where **lswtch**=1 enables, and **lswtch**=0 disables clipping.

As GPGS-F is initialized, clipping is disabled.

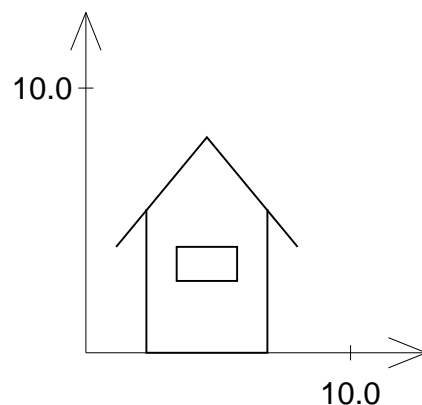
When perspective projection is selected by setting the eye position (see **page 6-12**), the clipping algorithm will ensure that points behind the eye will not be displayed.

Clipping could be left disabled if an application knows that all coordinates will be within the window. The result will of course be correct if clipping is enabled, but this will involve quite a lot of 'unnecessary' computations within GPGS-F.

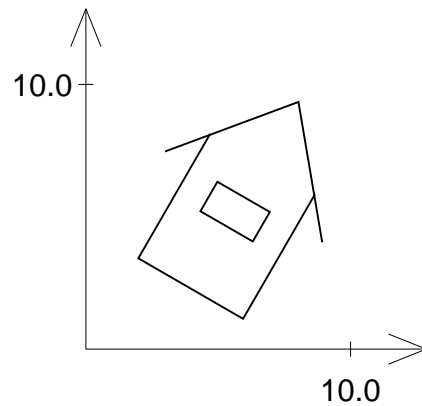
## 2.3 Coordinate Processing

The coordinates passed from the application program are processed by GPGS-F before being displayed on the graphic device. In brief, this processing is made up by the following steps:

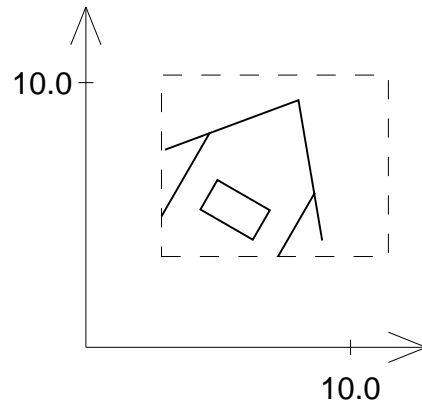
1. The user coordinates are received by GPGS-F.



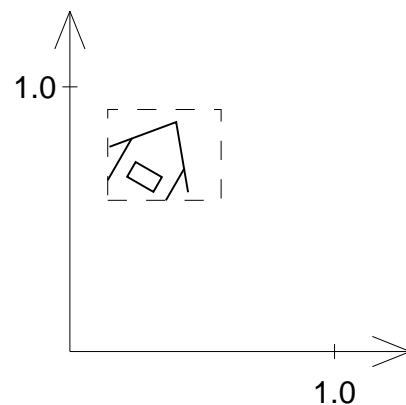
2. The coordinates are transformed by the GPGS-F transformation matrix (see **Chapter 6**) into window coordinates.



3. If clipping is enabled, the primitives are clipped by the specified window. The clipped coordinates are passed to the driver.



4. The device driver will map the coordinates to NDC according to the specified viewport. Finally, the NDC coordinates are mapped to the physical coordinates used by the device and displayed.





## Chapter 3

# Introduction to Picture Segments

---

A complete GPGS-F picture is a collection of basic graphic primitives such as lines, circular arcs, text strings etc. GPGS-F requires that these primitives are grouped into **picture segments**.

There are no limits on the size of a picture segment, and GPGS-F has no rules on how to split a picture into segments. Thus, applications that just present a picture on the display, may define the complete picture as a single picture segment.

To programmers wanting to make more advanced applications, GPGS-F does however provide features like dynamic picture modifications and picture storing for later reuse (introduced in **Chapter 14**). When using these features, the picture segment is the smallest unit that may be referenced by the application program.

A picture segment is defined and opened by

**CALL BGNPIC (Ident)**

where **Ident** is the picture segment identifier. The identifier is stored within GPGS-F as a Fortran integer type, together with a 4-bit status word. Thus, on computers using 16 bits integers, the value must be between 1 and 4095, on computers using 32 bits integers the allowed range is 1 to 268435455.

If segment storing is used, there are some restrictions on whether the same identifier may be used more than once within an application. These restrictions are given on **page 14-2**.

If segments are not stored, there are no such restrictions.

When a segment is completed, it must be closed by

**CALL ENDPIC**

Some GPGS-F routines, such as all routines generating graphic output, require that a segment is opened when called. Other routines, such as the device handling routines described in **Chapter 1**, are *not* allowed to be called when a segment is open. If a routine is called when not allowed, an error message is generated.



# Chapter 4

## Basic Graphic Primitives

GPGS-F provides routines for drawing all kinds of basic graphic primitives (also called graphic elements, or picture elements). This chapter describes the routines for drawing single lines, circular arcs, elliptic arcs and markers (symbols). The other primitives available with GPGS-F are described in **Chapter 7** (text), **Chapter 10** (polylines and curves) and **Chapter 12** (polygons and pixel arrays).

All graphic primitives except elliptic arcs are related to the *current position*. The current position is defined to be the position of the (logical) pen during drawing. Note that when a new picture segment is opened, the current position is undefined.

2- and 3-dimensional primitives may be mixed. 2D primitives will then be drawn in the XY plane at the current, i.e. the last specified, Z coordinate.

### 4.1 Linetypes

With the routines for drawing single lines, circular arcs, elliptic arcs, polylines and curves, a linetype is included as one of the arguments. The number of available linetypes is device dependent, and is specified with the driver descriptions in **Appendix E**.

**Figure 4.1** *GPGS-F linetypes.*

Invisible line	0	
Solid line	1	_____
Endpoint	2	_____.
Dotted line	3	.....
Dashed line	4	- - - - -
Dash-dot line	5	- . - . - . - . - . - . - . - .
Linetypes 6 to 9 as they are defined by the PostScript driver	6	- - - - -
	7	- . - . - . - . - . - . - . - .
	8	- - - - -
	9	- - - - -

Linetypes 1 to 5 are standardized by GPGS-F, and will produce the same linepattern with all devices. The patterns obtained by linetypes 6 and higher are selected by the device drivers, and will not necessarily be the same on different devices. If the device in use does not support the specified linetype, a solid line will be drawn.

Linetype 0 is used to move the current position. Although 0 may be specified with any primitive, it is normally used only with the line drawing routines.

In addition to the predefined linetypes, GPGS-F contains a module allowing the application to define its own linetypes. This module is described in **Chapter 9**.

### 4.1.1 Linepattern Length

To distinguish separate parts of a drawing from each other, using different colours (see **Chapter 11**) or linewidths (see **Chapter 13**) are the preferred methods. If these facilities are not available, using different linetypes is a obvious choice.

To give the application programmer further possibilities, some device drivers allow the length of a single pattern segment to be specified by

**CALL LPSCAL (Scale)**

where **Scale** is a scaling factor to be applied to the default length defined by the driver. Which drivers provide this feature is given in **Appendix E**.

The scaling factor is reset to its default value (1.0) when a picture segment is opened.

## 4.2 Drawing Single Lines

GPGS-F provides routines for drawing lines in 2 or 3 dimensions, based on floating or integer coordinates, given either relative to the current position or as absolute values.

The following routines are available:

**CALL LINE (X, Y, Ivis)**

**CALL LINE3 (X, Y, Z, Ivis)**

**CALL LINER (Dx, Dy, Ivis)**

**CALL LINER3 (Dx, Dy, Dz, Ivis)**

**CALL LINI (lx, ly, Ivis)**

**CALL LINI3 (lx, ly, lz, Ivis)**

**CALL LINIR (ldx, ldy, Ivis)**

**CALL LINIR3 (ldx, ldy, ldz, Ivis)**



The arguments supplied specify the endpoint of the line, while the starting point is the current position. After the line is drawn, the current position is updated to the given endpoint.

With the *LINE..* routines, coordinates are given as floating point numbers, either absolute (**X, Y, Z**), or relative to the current position (**Dx, Dy, Dz**).

With the *LINL..* routines, coordinates are given as integers, with the same choice of absolute (**lx, ly, lz**) or relative (**ldx, ldy, ldz**) values.

As shown, the routine names are suffixed with an **R** to mark the relative routines, and/or a **3** to mark the 3-dimensional routines.

Integer coordinates are converted to floating point values by GPGS-F before being processed, i.e. the execution time will not be improved by using *LINL..* routines instead of *LINE..* routines.

The **lvis** argument specifies the linetype to use, as described on **page 4-1**.

Linetype 0 is used to move the current position, not only to set the startpoint for lines, but also to specify the start or reference point for other graphic primitives.

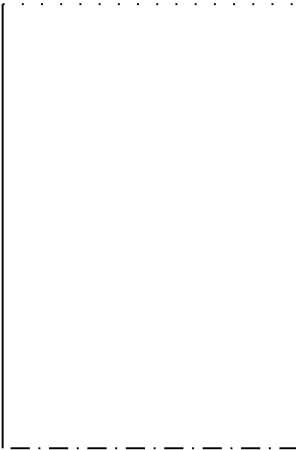
**Example 4.1**     ***Making a basic 2D drawing from coordinates read from the user's terminal.***

```

C  *****
C  COMPLETE WORKING EXAMPLE
C  *****
C
C  Default values are used for window and viewport
C
C  Initialize the device
C
      PRINT *, 'Type GPGS-F device number:'
      READ *, IDEV
      CALL GPGS
      CALL NITDEV(IDEV)
C
C  Open a picture segment
C
      CALL BGNPIC(1)
C
1000 CONTINUE
C
C  Read coordinates and linetype.
C  A negative linetype marks exit.
C
      READ *, X, Y, IVIS
      IF (IVIS .GE. 0) THEN
          CALL LINE(X,Y,IVIS)
          CALL UPDAT(0)
          GO TO 1000
      ENDIF
C
C  Close the picture segment and release the device
C
      CALL ENDPIC
      CALL RLSDEV(IDEV)
      END

```

**Figure 4.2**     ***Running Example 4.1 with a given data set.***

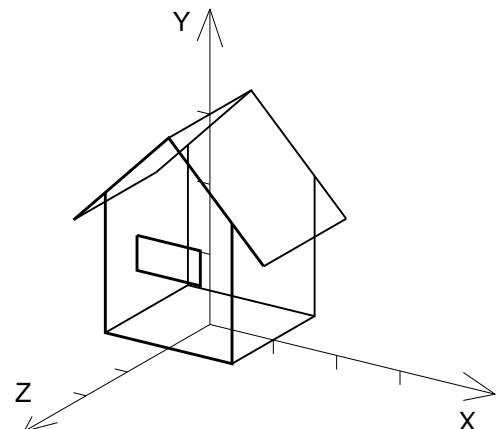
Supplied values	Comments	
72	device number	
0.2, 0.2, 0	move to lower left corner of figure	
0.2, 0.5, 1	left edge, solid line	
0.4, 0.5, 3	top edge, dotted	
0.4, 0.2, 4	right edge, dashed	
0.2, 0.2, 5	bottom edge, dash-dot	
0.0, 0.0, -1	to exit	

### Example 4.2    *Defining a 3 dimensional house.*

```

SUBROUTINE HOUSE
C
C Draw house with centre of ground floor at origin.
C
      DIMENSION DZ(2)
      DATA DZ/1.0, -2.0/
C
C Move current position to origin.
C
      CALL LINE3(0.0, 0.0, 0.0, 0)
C
C Draw front and back walls using relative lines.
C
      DO 1000 I=1,2
        CALL LINER3(-1.5,1.5,DZ(I),0)
        CALL LINER( 1.5, 1.5, 1)
        CALL LINER( 1.5,-1.5, 1)
        CALL LINER(-0.5, 0.5, 0)
        CALL LINER( 0.0,-2.0, 1)
        CALL LINER(-2.0, 0.0, 1)
        CALL LINER( 0.0, 2.0, 1)
C
C Add a 'window' on the front wall.
C
        IF (I .EQ. 1) THEN
          CALL LINER( 0.5,-0.5, 0)
          CALL LINER( 1.0, 0.0, 1)
          CALL LINER( 0.0,-0.5, 1)
          CALL LINER(-1.0, 0.0, 1)
          CALL LINER( 0.0, 0.5, 1)
          CALL LINER( 0.5,-1.5, 0)
        ELSE
          CALL LINER( 1.0,-2.0, 0)
        ENDIF
      1000 CONTINUE
C
C Draw connecting lines
C
      CALL LINER (-1.5, 1.5, 0)
      CALL LINER3( 0.0, 0.0, 2.0, 1)
      CALL LINER ( 1.5, 1.5, 0)
      CALL LINER3( 0.0, 0.0,-2.0, 1)
      CALL LINER ( 1.5,-1.5, 0)
      CALL LINER3( 0.0, 0.0, 2.0, 1)
      CALL LINER (-0.5,-1.5, 0)
      CALL LINER3( 0.0, 0.0,-2.0, 1)
      CALL LINER (-2.0, 0.0, 0)
      CALL LINER3( 0.0, 0.0, 2.0, 1)
C
      RETURN
      END

```



**Figure 4.3    3D house as defined in Example 4.2**

### **Example 4.3**     *Drawing the same object in different viewports.*

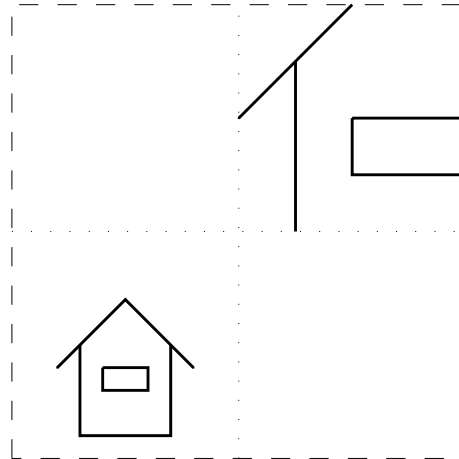
```

C *****
C COMPLETE WORKING EXAMPLE, combined with the previous example.
C *****
C
C      DIMENSION WDW1(6), WDW2(6), VP(4)
C      DATA WDW1/-2.5,2.5, -0.5,4.5, -2.5,2.5/
C      DATA WDW2/-1.5,0.5,  0.5,2.5, -0.5,1.5/
C
C Initialize a device.
C
C      READ *, IDEV
C      CALL GPGS
C      CALL NITDEV(IDEV)
C
C specify a window to contain the complete house.
C
C      CALL WINDW3(WDW1)
C
C Set the viewport to the lower left quarter of the display.
C
C      VP(1)=0.0
C      VP(2)=0.5
C      VP(3)=0.0
C      VP(4)=0.5
C      CALL VPORT(VP)
C
C Draw the house, using the subroutine of the previous example.
C
C      CALL BGNPIC(1)
C      CALL HOUSE
C      CALL ENDPIC
C
C Now look at detail in the upper right hand corner of
C the screen. Set window to contain the detail only.
C
C      CALL WINDW3(WDW2)
C
C Change viewport to upper right quarter of the display.
C
C      DO 1000 I=1,4
C          VP(I)=VP(I)+0.5
C 1000 CONTINUE
C      CALL VPORT(VP)
C
C Switch clipping on.
C
C      CALL CLICTL(1)
C
C Draw the house. All but the detail will be clipped away.
C
C      CALL BGNPIC(2)
C      CALL HOUSE
C      CALL ENDPIC
C
C Release the driver
C
C      CALL RLSDEV(IDEV)
C      END

```

**Figure 4.4** *Display of object in different viewports, using Example 4.3*

The dashed square indicate the default viewport.  
The dotted lines indicate the size of the specified viewports.



## 4.3 Circular Arcs

To create circular arcs in 2 or 3 dimensions, the following routines are available

**CALL CIRC (Xc, Yc, Angle, Ivis)**

**CALL CIRCR (Dxc, Dyc, Angle, Ivis)**

**CALL CIRC3 (Xc, Yc, Zc, Xp, Yp, Zp, Angle, Ivis)**

**CALL CIRCR3 (Dxc,Dyc,Dzc, Dxp,Dyp,Dzp, Angle, Ivis)**

**CALL CIRD (Xc, Yc, Dangle, Ivis)**

**CALL CIRDR (Dxc, Dyc, Dangle, Ivis)**

**CALL CIRD3 (Xc, Yc, Zc, Xp, Yp, Zp, Dangle, Ivis)**

**CALL CIRDR3 (Dxc,Dyc,Dzc, Dxp,Dyp,Dzp, Dangle, Ivis)**

As with the line drawing routines, the routine names are suffixed with an **R** to mark the relative routines, and/or a **3** to mark the 3-dimensional routines.

Arcs are defined by the current position, which marks the startpoint of the arc, the centre, and the angle of the arc. After drawing an arc, the current position is updated to the endpoint of the arc.

**Xc, Yc, Zc** specify the coordinates of the centre of the arc as absolute coordinates, while **Dxc, Dyc, Dzc** specify the centre relative to the current position.

With a 3-dimensional arc, the plane in which the arc is to be drawn is defined by the current position, the centre of the arc, and an additional third point specified by **Xp, Yp, Zp** (absolute coordinates) or **Dxp, Dyp, Dzp** (relative to the current position). Specifying this third point to lie on the (extended) line between the startpoint and the centre will not define a plane, and will result in a GPGS-F error message.

**Ivis** specifies the linetype to use for drawing the arc, as described on **page 4-1**.

With the **CIRC..** routines, **Angle** specifies the angle of the arc as radians, with the **CIRD..** routines the angle is specified as degrees through **Dangle**.

For 2D arcs, a positive angle is defined to be counterclockwise direction in the XY plane when seen from the positive Z axis, as illustrated by the example below.

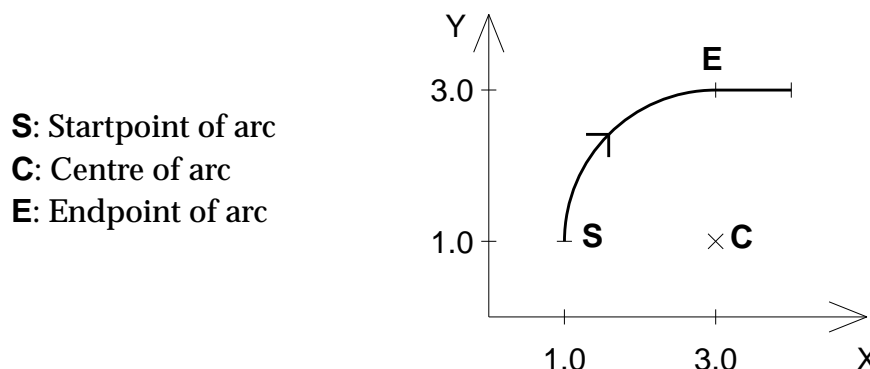
#### **Example 4.4**     *Drawing a 2D circle arc.*

```

C
C  Move to startpoint of arc.
    CALL LINE(1.0, 1.0, 0)
C
C  Draw a 90 degree clockwise arc, with centre defined relative
C  to the current position.
    CALL CIRD(2.0, 0.0, -90.0, 1)
C
C  Draw a horizontal line from the endpoint of the arc,
C  just to show that the current position has been updated.
    CALL LINER(1.0, 0.0, 1)

```

**Figure 4.5**     *2D circle arc (defined by Example 4.4).*



For 3D arcs, positive angle is defined to be counterclockwise direction in the plane of the arc when seen from the positive Z axis of a *local* right hand coordinate system. This coordinate system is defined by the line from the centre of the arc to the current position (X axis), and the line from the centre pointing in the direction of the third point (Y axis). The direction of the Z axis then follows from the right hand rule.

This means that the position of the additional point given does not only define the plane for the arc, but also the direction the arc is to be drawn.

#### **Example 4.5**     *Drawing circle arcs in the XZ plane.*

```

C
C  Move to startpoint of arc.
C      CALL LINE3 (1.0, 1.0, 0.0, 0)
C
C  Draw a 90 degree solid counterclockwise arc.
C  The centre and the third point are given as absolute values.
C
C      CALL CIRD3 (1.0, 1.0, 4.0, 60, 1.0, 4.0, 90.0, 1)
C
C  Using the same startpoint and centre, but a different
C  third point, draw a 60 degree dashed counterclockwise arc.
C
C      CALL LINE3 (1.0, 1.0, 0.0, 0)
C      CALL CIRD3 (1.0, 1.0, 4.0, -1.0, 1.0, 4.0, 60.0, 1)

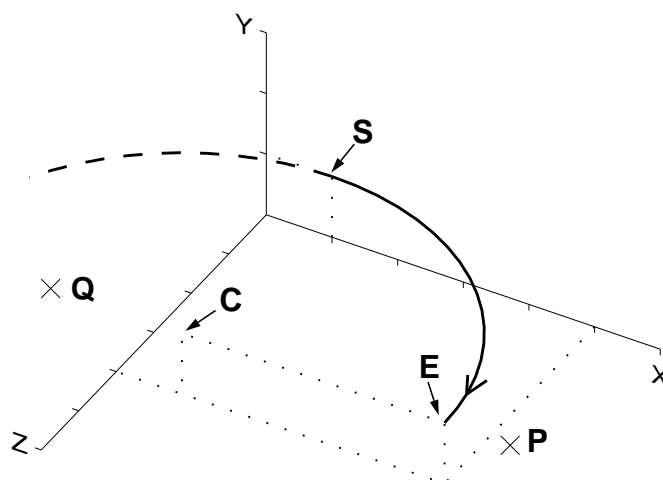
```

#### **Figure 4.6**     *3D circle arc (defined by Example 4.5).*

The tick marks on the axes are placed at 1.0, 2.0 etc.

The dotted lines are added just as an attempt to show the 3D position of the start, centre and endpoint.

- S:** Startpoint of arcs
- C:** Centre of arcs
- E:** Endpoint of solid arc
- P:** Extra point defining plane of solid arc
- Q:** Extra point defining plane of dashed arc



Following from the rule for defining positive direction for 3D arcs, the 2 arcs are drawn in different directions.

For the solid arc, the local Z axis points 'down', for the dashed arc it points 'up'.

### 4.3.1 Software / Hardware Generation

By default, GPGS-F will generate arcs by using a sequence of short vectors, allowing 3D arcs to be drawn at any orientation.

If the device in use has got the ability to generate arcs by hardware, this will however in most cases be faster. Therefore, GPGS-F allows the user to select whether to use hardware or software generation by

**CALL SOFCIR (lsw)**

**lsw** = 0 (zero) selects hardware generation, **lsw** = 1 selects software.

When drawing 3D arcs, either by using the 3D circle routines, or drawing 2D arcs in a transformed XY plane, the projection onto the display surface will be an elliptic arc. Using hardware generation for such arcs will (with most devices) not give the correct result, as the device hardware is capable of drawing arcs only in the XY plane of the display.

### 4.3.2 Circle Smoothness

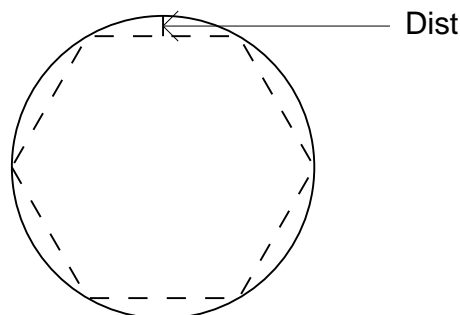
If the **lsw** argument to *SOFCIR* is set to a value greater than 1, the value specifies the number of vectors that form the complete circle of which the arc is a part.

A second method of specifying the smoothness of circles, is to set the circle approximation tolerance directly by

**CALL CIRAPR (Dist)**

**Dist** is the maximum distance, in user coordinates, between the mathematical correct circle and the approximated circle.

**Figure 4.7** *Effect of CIRAPR.*



If both *CIRAPR* and *SOFCIR* is used to set circle smoothness, GPGS-F will use the value supplied by the most recent call.



## 4.4 Elliptic Arcs

An ellipse or elliptic arc is drawn by

**CALL ELIP (Xcn, Ycn, Xrad, Yrad,  
Ang0, Angle, Rotang, Ivis)**

**Xcn** and **Ycn** specifies the centre, **Xrad** and **Yrad** the length of the radius in X and Y direction.

**Ang0** specifies the start angle of the arc, measured from the X axis of the ellipse. If a complete ellipse is to be drawn the value of **Ang0** is of no importance. **Angle** is the angle of the arc to draw, and **Rotang** is the rotation angle from the system X axis to the X axis of the ellipse. All angles are to be given as radians.

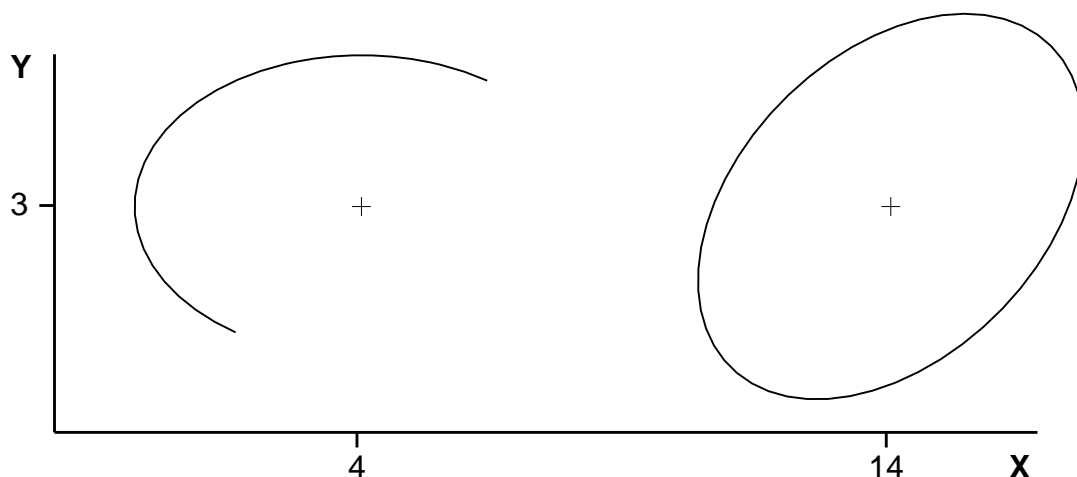
**Ivis** specifies the GPGS-F linetype to use for drawing. as described on [page 4-1](#).

Elliptic arcs are drawn in the XY plane at the current Z coordinate. The X and Y components of the current position are not used. After drawing, the current position is however updated to the end of the arc.

### **Example 4.6** *Drawing a elliptic arc and an ellipse.*

```
CALL ELIP( 7.0, 6.0, 3.0, 2.0, PI/4.0, PI, 0.0, 1)
CALL ELIP(14.0, 6.0, 3.0, 2.0, 0.0, 2.0*PI, PI/4.0, 1)
```

**Figure 4.8** *Elliptic arc and ellipse (defined by the example above).*



The arc (left) is defined by:

```
CALL ELIP( 4.0, 3.0, 3.0, 2.0, PI/4.0, PI, 0.0, 1)
```

The ellipse (right) is defined by:

```
CALL ELIP(11.0, 3.0, 3.0, 2.0, 0.0, 2.0*PI, PI/4.0, 1)
```

## 4.5 Markers

The purpose of markers is mainly to mark positions in the display surface. Because of this, markers are handled a bit different than other primitives by GPGS-F.

A marker (symbol) is drawn by

**CALL MARKER (Imar)**

where **Imar** specifies which marker to draw. The centre of the marker is placed at the current position, which is left unchanged afterwards.

The default size of a marker is determined by the device driver (normally 3-4 mm). The size may be changed by

**CALL MSIZE (Size)**

where **Size** gives the marker size in *Normalized Device Coordinates* (see [page 2-2](#)). The size may be reset to its default by setting **Size** to 0.0. The marker size is independent of the current window and viewport setting, and a given value of **Size** does not give the same physical size on different devices.

Setting the marker size to a given device independent physical size is still possible, using the same method as when making true scale plots. This is described in [Chapter 5](#).

The drawing of markers is left to the device driver. If possible, the markers are drawn by device hardware using some sort of symbol font. If not, the markers are software simulated within the driver, and will appear as shown in [Figure 4.9](#).

Markers are always drawn in the display plane, independent of the orientation of the user coordinate system, i.e. the markers are not affected by any of the modelling transformations described in [Chapter 6](#).

**Figure 4.9** *Standard software simulated markers.*



Markers are often used to mark points on curves and other business graphics type of plots. The fact that markers may appear different on different devices is then not very fortunate. However, the GRAPHISTO package (the business graphics add-on to GPGS-F) does contain its own set of markers to use for such plots.

## Chapter 5

# Drawing in True Scale

---

The window to viewport mapping method will, as described in **Chapter 2**, make the application programs independent of the physical size of the actual device in use. However, this obviously means that the physical size of the plot depends on the device. In some cases, this is not wanted, i.e. the size of a plot should be the same no matter what device is used.

To achieve this, the viewport limits must be calculated based on the size of the device. The physical dimensions of the current device is obtained by

**CALL DATDEV (Idum, 0, Farr(1), llthf)**

**Farr** will return **llthf** floating point numbers. **Farr(1)** and **Farr(2)** gives the extension in X and Y direction of the display surface, given in Normalized Device Coordinates. That is, these are the maximum values that may be used for specifying the viewport in X and Y direction. Following from the definition on **page 2-2**, one of these will always be 1.0, the other one will be larger.

**Farr(3)** gives the NDC extension in Z direction. For 2D devices, a value of 10000.0 will be returned, marking that any value may be used with the viewport specification.

Through **Farr(4)**, the physical length in *meters* of the default viewport size is returned, i.e. the length from 0.0 to 1.0 in normalized device coordinates.

**DATDEV** may also be used for getting a lot of other information about the current device driver. A complete description is given on **page 23-4**.

### **Example 5.1**     *Returned DATDEV data from a flatbed plotter.*

**Farr(1)** = 1.25     (NDC)  
**Farr(2)** = 1.0     (NDC)  
**Farr(3)** = 10000.0 (NDC), shows that this is a 2D device  
**Farr(4)** = 1.20     (meters)

These values show that the display surface of the plotter is 1.20 meters in the Y direction, and 1.50 meters (1.20 meters × 1.25) in the X direction.

Given the values returned from **DATDEV**, the NDC value **Vndc** corresponding to the physical size **Vm** (in metres) is simply computed as:

$$\mathbf{Vndc} = \mathbf{Vm} / \mathbf{Farr(4)}$$

### **Example 5.2**     *Drawing a map in scale 1:5000.*

#### **Purpose of the example:**

We want to plot a map in scale 1:5000, using the ground coordinates in meters as input.

A suitable map size is 64 by 48 cm, which gives room for an area of 3200 by 2400 meters. The actual coordinate area we want to map, is between 0.0 and 3200.0 in X, and between 2400.0 and 4800.0 in Y direction.

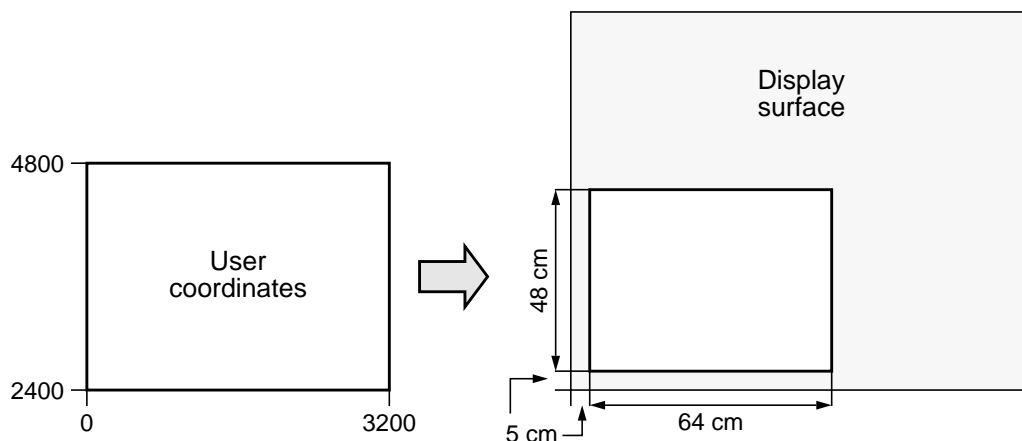
We want a margin of 5 cm below and to the left of the map.

#### **Program code:**

```

C
C  Declare necessary arrays, and set values in the window array.
      DIMENSION WDW(4),VP(4),FARR(4)
      DATA WDW/0.0,3200.0,2400.0,4800.0/
C
C  Initialize GPGS-F and the device driver.
      CALL GPGS
      CALL NITDEV (idev)
C
C  Set the window to 0-3200 in X, and 2400-4800 in Y direction.
      CALL WINDW (WDW)
C
C  Get physical dimensions of the device in use
      CALL DATDEV (IDUM, 0, FARR, 4)
C
C  Compute and set the viewport. In meters this is to be
C  from 5 to 5+64 cm in X direction,
C  from 5 to 5+48 cm in Y direction
      VP(1)=0.05 / FARR(4)
      VP(2)=VP(1) + 0.64 / FARR(4)
      VP(3)=0.05 / FARR(4)
      VP(4)=VP(3) + 0.48 / FARR(4)
      CALL VPORT(VP)
C
C  Ready to plot the map, using meters as user coordinates.

```



Some plotter packages address the display surface directly in physical units, e.g. millimetres or inches. Applications using such packages may easily be converted to using GPGS-F without having to convert the coordinates.

### **Example 5.3     *Addressing the display surface using millimetres.***

#### **Purpose of the example:**

We want to set up a window / viewport mapping that allows the application to address the display surface directly using millimetres as units.

If the plot is larger than the device in use, we want to show as much as possible of it.

#### **Program code:**

```
C
    DIMENSION WDW(4), VP(4), FARR(4)
C
C  Initialize GPGS-F and the device driver.
    CALL GPGS
    CALL NITDEV (idev)
C
C  Get physical dimensions of the device in use.
    CALL DATDEV (IDUM, 0, FARR, 4)
C
C  Specify a viewport covering the complete display surface.
    VP(1) = 0.0
    VP(2) = FARR(1)
    VP(3) = 0.0
    VP(4) = FARR(2)
    CALL VPORT(VP)
C
C  Specify a window that corresponds to the viewport limits,
C  converted to millimetres.
C
    WDW(1) = 0.0
    WDW(2) = FARR(1)*FARR(4)*1000.0
    WDW(3) = 0.0
    WDW(4) = FARR(2)*FARR(4)*1000.0
    CALL WINDOW (WDW)
C
C  Switch clipping on, in case the plot is larger than
C  the actual device being used.
    CALL CLICTL(1)
C
C  Ready to draw using millimetres as user units.
```



# Chapter 6

## Transformations

---

GPGS-F provides two groups of transformation routines. The first one, described in this chapter, is called modelling transformations. Modelling transformations are used by the application programs to model the objects.

The second group of transformations is called image transformations. These are used to manipulate the picture after it has been displayed. Image transformations are described in **Chapter 18**.

### 6.1 System Transformation Matrix

Each object, or part of an object, is defined in its own local coordinate system. This coordinate system is transformed into the window coordinate system by the GPGS-F system transformation matrix. Whenever a transformation routine is called, the system matrix will change.

Changing the matrix will not affect objects already displayed.

Initially, the system transformation matrix is set to the identity matrix, and may be reset to this setting at any time by

**CALL IDEN**

When the transformation matrix equals the identity matrix, user coordinates and window coordinates are equal.

## 6.2 Basic Transformation Routines

To illustrate the effect of the transformation routines, the *HOUSE* subroutine shown on page 4-5 is used. The initial user coordinate system is drawn with solid axes and marked **XYZ**. The transformed coordinate system is drawn with dashed lines and marked **X'Y'Z'**. When 2 transformations are illustrated by the same figure, the axes resulting from the first transformation is drawn dashed, the final axes are drawn as dash-dot lines.

As explained in **Chapter 2**, the Z axis initially points from the display surface towards the viewer. Thus, to make the Z axis visible in the figures, the *AXON* routine (see page 6-12) is called prior to the basic transformation routines shown with the individual figures.

### Example 6.1 Program used to illustrate transformations.

```

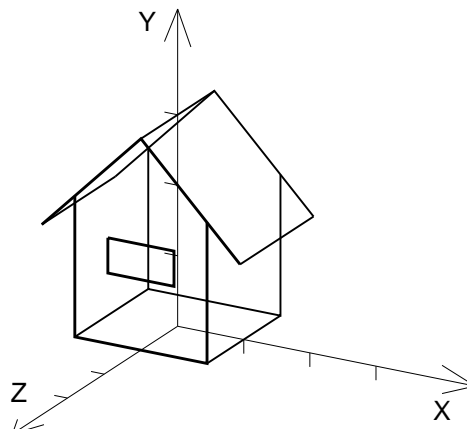
C
C Set the window limits. Symmetric around the origin.
C Large enough to contain the transformed house.
C
      REAL WDW(6)
      DATA WDW/-7.5,7.5, -7.5,7.5, -7.5,7.5/
C
      CALL GPGS
      CALL NITDEV(IDEV)
      CALL WINDW3(WDW)
C
      CALL BGNPIC(1)
      CALL AXON(5.0, 4.0, 9.0)
C
C TRANSF is one (or more) of the basic transformation routines.
C Each figure will show what routine(s) was actually used.
      CALL TRANSF()
C
      CALL HOUSE
      CALL ENDPIC
      CALL RLSDEV(IDEV)
C
C The code for drawing the axes is not included here.

```

**Figure 6.1** The object used to illustrate transformations.

The house as it will appear when no transformation routine is called.

The tick marks are placed at positions 1.0, 2.0 and 3.0 along the axes.





## 6.2.1 Translation

Translation means that the coordinates received by GPGS-F are shifted by the specified distances parallel to the coordinate axes. A 2-dimensional translation is specified by

**CALL XLAT (Xdisp, Ydisp)**

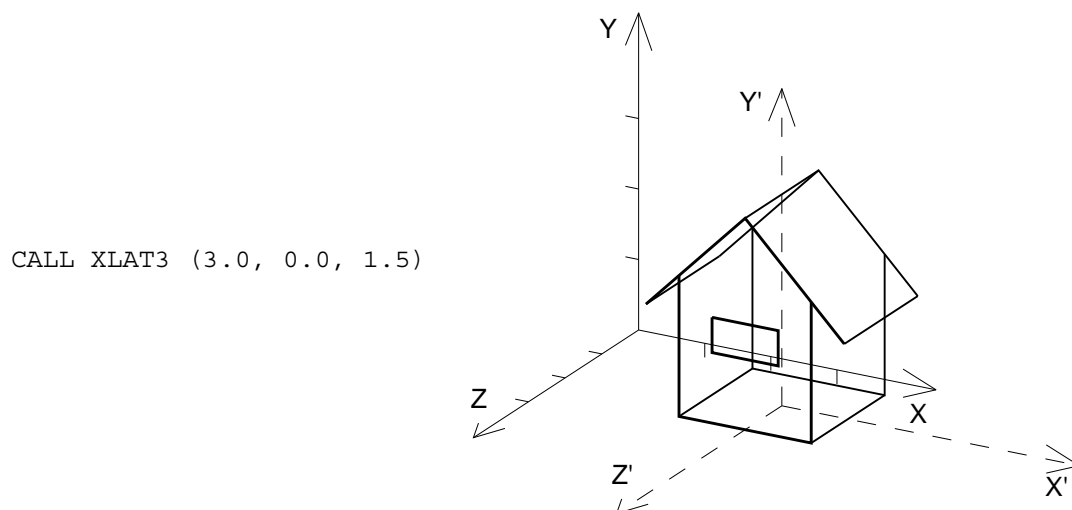
which will change each input X coordinate by **Xdisp** and each input Y coordinate by **Ydisp**.

The corresponding 3-dimensional routine is defined as

**CALL XLAT3 (Xdisp, Ydisp, Zdisp)**

adding **Zdisp** to each input Z coordinate.

**Figure 6.2** *3-dimensional translation.*



### 6.2.2 Scaling

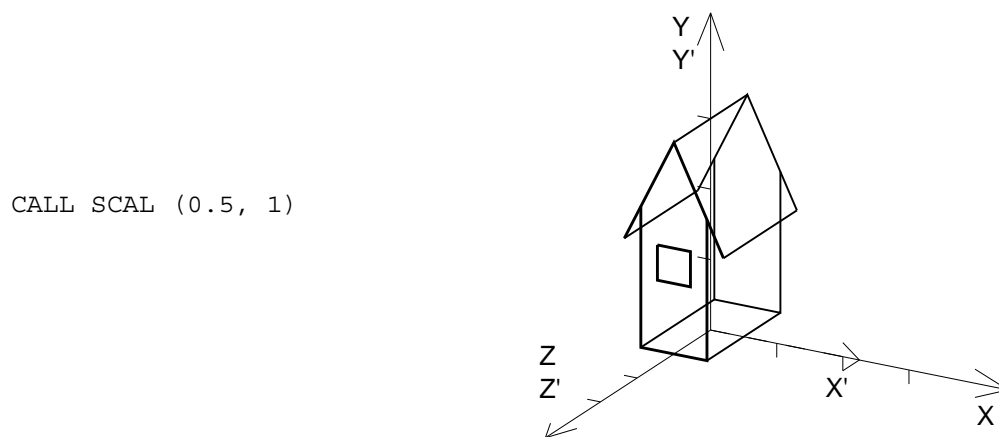
The input coordinates may be scaled by

**CALL SCAL (Scale, laxis)**

where **Scale** is the scaling factor to be applied to the coordinates along the **laxis** axis (1=X, 2=Y, 3=Z).

If **laxis** is set to 0 (zero), coordinates are scaled along all three axes.

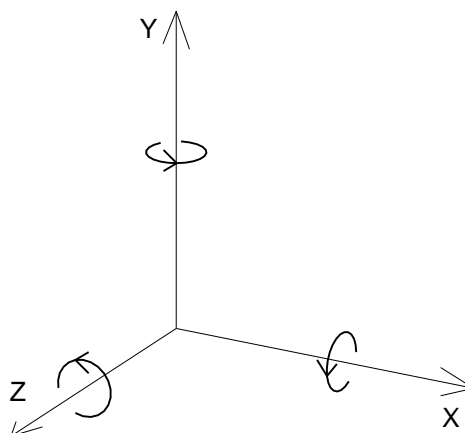
**Figure 6.3** *Scaling along the X axis.*



### 6.2.3 Rotation

To rotate the input coordinates around one of the axes of the coordinate system, the axis to rotate around and the angle to rotate must be specified. A positive angle of rotation is defined to be counterclockwise when looking towards the origin from a point on the positive part of the axis, as illustrated by the figure below.

**Figure 6.4** *Positive direction of rotation.*



Rotation is specified by

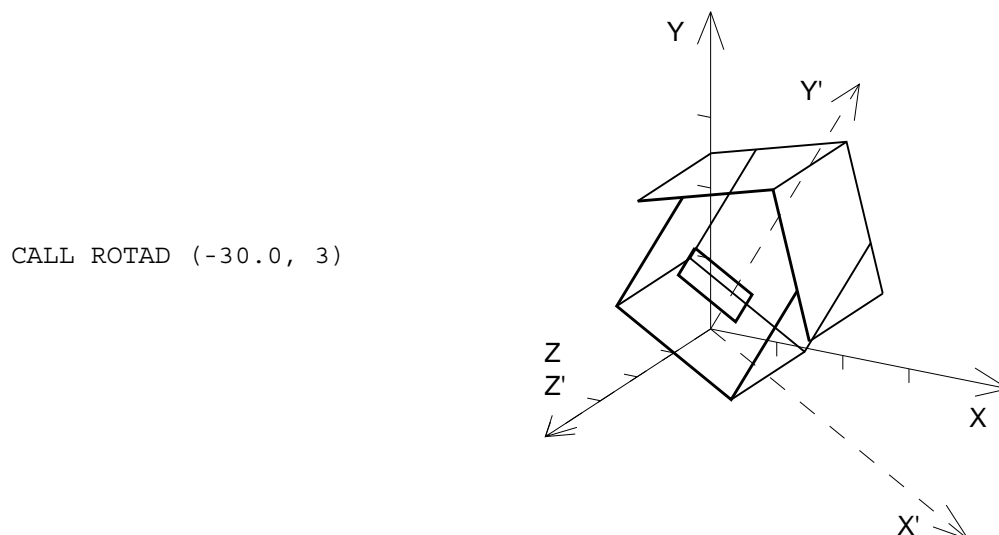
**CALL ROTA (Angle, laxis)**

where **Angle** is the rotation angle in radians, or by

**CALL ROTAD (Dangle, laxis)**

where **Dangle** is the angle in degrees. **laxis** specifies the axis to rotate around (1=X, 2=Y, 3=Z).

**Figure 6.5** *Clockwise rotation around the Z axis.*



There is no separate routine for 2D rotation, as this is equivalent to rotating around the Z axis, i.e. around the origin of the XY plane.

A 2D rotation is often used to change the orientation of a plot from landscape to portrait mode. In doing so, the position of the rotated plot will depend on the window limits. If the default window is used, and the only transformation applied is a 90 degree rotation, the complete plot will be rotated out of the window. To avoid this, the rotation must be combined with a translation.

Rotation around an arbitrary point in 3D space is achieved by using the sequence:

```
CALL XLAT3 (DX, DY, DZ)
CALL ROTAD (Dangle, laxis)
CALL XLAT3 (-DX, -DY, -DZ)
```

## 6.2.4 Shearing

Applying a shearing transformation to a point means that the value along one axis is changed by adding the value along one of the other axes multiplied by a constant value (the shearing factor).

Shearing is specified by

```
CALL SHEA (Shear, laxis1, laxis2)
```

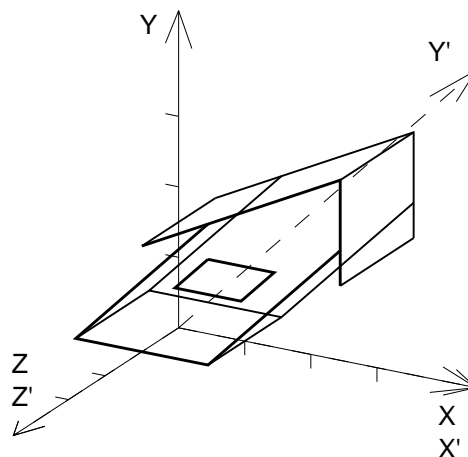
defining the new values of coordinates along the **laxis1** axis to be computed by

$$\text{Coord}(\mathbf{laxis1}) = \text{Coord}(\mathbf{laxis1}) + \mathbf{Shear} \times \text{Coord}(\mathbf{laxis2})$$

**laxis1** and **laxis2** are defined as for the other transformation routines (1=X, 2=Y, 3=Z).

**Figure 6.6** *Shearing the Y axis in X direction.*

```
CALL SHEA (1.0, 1, 2)
```



Shearing is not very commonly used, except when drawing text. In that case, the *SHEA* routine is however not necessary, as there is a separate routine available to specify shearing of text only (see [page 7-5](#)).

### 6.2.5 Vanishing Point

Using the basic transformation routines described so far, will produce an axonometric projection of the object into the display plane. This means that one user unit will have the same physical length no matter how far from the eye the object is.

In many cases, depending on what kind of object is to be drawn, using a perspective projection will give a more realistic image. Perspective projection may also be used to emphasize the 3D effect.

The normal method for specifying a perspective projection with GPGS-F is to set the eye position by using the *PERS* routine (see page 6-12).

GPGS-F does however allow the *vanishing point* for a perspective transformation to be explicitly set by

**CALL VANS (Xvan, Yvan, Zvan)**

where **Xvan**, **Yvan** and **Zvan** are the reciprocal values of the coordinates for the vanishing point.

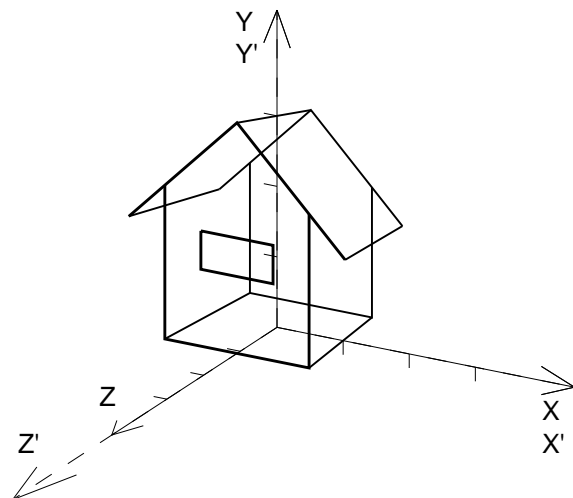
The effect of setting a vanishing point, is that lines defined to be parallel to the line passing through origin and the specified vanishing point, will be projected so that they will meet at the given point.

Initially the vanishing point is at infinity, which is equal to setting all three arguments to zero (zero is defined to be the reciprocal of infinity).

**Figure 6.7** *Vanishing point.*

Specify a vanishing point on the negative Z axis.

```
CALL VANS (0.0, 0.0, -1.0/12.0)
```



When the *PERS* routine is used to specify perspective projection, the vanishing point will be set by GPGS-F (see page 6-16).

## 6.3 Combining Basic Transformations

To achieve the required effect, basic transformations may be called in any sequence. The effect of transformations is cumulative, the final transformation being effected by all previous transformations.

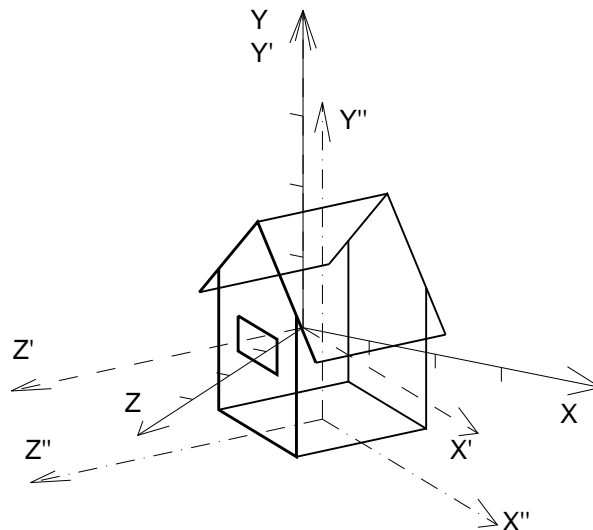
By default, transformations are performed in space mode (detailed description on **page 6-16**), i.e. new transformations are post-multiplied with the old system matrix to build the new system matrix.

The effect of this is that when a transformation routine is called, the specified transformation is relative to the current transformed coordinate system. Thus, the visual effect will change if the transformation sequence is changed.

**Figure 6.8** *Combining basic transformations, rotate and shift.*

```
CALL ROTAD(-30.0, 2)
CALL XLAT3(3.0, 0.0, 1.5)
```

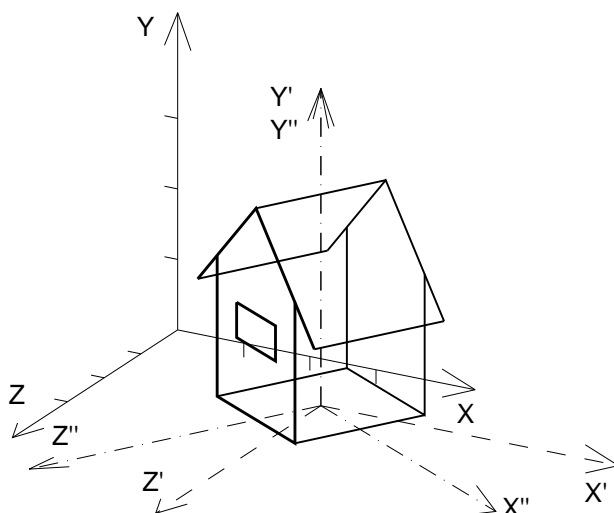
The rotation is relative to the initial axes (drawn solid). The translation is then relative to the rotated axes (drawn dashed).



**Figure 6.9** *Combining basic transformations, shift and rotate.*

```
CALL XLAT3(3.0, 0.0, 1.5)
CALL ROTAD(-30.0, 2)
```

As shown, the orientation of the house is as in the figure above, but because of the changed transformation sequence, the position is not.



## 6.4 Transformation Matrix Manipulation

Pictures are frequently built up of picture parts, each of which must be translated and rotated before display. A picture part is a collection of graphic elements that form a recognizable object on a display.

Quite often, several picture parts require the same set of transformations. If the parts are not drawn in sequence, but intermixed with parts using other transformations, there is a need to return the transformation matrix to some previous state.

### 6.4.1 Internal Matrix Stack

To simplify the operation of returning the transformation matrix to a previous state, a facility for stacking transformation matrices inside GPGS-F is provided.

A copy of the current transformation matrix is put on the stack by

**CALL BGNTRN**

This does *not change* the transformation matrix.

The top matrix is restored from the stack, and made the current transformation matrix, by

**CALL ENDTRN**

This will remove the matrix from the top of the stack. A maximum of 10 matrices may be stored in the stack.

Note that *ENDTRN*, in addition to restoring the transformation matrix, will reset the transformation mode (see **page 6-16**) to the state it was when the corresponding *BGNTRN* was called.

#### **Example 6.2     *Draw an untransformed object within a transformed object.***

```
C
C   .
C   Set up transformations for transformed object.
C       CALL SetupTrans( )
C
C   Draw transformed object
C       CALL DrawObj1( )
C
C   Then draw an untransformed object.
C       CALL BGNTRN
C       CALL IDEN
C       CALL DrawObj2( )
C       CALL ENDTRN
C
C   Continue with same transformation as before BGNTRN.
```

## 6.4.2 Direct User Manipulation

As an alternative to saving transformation matrices on the GPGS-F stack, it is also possible to copy the matrix into an application declared  $4 \times 4$  array by

**CALL SAVTRN (Tmat(1,1))**

When using Fortran, **Tmat** must be declared by

```
REAL Tmat (4,4)
```

At some later time, this matrix may be copied back by

**CALL TRAN (Tmat(1,1))**

resetting the transformation to the state it was when *SAVTRN* was called. The advantage of using *SAVTRN* / *TRAN* instead of *BGNTRN* / *ENDTRN*, is that when several matrices are saved, they need not be restored in the same sequence. In addition, the same matrix may be copied back more than once.

The matrix supplied through *TRAN* need not necessarily be one that has been fetched by *SAVTRN*. Programmers familiar with the construction of  $4 \times 4$  homogeneous coordinate transformation matrices may well build this matrix directly.

To those wanting to see how GPGS-F describes a single transformation by a  $4 \times 4$  matrix, just examine the matrix returned by *SAVTRN*.

In addition to replacing the system matrix with an application supplied matrix using *TRAN*, GPGS-F allows the application matrix and the system matrix to be combined using

**CALL COMP (Tmat(1,1))**

This will pre- or post-multiply **Tmat** with the current system matrix according to the current transformation mode (see **page 6-16**).

The following example shows how to avoid execution of the same trigonometric operations when performing a sequence of rotations. The rotation matrix is saved, and later used for concatenating. With more complex transformations involved, using this approach may speed up execution time quite a lot.



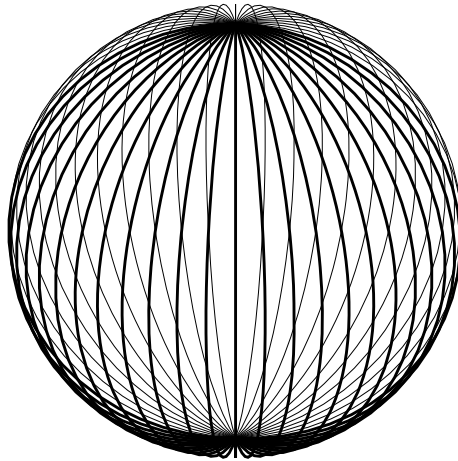
### **Example 6.3**     *Combining transformation matrices.*

```

C  ****
C  Complete working example
C  ****
C
      DIMENSION WDW(6), TMAT(4,4)
      DATA WDW/-5.0,5.0, -5.0,5.0, -5.0,5.0/
C
      PART=48.0
      IPART=INT(PART)/2
C
      READ *, IDEV
      CALL GPGS
      CALL NITDEV(IDEV)
      CALL WINDW3(WDW)
C
C  Save the matrix specifying a 360/48 degree rotation around Y.
C
      CALL ROTAD(360.0/PART,2)
      CALL SAVTRN(TMAT)
C
C  Reset matrix to identity, set eye position
C  (to see that it's really 3D).
C
      CALL IDEN
      CALL AXON(0.0, 4.0, 9.0)
      CALL BGNPIC(1)
C
C  Draw the 'front' of the globe, by drawing 180 degree arcs
C  in the XY plane.
C
      CALL LINWID(3.0)
      DO 1000 I=1,IPART
          CALL LINE(0.0, 3.0, 0)
          CALL CIRD(0.0, 0.0, 180.0, 1)
C
C  Apply the rotation by combining TMAT and the system matrix.
C
      CALL COMP(TMAT)
      1000 CONTINUE
C
C  Draw the back of the globe, using thinner lines.
C
      CALL LINWID(1.0)
      DO 1100 I=1,IPART
          CALL LINE(0.0,3.0,0)
          CALL CIRD(0.0,0.0,180.0,1)
          CALL COMP(TMAT)
      1100 CONTINUE
C
      CALL ENDPIC
      CALL RLSDEV(IDEV)
      END

```

**Figure 6.10** *Drawing produced by Example 6.3*



## 6.5 Viewing Routines

The basic transformation routines are well suited for building complex objects from smaller parts. Side or top views may also easily be specified by using these routines.

Using basic transformations to define how to see an object from an arbitrary point in 3D space, is however not very convenient. For this purpose, the GPGS-F viewing routines are easier to use.

There are two viewing routines available. These are

**CALL AXON (Xeye, Yeye, Zeye)**

to specify axonometric projection, and

**CALL PERS (Xeye, Yeye, Zeye)**

to specify perspective projection.

With *PERS*, **Xeye**, **Yeye** and **Zeye** are the coordinates of the eye position from which to view the objects. With *AXON*, the values specify the *direction* from the origin towards the eye, as the position of the eye (by definition) is at infinity.

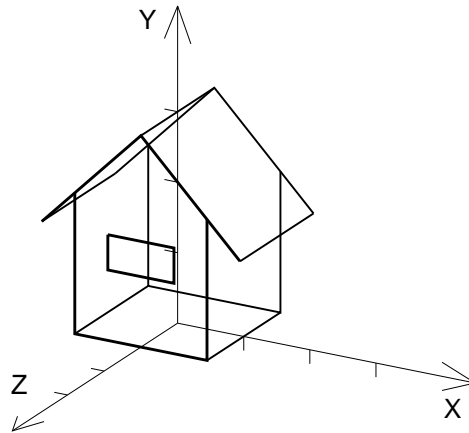
As with the other transformation routines, the values supplied through the viewing routines are relative to the current transformed coordinate system.

Although it is not illegal to use different eye positions for various picture parts, normally the same view is used for the complete picture. The most common approach is then to first specify the eye position, which will then be relative to the window origin, and later use the basic transformation routines for building the objects.

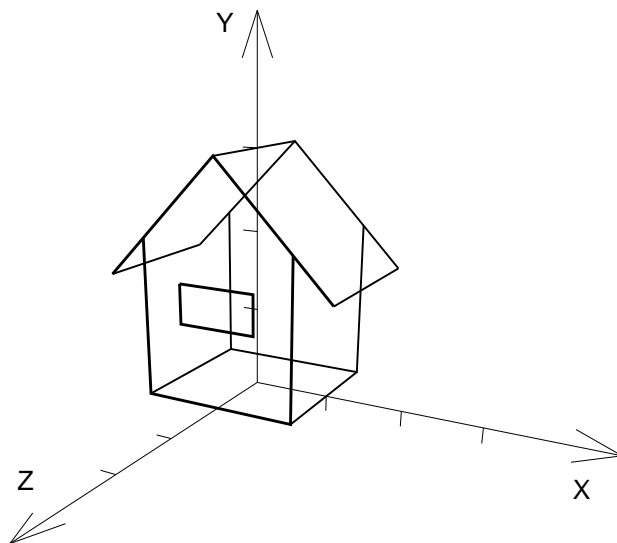
The initial viewing condition, set as GPGS-F is initialized or by a call to *IDEN* (see page 6-1), corresponds to calling *AXON* with **Xeye** and **Yeye** set to zero, and **Zeye** to a positive value, i.e. the eye is at infinity on the positive Z axis.

**Figure 6.11** *Axonometric and perspective projection.*

CALL AXON (5.0, 4.0, 9.0)



CALL PERS (5.0, 4.0, 9.0)



*AXON* and *PERS* implies rotations of the coordinate system. The new Z axis points from the origin towards the viewer, the new Y axis is kept upwards.

When specifying the window limits, these rotations must be considered. If, for example, the window limits in X direction are 0.0 and 10.0, with an object between 2.0 and 8.0, the object will be rotated completely out of the window if seen from behind (by specifying the eye position to be on the negative Z axis).

To make sure the objects stay within the window, no matter what eye position is used, the window should be symmetric around the origin, and the object defined with its centre at the origin.

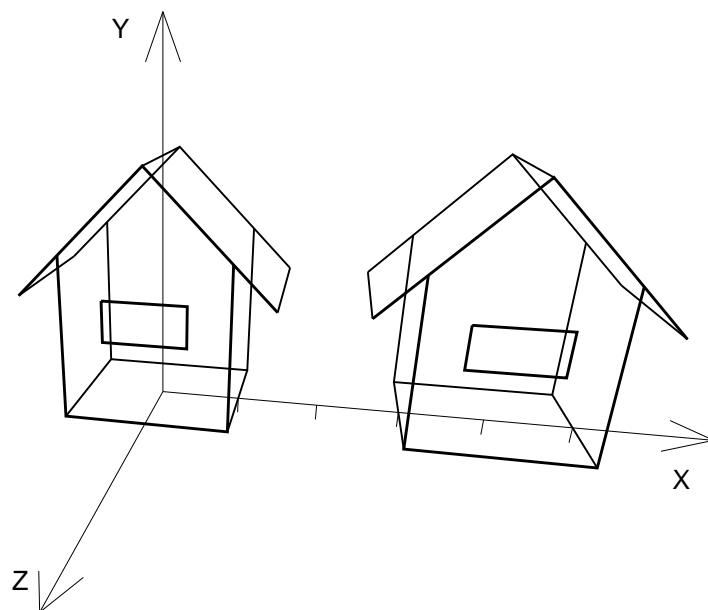
### 6.5.1 Focal Point

When issuing a call to *PERS* specifying the eye position, the focal point will be at the current origin. Thus, although there is no GPGS-F routine to set the focal point, this may still be specified by moving the origin prior to setting the eye position. To return the object back to its original position within the window, the origin should be translated back afterwards.

**Figure 6.12** *Setting the focal point.*

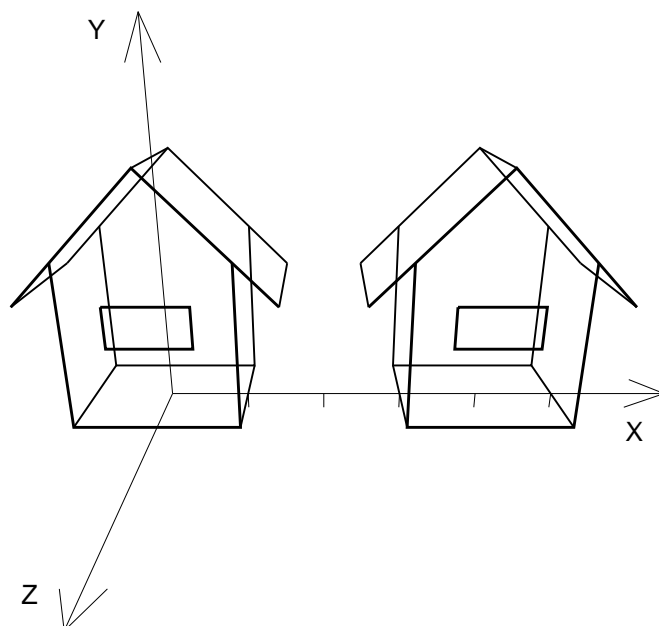
Draw 2 houses, one at origin and one moved 4 units along the X axis, using the default focal point.

```
CALL PERS(2.0, 4.0, 9.0)
CALL HOUSE
CALL XLAT(4.0, 0.0)
CALL HOUSE
```



Use the same drawing sequence as above, but move the origin to a point between the houses before setting the eye position.

```
CALL XLAT(2.0, 0.0)
CALL PERS(0.0, 4.0, 9.0)
CALL XLAT(-2.0, 0.0)
CALL HOUSE
CALL XLAT(4.0, 0.0)
CALL HOUSE
```

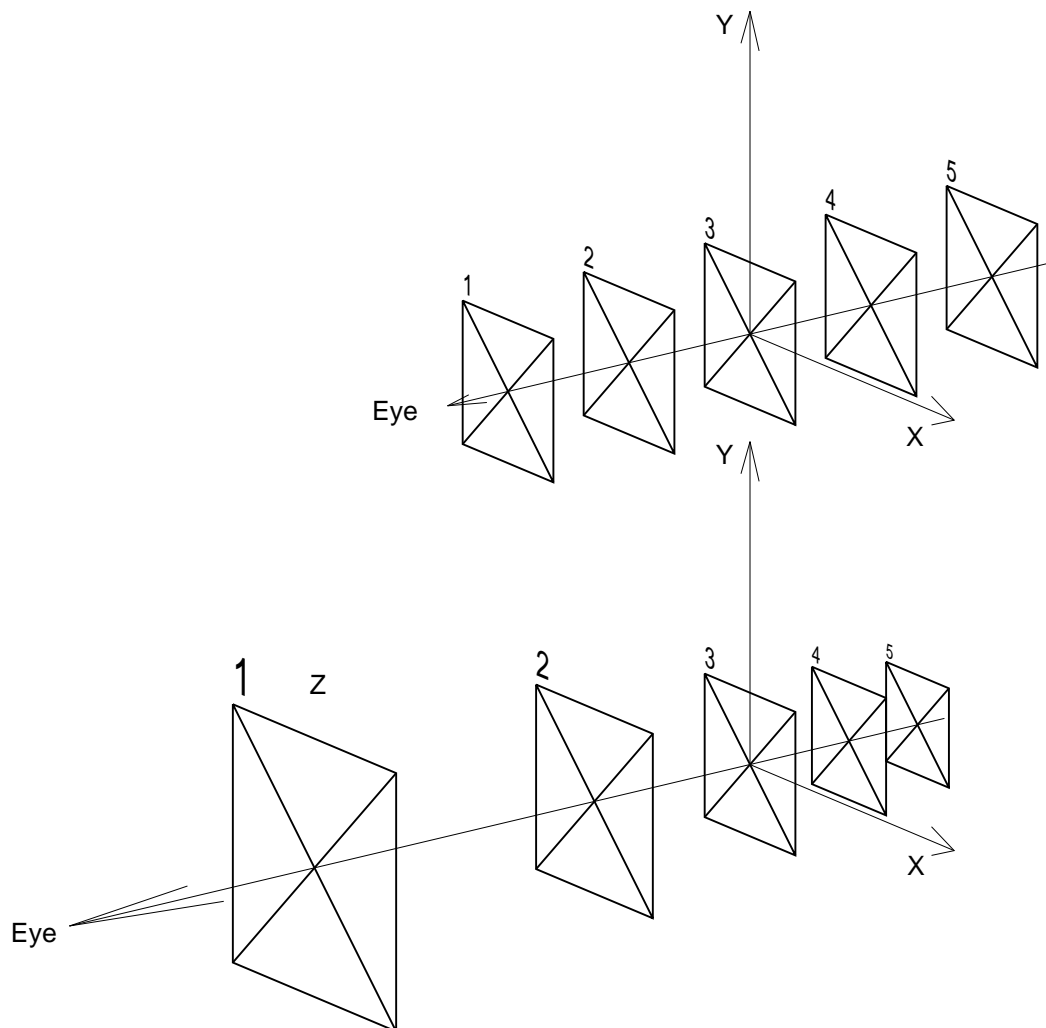


The eye position, relative to the *window origin*, is the same as above.

During perspective projection, X, Y and Z coordinates are scaled up for points between the eye position and the focal point. As a point approaches the eye, the transformed coordinates approach infinity. The coordinates of points behind the focal point are scaled down. This is visualized in **Figure 6.13**.

When specifying windows, this coordinate scaling effect must be taken into account.

**Figure 6.13** *Effect of perspective projection.*



**Top:** Untransformed objects. The focal point is at the centre of the square marked 3, the X and Y axes are drawn in the projection plane.

**Bottom:** Transformed objects, showing the scaling of coordinates depending on the position relative to the eye position and the focal point. Note that coordinates in the projection plane are not scaled.

### 6.5.2 Vanishing Point

As described on **page 6-7**, perspective projection could be achieved by explicitly specifying the vanishing point.

When *PERS* is used, GPGS-F will set the vanishing point at the viewing line, i.e. the line from the eye through the focal point. The distance from the focal point will be the same as the distance to the eye, but in the opposite direction.

The vanishing point is the point where the extension of lines defined to be parallel to the viewing line will meet after being transformed. To be precise, the vanishing point does not only specify a point, but a vanishing *plane*. The plane is perpendicular to the viewing line, at the specified point. Any lines defined to be parallel, will, if extended, meet at some point in this plane. The only exception is when the lines are perpendicular to the viewing line.

There is no need to define special treatment of points behind the vanishing point, as this is an impossible situation. An untransformed point behind the focal point will be transformed to be between the focal point and the vanishing point. As the untransformed point approaches infinity, the transformed point will approach the vanishing point.

## 6.6 Transformation Mode

Whenever a transformation routine is called, GPGS-F will build a matrix describing the specified transformation. This matrix is then multiplied with the current system matrix to build the new system matrix.

When the new matrix is post-multiplied with the system matrix ( $S' = S \times T$ ) the transformation is said to be performed in *space mode*. This is the default mode and implies that each new transformation is relative to the current user coordinate system.

If instead the new matrix is pre-multiplied with the system matrix ( $S' = T \times S$ ) the transformation is performed in *picture mode*. In this case, new transformations are specified in terms of the initial user coordinate system, i.e. the window.

Which transformation mode to use is selected by

**CALL MODTRN (lmod)**

where **lmod**=1 specifies picture mode, and **lmod**=0 specifies space mode.

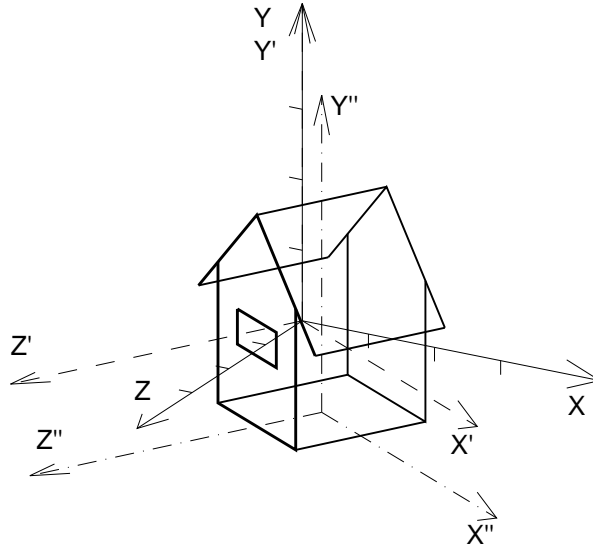
Note that when a transformation matrix is pushed on the stack by *BGNTRN* (see **page 6-9**), the transformation mode will be saved as well. When the matrix is restored by *ENDTRN*, the transformation mode will also be restored.

**Figure 6.14** *Transformations in space and picture mode.*

**Space mode (default)**

```
CALL ROTAD (-30.0, 2)
CALL XLAT3 (3.0, 0.0, 1.5)
```

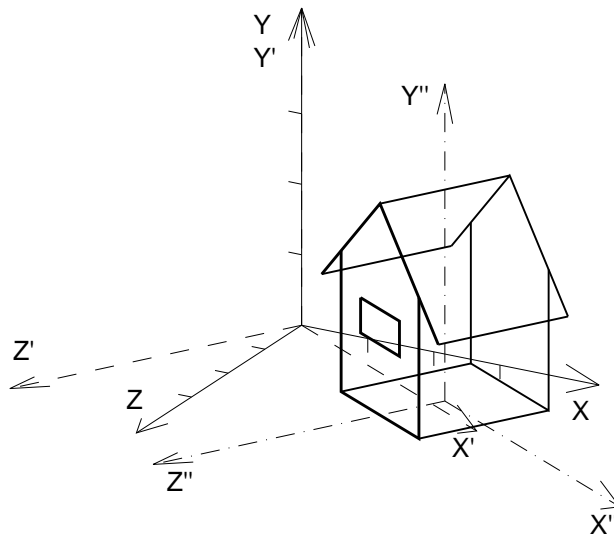
The translation is relative to the rotated (dashed) axes.



**Picture mode**

```
CALL MODTRN (1)
CALL ROTAD (-30.0, 2)
CALL XLAT3 (3.0, 0.0, 1.5)
```

The translation is relative to the initial (solid) axes.







# Chapter 7

## Character Strings

---

Although a picture says more than a thousand words, some text is still needed with most graphic applications.

To draw a text string, GPGS-F needs a reference position, and of course the text string itself. Other aspects, such as size, orientation, alignment etc., are set by separate routines. None of these *need* to be used, as default values are supplied by GPGS-F.

The reference position of text is the *current position*. In most cases this will be set by one of the *LIN.* routines (see **page 4-2**), but text may be drawn following any graphic primitive that defines the current position.

By default, the current position marks the lower left hand corner of the first character (may be changed by the *CJUST* routine, see **page 7-7**). After drawing the string, the current position is set to the lower right hand corner of the last character.

Text is always created in the XY plane at the current Z coordinate. 3D text is achieved by transforming the XY plane within the 3D space.

### 7.1 Drawing Text Strings

A text string is drawn by

**CALL CHARC (Chstri)**

where **Chstri** is a character variable, a character array element or a character constant. The string is terminated either when all characters in the string are drawn or when the sequence '\* .' is encountered (see next page).

For compatibility with previous versions of GPGS-F, text may also be drawn by

**CALL CHARS (larr(1))**

where **larr** contains a string in Hollerith compatible format (one ASCII value in each byte), or by

**CALL CHARA (larr(1), llth)**

where **larr** is an array of characters in A1 format, and **llth** is the number of characters to draw.

When *CHARS* is used, the string must be terminated by the sequence '\* .', as the length of the string is not supplied.

### 7.1.1 Format Control

To allow the user some control over the format of the characters in the supplied string, GPGS-F recognizes certain character combinations for format controls. The first of the two format control characters is the *system escape character* and the second indicates the action to be performed.

The default system escape character is '\*' (ASCII value 42), but it may be changed by

**CALL CESCAP (lchar)**

where **lchar** is the new escape character to precede any format codes. **lchar** is given in integer A1 format.

The A1 format is not standardized. On some computers the ASCII value must be put in the left byte, on others in the right byte. To keep the code as machine independent as possible, the same value could be put into all bytes of the integer **lchar**. This will work because GPGS-F extracts just the interesting byte, it does not care about the rest of the integer (according to the Fortran definition, the other bytes should be set to the ASCII value of space).

**Table 7.1** *Format control sequences.*

Sequence	Action to perform
**	Draw the * character.
*.	End of string.
*L	Switch to lower case.
*U	Switch to upper case.
*N	New line (line feed + carriage return).
*C	Carriage return.
*S	Compose 8 bit character (see <b>page 7-10</b> ).

\*L and \*U are retained from the ancient days when Fortran allowed upper case characters only.

#### **Example 7.1** *Using format control.*

The statement `CALL CHARC('3 lines of text*Nusing a*Nsingle call')`  
 will result in  
 3 lines of text  
 using a  
 single call

Note that if a string is *ended* by '\*N', the current position is set to one line below the starting character. Thus, left aligned lines of text may be drawn without explicitly positioning each individual line even when *CHARC* is invoked more than once.

## 7.2 Drawing Integer and Real Numbers

The value of integer and real variables or constants may be drawn as text without first converting to a character string by the application program.

Integer numbers are drawn by

**CALL CHARI (Intno, Length)**

where **Intno** is the integer variable or constant and **Length** is the number of character to use for the string. This corresponds to the Fortran format specification '**Iw**'.

Floating point variables are drawn by

**CALL CHARF (Flpno, Length, Lfrac)**

corresponding to the Fortran format '**Fw.d**' (**Length=w**, **Lfrac=d**), or by

**CALL CHARE (Flpno, Length, Lfrac)**

corresponding to the Fortran format '**Ew.d**'.

The generated string is filled with leading spaces to get the specified length. If the length is specified too small, the string is filled with asterisks.

Below is shown several examples of text strings generated by the number drawing routines. Note that GPGS-F allows floating point numbers to be written with no fraction part, and that the leading 0 for numbers between 0 and 1 need not be written.

### Example 7.2     *Strings generated by CHARI, CHARF and CHARE.*

Subroutine call	Resulting text string
CHARI (1234, 4)	1234
CHARI (12, 4)	12
CHARF (12.36, 6, 3)	12.360
CHARF (12.36, 5, 2)	12.36
CHARF (12.36, 4, 1)	12.4
CHARF (0.123, 5, 3)	0.123
CHARF (0.123, 4, 3)	.123
CHARF (123.0, 5, 1)	123.0
CHARF (123.0, 4, 1)	****
CHARF (123.0, 4, 0)	123.
CHARE (0.127, 9, 3)	1.270E-01
CHARE (0.127, 9, 2)	1.27E-01
CHARE (0.127, 9, 1)	1.3E-01
CHARE (16000.0, 8, 2)	1.60E+04
CHARE (16000.0, 8, 1)	1.6E+04

## 7.3 Character Size

The size of graphic text is set by

**CALL CSIZES (Xspace, Yspace)**

where **Xspace** and **Yspace** specifies the size of the character *space*, including character and line spacing. The values are given in user coordinates.

By default, GPGS-F defines a character size giving 40 lines of 80 characters within the current window, i.e. when the default window (0.0 to 1.0) is current, **Xspace**=1/80, **Yspace**=1/40. If the window limits are changed and **CSIZES** has not been called, GPGS-F will change the character size to 1/80 by 1/40 of the *new* window. If **CSIZES** has been called, changing the window will not affect the character size values.

As the character space is given in user coordinates, the method described in **Chapter 5** may be used to specify the character size in a given physical unit.

Within the character space, a letter occupies a smaller rectangle, set by

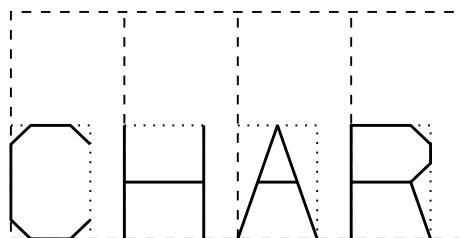
**CALL CSIZEL (Xlett, Ylett)**

where **Xlett** and **Ylett** are specified as fractions of the character space, i.e. the size of the letter is **Xspace**×**Xlett** in X direction and **Yspace**×**Ylett** in Y direction. Following from this, the spacing between letters will be **Xspace**×(1-**Xlett**) and the line spacing will be **Yspace**×(1-**Ylett**).

The default values are 0.7 for **Xlett** and 0.5 for **Ylett**, i.e. the height to width ratio of the letter size is 1.0 to 0.7

**Figure 7.1** *Character size definition.*

The dashed lines show the character *space*, the dotted lines show the *letter size*.



When using default values for **CSIZEL**, and setting the character height to twice the width through **CSIZES**, the character spacing and aspect ratio will be as defined by the software font designer. When using hardware text (see **page 7-6**), text will appear as defined by device hardware.

Using the default values will in most cases give satisfactory results, but there are no limitations to the values that may be used with **CSIZES** and **CSIZEL**, allowing any size, spacing and aspect ratio to be obtained.

## 7.4 Character Transformations

As mentioned on **page 7-1**, graphic text is always written in the user's XY plane at the current Z coordinate. To draw 3 dimensional text, the XY plane must then be transformed within the 3D space. If perspective projection is selected, this will be applied to the text as well, unless hardware text is used (see **page 7-6**).

**Figure 7.2** *Perspective view of 3 dimensional text.*



In addition to the general modelling transformation routines, GPGS-F provides two transformation routines that are used for text only.

### 7.4.1 Shearing

Shearing is applied to a character string by

**CALL CSHEA (Shear)**

where the shearing angle is the arc tangent of **Shear**. Note that a positive value means that the 'up-vector' of the string is sheared *clockwise*, while with rotation a positive angle means *counterclockwise* rotation.

**Figure 7.3** *Character shearing.*

CSHEA(-0.25)

CSHEA(0.25)

CSHEA(0.5)

CSHEA(0.75)

The default value of **Shear** is 0.0, i.e. no shearing.

## 7.4.2 Rotation

By default, graphic text is drawn parallel to the X axis. This may be rotated by

**CALL CROTA (Angle)**

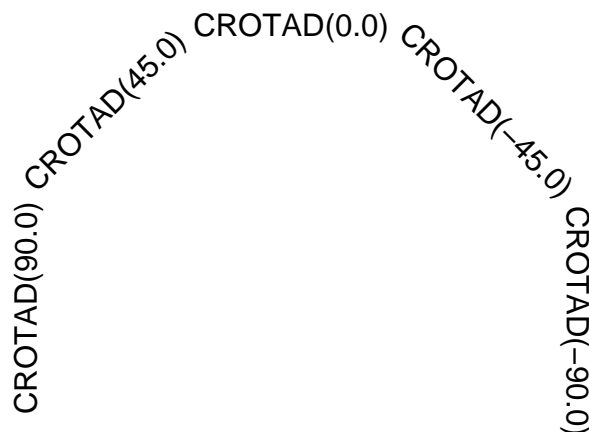
where **Angle** specifies the rotation angle in radians, or

**CALL CROTAD (Dangle)**

where **Dangle** specifies the rotation angle in degrees. Note that the rotation angle is *absolute*, not relative to the previous value. There is no need to supply the rotation axis, as this is a 2D rotation, i.e. the rotation axis is always the Z axis.

When text rotation is set, each string will be rotated around its reference position, i.e. the current position.

**Figure 7.4** *Character rotation.*



## 7.5 Software / Hardware Text Generation

It is possible to select whether to use GPGS-F software or device hardware to produce graphic text by

**CALL SOFCHA (lsw)**

where **lsw**=1 will select software (default), **lsw**=0 will select hardware.

When software text is selected, each character is made up by a number of straight lines, allowing any size and transformation to be applied to the text.

With most devices, hardware text will be faster to draw, but there will often be a limited number of sizes and rotation angles available.

The driver descriptions in **Appendix E** lists the capabilities of the individual drivers.

## 7.6 Text Alignment

By default, a text string is positioned with its lower left hand corner at the current position. GPGS-F is however able to produce centred and right aligned text by specifying a horizontal and vertical alignment factor.

These alignment factors are set by

CALL CJUST (Horiz, Vert)

**Horiz** specifies the position of the current position along the baseline of the string, where 0.0 is the start and 1.0 is the end of the string. The value is however not limited to be within the range 0.0 to 1.0, any value may be used.

**Vert** specifies the position of the current position along a line from the baseline (0.0) to the character *space* height (1.0). Thus, the exact position of the string will also depend on the character *box* height set by *CSIZEL* (see [page 7-4](#)). As with **Horiz**, any value may be given for **Vert**.

The default values for both **Horiz** and **Vert** are 0.0

**Figure 7.5** *Text alignment.*

Arguments to <i>CJUST</i>		
Horiz	Vert	
0.0	0.0	✕String
0.5	0.0	String✕
1.0	0.0	String✕
0.0	0.25	✕String
0.5	0.25	String
1.0	0.25	String✕
0.0	0.5	✕String
0.5	0.5	String✕
1.0	0.5	String✕

The vertical alignment shown is based on the default value of the character box height.

If either **Horiz** or **Vert** is set to a value unequal to 0.0, the current position is **not** updated to the end of the string after it is drawn.

When aligning a string containing several lines of text, the horizontal alignment is applied to each substring, while the vertical alignment factor is used to position the first substring.

**Example 7.3**     *Two methods of aligning multiple text strings.*

```
C
C First draw 3 right aligned strings using a single call.
  CALL CJUST (1.0,0.0)
  CALL LINE (5.0,8.0,0)
  CALL CHARC ('3 lines of text*Nusing a*Nsingle call')
C
C Use 3 individual calls, adding 1.0 to the vertical alignment
C for each new line of text, thus moving it down 1 line.
  CALL CJUST (1.0,0.0)
  CALL LINE (5.0,3.0,0)
  CALL CHARC ('3 lines of text')
  CALL CJUST (1.0,1.0)
  CALL CHARC ('using three')
  CALL CJUST (1.0,2.0)
  CALL CHARC ('individual calls')
```

**Figure 7.6**     *Aligning strings (using the code from Example 7.3).*

3 lines of text<sub>x</sub>  
using a  
single call

3 lines of text<sub>x</sub>  
using three  
individual calls

## 7.7 Text Fonts

With software text, one of eight different text fonts may be used. Five of these contain roman characters in different styles, two contain greek characters and one contains mathematical symbols. In addition, cyrillic characters are included with one of the greek fonts.

The font to use is selected by

**CALL CFONT (lfont)**

where **lfont** is the font number as given by **Table 7.2**. If a font number outside the range 0 to 7 is given, font 0 will be used.

When hardware text is selected, the font number is just passed to the device driver in use, and the result will depend on the driver capabilities. Some drivers just supply a single font, others may have several (with the PostScript driver, a total of 19 fonts are available).



**Table 7.2**    *Software text fonts.*

Font number	Font name
0	GPGS-F default font
1	Simplex Roman
2	Complex Roman
3	Complex Italic
4	Duplex Roman
5	Simplex Greek
6	Complex Greek and Cyrillic
7	Mathematical Symbols

**Figure 7.7**    *GPGS-F font examples.*

Font 0 :	012	ABC	abc	012	ABC	abc
Font 1 :	012	ABC	abc	012	ABC	abc
Font 2 :	012	ABC	abc	012	ABC	abc
Font 3 :	012	ABC	abc	012	ABC	abc
Font 4 :	012	ABC	abc	<b>012</b>	<b>ABC</b>	<b>abc</b>
Font 5 :	012	ABΓ	αβγ	<b>012</b>	<b>ABC</b>	<b>abc</b>
Font 6 :	012	ABΓ	αβγ	<b>012</b>	<b>ABC</b>	<b>abc</b>
Font 7 :	ϕ†‡	∞ §	∞ §	012	ABC	abc

The characters of software text font number 0 is coded within the GPGS-F Fortran code. When one of the fonts 1 to 7 is selected, the character definitions are read from a data file that are delivered with the GPGS-F system.

When the fontfile is opened by GPGS-F, the Fortran unit number set by the installation dependent parameter **MFUNIT** (see **Appendix A**) is used by default.

If this number is used by the application for other purposes, it is possible to tell GPGS-F to use a different unit number by

CALL    SETFNU (lfontu, lerru)

where **lfontu** is the new Fortran unit number to use. **lerru** is described on **page 24-4**.

If -1 is given for **lfontu** and/or **lerru**, the default value is reset, while 0 means no change.

## 7.8 Character Encoding

GPGS-F interprets character codes as 8-bits international ASCII values according to the definitions in the ISO 8859 standard, when using software fonts 0 to 4. The encoding of fonts 5 to 7 are GPGS-F specific. **Appendix B** shows all characters available with each software font. Note that some rarely used characters are not defined in all fonts.

When hardware text is selected, all character codes are just transferred to the device driver in use, and interpreted by this. Some drivers are able to print any 8-bits characters, others will just mask out the most significant bit.

Whether 8-bits characters may be included in strings passed to *CHARC* depends on both the editor and the Fortran compiler in use. If the editor and/or the compiler does not allow 8-bits characters, such characters may be composed by either using the Fortran Intrinsic function *CHAR*, or the format control sequence '*\*Sx*', where '*x*' is the character whose ASCII value is 128 less than the character to print (other format control sequences are described on **page 7-2**).

Thus, the word **større** may be printed by using one of the following calls:

- a) `CALL CHARC ('større')`  
if the editor and compiler accepts 8-bit characters  
(and the keyboard is able to produce the ø letter)
- b) `CALL CHARC ('st'//CHAR(248)//'rre')`  
the ASCII value of ø is 248
- c) `CALL CHARC ('st*Sxrre')`  
248-128=120, i.e. x

### 7.8.1 National Character Sets

As an alternative to using 8-bits character codes to produce international characters, it is possible to select that a language specific encoding is to be applied to 7-bits characters codes. This encoding is selected by

**CALL CLANG (llang)**

where the number **llang** specifies the encoding to use. The available language specific encodings are given by **Table 7.3**. The default value of **llang** is 0, which is mapped to one of the other values according to a rule specified when GPGS-F is installed.

Which 7-bit character codes are affected by selecting language specific encoding is shown in **Table 7.4**.

**Table 7.3** *National character sets.*

Language number	Language name
0	Installation dependent
1	ISO 8859
2	Norwegian
3	Swedish
4	German
5	French
6	British
7	Italian
8	Spanish
9	Portuguese
10	Norwegian (version 2)
11	Swedish for names

**Table 7.4** *Language dependent encoding of 7-bit ASCII values.*

	ASCII Values											
	35	36	64	91	92	93	94	96	123	124	125	126
ISO 8859 :	#	\$	@	[	\	]	^	`	{		}	~
Norwegian :	#	\$	@	Æ	Ø	Å	^	`	æ	ø	å	-
Swedish :	#	¤	@	Ä	Ö	Å	^	`	ä	ö	å	-
German :	#	\$	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
French :	ℒ	\$	à	°	ç	§	^	`	é	ù	è	¨
British :	ℒ	\$	@	[	\	]	^	`	{		}	-
Italian :	ℒ	\$	§	°	ç	é	^	ù	à	ò	è	ì
Spanish :	ℒ	\$	§	¡	Ñ	¿	^	`	°	ñ	ç	~
Portuguese :	#	\$	§	Ã	Ç	Õ	^	`	ã	ç	õ	°
Norwegian V.2 :	§	\$	@	Æ	Ø	Å	^	`	æ	ø	å	~
Swedish (names) :	#	¤	É	Ä	Ö	Å	Ü	é	ä	ö	å	ü

## 7.9 Proportional Spacing

When using software fonts 1 to 7, the text is drawn using proportional spacing. That is, the space occupied by each single character depends on the width of the character as defined by the font designer.

If text is to be aligned on a character basis, proportional spacing may be switched off by

**CALL CFPROP (lsw)**

with **lsw** set to 0. Proportional spacing is reselected by setting **ISW** to 1.

Note that it is not necessary to switch proportional spacing off to get numbers aligned, as all numbers are defined with equal width within each font.

**Figure 7.8** *Proportional spacing.*

Proportional spacing ON  
Proportional spacing OFF

## 7.10 Inquiring Text Extent

In some cases, such as when accurate positioning of text is necessary, it may be useful to know the length of a text string.

For this purpose, routines finding the length of a character string in user coordinates are provided.

For a string drawn by *CHARC* (see page 7-1)

**CALL DATCXC (Chstri, Strlen, Nlines)**

will return the length of the longest line through **Strlen**, and the number of lines in the string through **Nlines**. As with *CHARC*, **Chstri** may be a character variable, a character array element or a character constant.

For compatibility with previous versions of GPGS-F, two additional routines are available.

**CALL DATCXS (larr(1), Strlen, Nlines)**

will return the length of the string **larr** given in Hollerith compatible format as with *CHARS*, and

**CALL DATCXA (larr(1), llth, Strlen, Nlines)**

will return the length of the string **larr** given in A1 format as with *CHARA*.

In addition to finding the length of a string, the coordinates of a box enclosing the string may be inquired by

**CALL DATCBX (Tx, Ty, Strlen, Nlines, Cx, Cy,  
Bxarr(1), Byarr(1))**

(**Tx, Ty**) is the reference position used when drawing the string (starting point if no text alignment is specified), **Strlen** and **Nlines** are the values returned from the *DATCXC* routine.

The returned position (**Cx, Cy**) marks the end of the string. This may be used as the starting point if a second string if strings are to be concatenated.

The arrays **Bxarr** and **Byarr** return the coordinates of the rectangle enclosing the text string. The coordinates are given in sequence lower left, lower right, upper right, upper left, i.e. the length of the arrays must be 4.

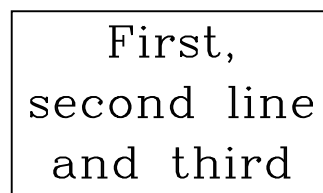
Finding the surrounding box of a text string may, for example, be used to draw a frame around the text, as shown in **Example 7.4**.

Note that *DATCXC* and *DATCBX* are not guaranteed to return correct values if hardware text is selected. If the device driver is not able to find the requested information, the returned values will be the same as if software font 0 was used.

#### **Example 7.4**     *Framing a text string.*

```
C
C   Assuming that the window and viewport has been defined so that
C   user units are centimetres.
C
C       REAL BXARR(4), BYARR(4)
C       CHARACTER CHSTRI*30
C       ...
C       CHSTRI='First,*Nsecond line*Nand third'
C       ...
C
C   Set the character size to 4 by 8 mm.
C   Specify that the text is to centred.
C
C       CALL CSIZES (0.4, 0.8)
C       CALL CJUST (0.5, 0.0)
C       CALL LINE (5.0, 8.0, 0)
C       CALL CHARC (CHSTRI)
C       CALL DATCXC (CHSTRI, STRLEN, NLINES)
C       CALL DATCBX (5.0, 8.0, STRLEN, NLINES, CX, CY, BXARR, BYARR)
C
C   Draw a frame with offset 2 mm from text.
C
C       DXY = 0.2
C       CALL LINE (BXARR(1)-DXY, BYARR(1)-DXY, 0)
C       CALL LINE (BXARR(2)+DXY, BYARR(1)-DXY, 1)
C       CALL LINE (BXARR(2)+DXY, BYARR(3)+DXY, 1)
C       CALL LINE (BXARR(1)-DXY, BYARR(3)+DXY, 1)
C       CALL LINE (BXARR(1)-DXY, BYARR(1)-DXY, 1)
```

**Figure 7.9**     *Framing a text string (using the code from Example 7.4).*



First,  
second line  
and third

The reason why the space below the text in the figure is larger than 2 mm, is that character descenders are taken into account when the surrounding box is computed.

## Chapter 8

# Interaction Facilities

---

To be truly general purpose, a graphic subroutine package like GPGS-F must not only provide routines for generating graphic output. To allow interactive applications, there must also be methods for reading data from the various input (interaction) tools of a graphic device.

With GPGS-F, the different interaction tools are divided into classes according to the information they return. The different classes are:

- Text:** returns a character string.
- Pick:** returns the segment number and an element namestack of a graphic element pointed at.
- Valuator:** returns a real number between a defined lower and upper limit.
- Locator:** returns a position in Normalized Device Coordinates (NDC).
- Button:** returns a button number and button status.

Addressing input tools in a device independent way is not as straightforward as addressing the display surface. With GPGS-F, the method selected is to identify the tools by assigning an integer identification to each distinct tool of the graphic device. This number is called a tool number.

**Table 8.1**     *Interaction tool numbers.*

Tool number(s)	Type of tool(s)
2	Text
3	Pick (hit)
101 - 199	Valuators
201 - 299	Locators
400 - 499	Buttons or function keys
900 - 999	Device dependent ESCAPE functions

If there is more than one tool available of a given class, these will be numbered x01, x02 etc. (Example: if a graphic terminal has both a graphic cursor and a tablet that may be used for locator input, the cursor is assigned number 201, the tablet 202).

Note that the same physical tool may be used as different logical tools, e.g. a graphic cursor controlled by a mouse may be used both as pick and locator tools.

The driver descriptions in **Appendix E** show what interactive tools are available with each driver.

## 8.1 Basic Interactive Programming

This section describes the routines giving access to the basic interactive facilities of GPGS-F. Other routines and more detailed information is given in the following sections.

For each input class there is a routine that halts program execution, waits for the user to generate an interrupt by activating a trigger, and returns the input data. This sort of interaction is called **Request mode input**.

The most commonly used interaction method is to read the position of some locator tool (often a graphic cursor). With GPGS-F this is done by

**CALL REQLOC (*ltool*, *Xndc*, *Yndc*)**

**ltool** is a locator tool, between 201 and 299. **Xndc** and **Yndc** is the returned position of the cursor in *Normalized Device Coordinates* (see [page 2-2](#)). Utility routines are available to convert these coordinates to window or user coordinates. These routines are described on [page 8-14](#).

The trigger used with locator tools is often a mouse button or a keyboard key. With some devices, the trigger used may be identified by using the *REATOL* routine described on [page 8-12](#).

### Example 8.1     *Basic interaction using the locator.*

```
C *****
C COMPLETE WORKING AXAMPLE
C *****
C
C Read device number, initialize device.
C
C     READ *, IDEV
C     CALL GPGS
C     CALL NITDEV(IDEV)
C     CALL BGNPIC(1)
C
C Read a position to use as starting point.
C
C     CALL REQLOC(201, XNDC, YNDC)
C     CALL LINE(XNDC, YNDC, 0)
C
C Then draw a line to each new point, until giving a position
C close to the left edge of the display.
C
1000 CONTINUE
C     CALL REQLOC(201, XNDC, YNDC)
C     IF (XNDC .GT. 0.05) THEN
C         CALL LINE(XNDC, YNDC, 1)
C         GO TO 1000
C     ENDIF
C     CALL ENDPIC
C     CALL RLSDEV(IDEV)
C     END
```



A text string is fetched using

**CALL REQTXC (Itool, Chstri, Length)**

**Itool** is the text input tool (always 2), **Chstri** is a Fortran-77 character string where data is to be returned, **Length** is the number of characters returned, not including trailing spaces. **REQTXC** will wait for the operator to enter a text string terminated by Carriage Return.

If retained segments (see **Chapter 14** and **Chapter 16**) are used, graphic elements may be made *detectable* (pickable) by using the routines described in **Chapter 20**. Once an element is detectable, it may be pointed at and be identified by

**CALL REQHIT (Itool, Maxnam, Namarr(1), Lennam)**

**Itool** is the pick input tool (always 3), **Maxnam** specifies the maximum number of identifiers to be returned through **Namarr**. On return, **Lennam** gives the actual number of identifiers returned, less or equal to **Maxnam**. The identifiers returned through **Namarr** is the picture segment number, followed by the *namestack* of the graphic element. If no detectable element was pointed to, **Lennam** is returned zero.

Pick input, and the picture element namestack, is described in detail in **Chapter 20**.

Note that the position of the pick tool is not returned. For some devices this is however available, and may be fetched using the **REATOL** routine (**page 8-12**).

A valuator tool is activated by

**CALL REQVAL (Itool, Value)**

**Itool** is the valuator tool number, between 101 and 199. **Value** is the value returned, normally between 0.0 and 1.0 (this tool is not often available).

A change in button status may be requested by

**CALL REQBUT (Itool, Ibutno, Istat)**

**Itool** is the button number, between 400 and 499. 400 means wait for any button to be triggered. **Ibutno** returns the tool number of the button actually activated. **Istat** is the button action (new status), 0=Button released (up), 1=Button pressed (down), -1=Button activated, GPFS-F cannot decide whether it was pressed or released.

A button may be a mouse button or some sort of function key. In both cases, some devices may generate interrupts both when a button is pressed and released, other devices generate interrupts only when a button is pressed.

The driver descriptions in **Appendix E** show what button tools are available with the different devices, and how interrupts are generated.

## 8.2 Interaction Modes

Each of the input tools can operate in three modes. These modes are **Request**, **Sample** and **Event**. Input from a tool is obtained in different ways depending on the mode:

**Request:** GPGS-F will wait until the input is entered by the operator, similar to Fortran or C read operation. This is the default mode for all tools. Request mode input was described in **section 8.1**.

**Sample:** Sampling a tool causes GPGS-F to return the current logical input value from the specified tool, without waiting for any operator action. If sampling the state of the tool is not possible, GPGS-F will return the value set by tool initialization or generated by the last request or event mode action. Sample mode input is described in **section 8.2.1**

**Event:** The device driver maintains an input queue containing ordered event reports. An event report contains the identification of the tool and the associated input value(s). More than one tool may be set in event mode, and events are entered asynchronously with the user program. The oldest event in the queue may be made the current event and fetched by the user program. A tool is set in event mode by explicitly enabling/disabling it for event input. Event mode input is described in **section 8.2.2**

### 8.2.1 Sample Mode Input

Note that only a very few devices support sample mode input. Most devices just return the value set by the last request mode input or tool initialization.

For each input class there is a routine available to sample the current state of a tool. These routines are very similar to the corresponding routines for request input. For each request input routine '*REQ...*' there is a sample input routine '*SMP...*' with identical argument list, and returning the same kind of data.

The available routines for sample mode input are just listed without describing the arguments, as these will be identical to the arguments of the corresponding '*REQ...*' routine described in **section 8.1**

Tool number 2, Text:

**CALL SMPTXC (Itool, Chstri, Length)**

Tool number 3, Pick

**CALL SMPHIT (Itool, Maxnam, Namarr(1), Lennam)**

Tool numbers 101 to 109, Valuator

**CALL SMPVAL (Itool, Value)**

Tool numbers 201 to 299, Locator

**CALL SMPLOC (ltool, Xndc, Yndc)**

Tool numbers 400 to 499, Button

**CALL SMPBUT (ltool, lbutno, lstat)**

## 8.2.2 Event Mode Input

The event mode input routines are included in GPGS-F for future extensions to existing device drivers and for coming new drivers. There is currently no device drivers supporting these routines.

GPGS-F keeps a list of tools that may be used for event mode input. A tool is added to this list (enabled for event mode input) by

**CALL ENABLE (ltool)**

where **ltool** is the input tool number. A tool that is enabled for event mode input, cannot be used for request or sample mode input. If tool number 400 is specified, this means that all buttons may be used in event mode.

If a tool no longer is to be used for event mode input, it must be removed from the list (disabled) by

**CALL DSABLE (ltool)**

The tool may then be used for request or sample mode input again. If tool 400 is enabled, it is not possible to disable a subset of the buttons and leave the rest enabled. This effect may however be achieved by first disabling tool 400, and then enabling one or more individual buttons.

As a tool is disabled, event reports from that tool will not be removed from the event queue. If this is wanted, it must be explicitly executed by

**CALL FLUSHE (ltool)**

saying that all events generated by **ltool** is to be removed from the event queue. Note that flushing events is also possible for a tool that is currently enabled.

For button events, it is possible to flush events generated by a single button using tool number 40x, or all buttons using tool number 400.

At most, 20 tools may be enabled for event mode input at a time. The operator may activate any of these tools and enter data into the event queue asynchronously of the user program. Events from the queue are fetched by first finding what tool generated the oldest event, and depending on this, getting the actual event report by calling a tool specific routine.

**CALL AWAIT (Time, Ittool)**

will return the tool number that generated the oldest event through **Ittool**. If the event queue is empty, the program will wait maximum **Time** seconds for user action (i.e. for a new event to be entered). If **Time** is specified negative, timeout will not occur, i.e. the program will wait until a new event report is generated (similar to request mode input).

**Ittool** returns the tool number of the oldest event in the queue. If **Ittool** is zero on return, the event queue was empty, and no new event was generated within the period specified by **Time**.

Note that if tool 400 is enabled for event input, **Ittool** will not return 400 for button events, but the tool number of the actual button generating the event.

The actual event report is fetched by calling one of the following routines, depending on the tool number returned by *AWAIT*.

A text report is fetched by

**CALL GETTXC (Chstri, Length)**

**Chstri** is a Fortran-77 character string returning the text string. **Length** is the number of characters returned, not including trailing spaces. A text report is entered when the operator types Carriage Return, or when the text input buffer is full (may be device dependent).

A pick report is fetched by

**CALL GETHIT (Maxnam, Namarr(1), Lennam)**

As with *REQHIT* (see page 8-3), this function requires use of retained segments. **Maxnam** specifies the maximum number of identifiers to be returned through **Namarr**, **Lennam** gives the number actually returned. **Namarr** returns the segment number and the element namestack as *REQHIT*. If no detectable element was pointed to, **Lennam** is returned zero.

Once a pick report has been fetched, it is possible to get the position of the pick tool as that report was generated by

**CALL HITPOS (Xndc, Yndc)**

The position is returned in Normalized Device Coordinates.

A valuator report is fetched by

**CALL GETVAL (Value)**

returning **Value**, a real value between a tool dependent lower and upper limit (normally between 0.0 and 1.0).

A locator report is fetched by

**CALL GETLOC (Xndc, Yndc)**

where **Xndc** and **Yndc** is the position in normalized device coordinates of the point given. The key or button pressed is not available with event mode input.

A button report is fetched by

**CALL GETBUT (Istat)**

where **Istat** is the button action (new button status), 0=Button released (up), 1=Button pressed (down), -1=Button activated, GPGS-F cannot decide whether it was pressed or released.

### **Example 8.2    *Event mode input.***

```

C
C  Enable tools (text, locator, button 1) for event input.
C
      CALL ENABLE(2)
      CALL ENABLE(201)
      CALL ENABLE(401)
C
C  Do computations, draw something
C      ...
C
C  Wait maximum 1 minute for tool to be activated.
C
      CALL AWAIT(60.0, ITOOL)
      IF (ITool .EQ. 0) THEN
          PRINT *, ' Please wake up, buddy'
      ELSEIF (ITool .EQ. 2) THEN
          CALL GETTXC(Chstri, LENGTH)
          CALL ACTION(Chstri(1:LENGTH))
      ELSEIF (ITool .EQ. 201) THEN
          CALL GETLOC(XNDC, YNDC)
          CALL DOIT (XNDC, YNDC)
C  Help function if button pushed, (ignore button status)
      ELSEIF (ITool .EQ. 401) THEN
          CALL HELP(Program-context)
      ELSE
          PRINT *, ' Illegal tool (system error ?????)'
      ENDIF
      ...

```

## 8.3 Echo Control

Echoing is used to give the operator indication of the current state of the input tool, and/or some feedback to indicate when the tool is triggered.

In request mode, the echo is shown when calling the interaction routine and removed when the input value has been entered. In event mode, echoing is shown when the tool is enabled for events and continuously visible until the tool is disabled.

The echo may be turned on/off by

**CALL ECHCTL (ltool, lstat)**

where **ltool** is the input tool number and **lstat** is 0 to turn echoing off and 1 to turn it on. By default echoing is on. With most devices it is not possible to turn echoing off.

The echo area is defined to be the part of the display surface that may be used for interaction by locator and pick tools. By default this is set to cover the complete display surface, but may be changed by

**CALL ECHVP (ltool, Varr(1) )**

where **Varr** is a viewport array (NDC) in sequence [**Xlow**, **Xhigh**, **Ylow**, **Yhigh**]. **ltool** is the logical tool number. This facility is also available with very few devices.

### 8.3.1 Echo Specification

The appearance of the input tool echo may be specified by

**CALL ECHTOL (ltool, larr(1), Lthi, Farr(1), Lthf)**

where **ltool** is the logical tool number, **larr** is an integer array of length **Lthi** and **Farr** is a real array of length **Lthf**. The contents of **larr** and **Farr** are interpreted by the device drivers, depending on the tool selected.

Obviously, **larr** and **Farr** will contain quite different kind of data for the different tools. **larr(1)** is however always used to select the echo *type*, and setting this to 4 will select the default echo type defined by the device driver. This default echo type will also be used if an echo type that is not available with a given driver is specified.

Some echo types are selected by using the value of **larr(1)** only, others require additional data to be supplied through **larr** or **Farr**. If some of these required data are not supplied by the application program, the device drivers will either use the values from a previous call, or if there was no previous call, some device dependent default values.

With the following description of the various echo types for each input tool, the echo types defined by GPGS-F are listed. Most drivers will provide only a subset of these echo types, while others may provide device specific echo types in addition to those listed.

### Echo specification for tool number 2, text

<b>larr(1)</b> = 4	→ Default. Echo the input text in a device dependent position. <b>larr(2)</b> : Input buffer length.
<b>Farr(1)</b>	: New text buffer start position in X direction (NDC).
<b>Farr(2)</b>	: New text buffer start position in Y direction (NDC).

### Echo specification for tool number 3, pick

<b>Farr( )</b>	: For all echo types unless otherwise specified: <b>Farr(1)</b> : New pick cursor X position (NDC). <b>Farr(2)</b> : New pick cursor Y position (NDC). If possible, the driver will move the pick cursor to this new position the next time the pick tool is activated.
<b>larr(1)</b> = 4	→ Use default echo type defined by the device driver.
<b>larr(1)</b> = 5	→ Set pick aperture and cursor shape. Note that this does not change the echo type. <b>Farr(1)</b> : New pick aperture size in X direction (NDC). <b>Farr(2)</b> : New pick aperture size in Y direction (NDC). If <b>Lthf</b> = 1, the pick aperture becomes a square. <b>larr(2)</b> = 1 → Use hit rectangle with the size of the pick aperture. <b>larr(2)</b> = 2 → Use a tracking cross or cursor.
<b>larr(1)</b> = 6	→ Highlight the graphic element picked.
<b>larr(1)</b> = 7	→ Highlight the picture segment picked.

### Echo specification for tool numbers 101 to 199, valuator

<b>Farr( )</b>	: For all echo types unless otherwise specified: <b>Farr(1)</b> : New initial value. <b>Farr(2)</b> : New lower limit (default value 0.0). <b>Farr(3)</b> : New upper limit (default value 1.0).
<b>larr(1)</b> = 4	→ Use default echo type defined by the device driver.
<b>larr(1)</b> = 5	→ Use a dial or pointer as echo.
<b>larr(1)</b> = 6	→ Use a digital representation as echo.
<b>larr(1)</b> = 7	→ Translate a retained picture segment in X direction. <b>larr(2)</b> : Picture segment number.
<b>larr(1)</b> = 8	→ Translate a retained picture segment in Y direction. <b>larr(2)</b> : Picture segment number.

### Echo specification for tool numbers 201 to 299, locator

<b>Farr( )</b> : For all echo types unless otherwise specified: <b>Farr(1)</b> : New locator cursor X position (NDC). <b>Farr(2)</b> : New locator cursor Y position (NDC). If possible, the driver will move the pick cursor to this new position the next time the pick tool is activated.	
<b>larr(1)</b> = 4	→ Use default echo type defined by the device driver.
<b>larr(1)</b> = 5	→ Use a crosshair as echo.
<b>larr(1)</b> = 6	→ Use a graphic cursor as echo. <b>Farr(1)</b> : Cursor size in X direction (NDC). <b>Farr(2)</b> : Cursor size in Y direction (NDC). If <b>Lthf</b> is 1, <b>Farr(1)</b> sets the size in both directions.
<b>larr(1)</b> = 7	→ Use a rubberband line as echo. One point is at the current locator position, the other following the locator cursor.
<b>larr(1)</b> = 8	→ Use a rubberband rectangle as echo. One corner is at the current locator position, the other following the locator cursor.
<b>larr(1)</b> = 10	→ Use a retained picture segment as echo (segment dragging). The segment will be moved relative to the current cursor position, and will remain where it is moved to after dragging. It may be reset to its original position by the <i>VIDEN</i> routine (see <b>page 18-1</b> ). <b>larr(2)</b> : Picture segment number.

### Echo specification for tool numbers 400 to 499, button

<b>larr(1)</b> = 4	→ Use default echo type defined by the device driver.
<b>larr(1)</b> = 5	→ Blink a retained picture segment when button is activated. <b>larr(2)</b> : Picture segment number.

### Echo specification for tool numbers 900 to 999, ESCAPE functions

Tool numbers in the range 900 to 999 are used to access device dependent features of a device driver. What features are available with the various drivers, and what kind of data must be supplied, are described in <b>Appendix E</b> . Two tool numbers are used by several drivers, and these are therefore standardized.	
<b>Tool 912</b>	<b>Farr(1)</b> : Pen speed in cm/s. <b>Farr(2)</b> : Acceleration in g's.
<b>Tool 920</b>	Send characters directly to the device. <b>larr( )</b> : One ASCII value in lower byte of each element.

As shown by the descriptions above, the specifications set by *ECHTOL* in some cases does not only describe the echo type and method, but also set some initial values for the given input tool. Thus, *ECHTOL* is sometimes referred to as a tool initialization routine.



## 8.4 Methods for Text Output

Interaction with the text tool (keyboard) is assigned tool number 2. Beside fetching a text string, this tool may also be used to *output* text, e.g. for prompting messages to the user (this applies to terminal devices only).

A text string is output by

**CALL ECHTXC (ltool, Chstri)**

where **ltool** is the text tool number (=2), and **Chstri** is a Fortran-77 character string containing the text to be printed. The text string is printed in the current text position. This may be specified by *ECHTOL* as described on **page 8-9**.

If setting the text position is possible for a given device, text written by Fortran (or other language) will also be placed at the position set by GPGS-F. Thus, the *ECHTXC* routine is just an alternative to using language supplied output routines independent of GPGS-F.

With most (if not very old) graphic terminals, the graphic and alphanumeric screens are logically two different screens, even though they may use the same physical display. For this kind of devices, the text position can not be set by *ECHTOL*, as GPGS-F only addresses the graphic screen.

If the application program performs direct I/O outside GPGS-F, this must be synchronized with GPGS-F as described on **page 1-5**.

Following from the above, a GPGS-F application program may generate text output in 3 different ways, GPGS-F graphic text routines (described in **Chapter 7**), the *ECHTXC* routine, and I/O operations independent of GPGS-F.

Graphic text should always be used if the text is to be a part of the GPGS-F drawing. It may however also be used for other purposes, such as writing error or help messages on the screen, as the text then will appear at the same position and with the same attributes, independent of what device is used.

## 8.5 Interaction With a Second Device

By default, the current device is used for both input and output by GPGS-F. It may however in some cases be convenient to get input from a different device than the one that is used for output. A typical example of this is if a digitizer is implemented as a separate device.

To accomplish this, it is possible to specify an alternate device to use for input only by

**CALL INPDEV (ldev)**

where **ldev** is the number of the device to which all subsequent input and echo control commands are to be directed. To reselect the default condition, i.e. reading input from the current output device, a value of -1 may be supplied through **ldev**.

### Example 8.3     *Using a separate input device.*

```

CALL GPGS
C
C Initialize a digitizer device and a graphic terminal.
C
CALL NITDEV(Digitizer)
CALL NITDEV(Screen)
C
C By default, input is fetched from the 'Screen' device.
C
CALL REQLOC(201, X1, Y1)
C
C Select the 'Digitizer' as input device, and read some points.
C For each point, draw a marker on the 'Screen' device.
C Call UPDAT to empty the command buffer as each marker
C is generated.
C
CALL INPDEV(Digitizer)
DO 1000 I=1,inumb
CALL REQLOC(201, X2, Y2)
CALL LINE(X2,Y2,0)
CALL MARKER(3)
CALL UPDAT(0)
1000 CONTINUE
C
C Switch back to input from current device ('Screen').
C
CALL INPDEV(-1)

```

## 8.6     Reading Additional Input Data

As mentioned in **section 8.1**, some input tools may generate more data than is returned through the request and sample mode input routines.

Such additional data may be fetched by

**CALL REATOL (Ittool, larr(1), Lthi, Farr(1), Lthf)**

where **Ittool** is the tool number, **larr** is an integer array and **Farr** is a real array to receive data. **Lthi** and **Lthf** specifies the number of integer and real values to return.

The most commonly available data are given below. These data are however not available with all drivers, while other drivers may provide data not listed here. Thus, the driver descriptions in **Appendix E** should be consulted for information on a given device.

#### Additional data available for pick tool

<b>Farr(1)</b> : X position (NDC) of the pick cursor.
---

<b>Farr(2)</b> : Y position (NDC) of the pick cursor.
---

#### Additional data available for locator tools

<b>larr(1)</b> : Identification of the trigger used to activate the locator (request mode only). The format of the identifier is device dependent (often A1).
---

## 8.7 Compatibility With Previous Versions

Two interaction routines are part of GPGS-F to ensure compatibility with previous versions of the system.

Request mode input may be performed by

**Index = INWAIT (Time, led(1), larr(1), Lthi, Farr(1), Lthf)**

**led** is an array of input tools, the end of the array is marked with a '-1' element. **larr** is an array of integer values returned, **Farr** is an array of real values returned. **Lthi** and **Lthf** specifies the number of integer and real values to return.

**Time** specifies the time in seconds to wait for an interrupt. If this is positive, GPGS-F will return to the user program when the time has elapsed, whether an interrupt has occurred or not. If **Time** is negative, GPGS-F will wait until an interrupt occurs. Note that most device drivers do not support positive **Time** values.

**Index** is the position in the **led** array of the input tool causing interrupt. **Index=0** means that no interrupt occurred during the specified time.

The data returned through **larr** and **Farr** for the different input tools are listed below.

**Tool number 2, text: larr** returns one character per word in Fortran A1 format.

**Tool number 3, pick: larr** returns the picture segment number and element namestack (see *REQHIT*, page 8-3). **Farr(1)** and **Farr(2)** returns the position of the pick cursor.

**Tool numbers 101 to 199, valuator: Farr(1)** returns the value.

**Tool numbers 201 to 299, locator: larr(1)** returns the ASCII value of the key or button pressed, in A1 format. **Farr(1)** and **Farr(2)** returns the cursor position (NDC).

**Tool numbers 400 to 499, button: larr(1)** returns the new button status. If tool 400 is specified, **larr(2)** returns the button actually activated.

Echo specification and/or tool initialization may be done by

**CALL WRITOL (larr(1), Lthi, Farr(1), Lthf)**

All arguments are identical to those of the *ECHTOL* routine described on page 8-8.

## 8.8 Coordinate Conversion Routines

Some utility routines are available to help the user convert coordinate values from one coordinate system to another.

This is mostly used with the interaction routines, as all coordinates returned are NDC, while the user in most cases is interested in knowing the user or window coordinate values.

Coordinates are converted from NDC to window coordinates by

**CALL NDCWIN (Xndc, Yndc, Zndc, Xwin, Ywin, Zwin)**

**Xndc, Yndc, Zndc** is a position in NDC coordinates. This position is transformed by the inverse window/viewport transformation and the position **Xwin, Ywin, Zwin** in window coordinates is returned.

The inverse transformation, from window coordinates to NDC is performed by

**CALL WINNDC (Xwin, Ywin, Zwin, Zndc, Yndc, Xndc)**

**Xwin, Ywin, Zwin** is the given position in window coordinates, **Xndc, Yndc, Zndc** is the returned position in NDC.

Converting from user coordinates to window coordinates is done by

**CALL USRWIN (Xusr, Yusr, Zusr, Xwin, Ywin, Zwin)**

**Xusr, Yusr, Zusr** is the position in user coordinates, **Xwin, Ywin, Zwin** is the returned transformed position.

The last transformation, from window to user coordinates, is done by

**CALL WINUSR (Invert, Xwin, Ywin, Zwin,  
Xusr, Yusr, Zusr)**

**Xwin, Ywin, Zwin** is the position in window coordinates, **Xusr, Yusr, Zusr** is the returned user coordinate position.

To perform this transformation, the system transformation matrix must be inverted. If no transformation routines are called between successive call to *WINUSR*, there is no need to perform this inversion each time. The **Invert** argument marks whether the matrix must be inverted. **Invert=1** means invert, 0 means do not invert (the inverted matrix computed by the previous call is used).

As the interaction routines return a position in 2D only, the user must find the Z value to supply to the conversion routines by some other means if 3D transformations are used for output. The *NDCWIN* conversion will return correct values for X and Y no matter what Z value is given, but with *WINUSR*, the X and Y values will depend on the specified Z value.

**Example 8.4**     *Converting locator input coordinates.*

```
C  *****
C  COMPLETE WORKING AXAMPLE
C  *****
C
C      REAL WDW(4), VP(4)
C
C  Read device number, window and viewport.
C
C      READ *, IDEV, WDW, VP
C      CALL GPGS
C      CALL NITDEV(IDEV)
C      CALL WINDW(WDW)
C      CALL VPORT(VP)
C
C  Specify (a 2D) transformation, open picture segment.
C
C      CALL ROTAD(10.0, 3)
C      CALL BGNPIC(1)
C      IVIS=0
C
C  Digitize 10 points.
C
C      INVRT=1
C      DO 2000 I=1,100
C          CALL REQLOC(201, XNDC, YNDC)
C
C  Convert the returned position to user coordinates,
C  must first convert to window coordinates.
C
C      CALL NDCWIN(XNDC, YNDC, 0.0, XWIN, YWIN, ZWIN)
C      CALL WINUSR(INVRT, XWIN, YWIN, 0.0, XUSR, YUSR, ZUSR)
C
C  Mark that there is no need for inverting the
C  transformation matrix more than once.
C
C      INVRT=0
C      CALL LINE(XUSR, YUSR, IVIS)
C      IVIS = 1
C  2000 CONTINUE
C      CALL ENDPIC
C      CALL RLSDEV(IDEV)
C      END
```



# Chapter 9

## Defining Line Patterns and Representation

---

As described in **Chapter 4**, a *linetype* is specified with all GPGS-F routines used to draw single lines, polylines and arcs. This linetype is sent to the device driver in use, and the resulting line pattern depends on the capabilities of the driver. Thus, using different linetypes may be used to distinguish between parts of a drawing, but the line pattern itself is not controlled by the application.

With some applications it is essential that a given linetype appear identical on different devices. Other applications may need linetypes that are defined by some (inter)national standard. For these purposes, GPGS-F contains a family of routines that makes it possible to define the pattern and representation of linetypes.

**Line pattern and representation may be defined for linetypes 6 to 15.**

When a pattern and/or representation is defined for a given linetype, lines using that linetype are buffered in GPGS-F. This internal buffer is emptied either when a line with a different linetype is drawn, a move is performed, or the picture segment is closed. The buffering is needed to compute a nice transition between connected lines.

All lengths and widths specified are given in window coordinates. The window and viewport should have the same X to Y ratio to avoid distortion of the patterns. If these ratios are not equal the pattern length and line width will vary depending on the slope of the line.

Quite often, the length of a pattern to be defined is to have a given physical length. To achieve this, the *DATDEV* (see **page 5-1**) and *NDCWIN* (see **page 8-14**) routines may be used to find the value in window coordinates corresponding to a given physical unit. An example is shown below.

### **Example 9.1**     *Finding a window unit equalling 1 cm.*

```
REAL FARR(4)
C
C 1 cm given in meters (the unit returned by DATDEV).
  UNIT = 0.01
C
  CALL DATDEV( Idum, 0, FARR, 4)
  CALL NDCWIN( 0.0, 0.0, 0.0, XZ, YZ, ZZ)
  CALL NDCWIN(UNIT/FARR(4), UNIT/FARR(4), 0.0, XW, YW, ZZ)
C
  XUNIT = XW - XZ
  YUNIT = YW - YZ
C XUNIT and YUNIT now give the value in window coordinates
C that equals 1 cm on the display surface.
```

## 9.1 Defining Line Patterns

A line pattern is defined by

**CALL LPPATT (Ivis, Itypar(1), Rlenar(1), Length)**

**Ivis** is the GPGS-F linetype, within the range 6 to 15, to define. **Itypar** is an array giving the visibility or angle of each line segment of the pattern, and **Rlenar** gives the length in window coordinates of each line segment. **Length** is the number of line segments in the pattern, that is the length of arrays **Itypar** and **Rlenar**.

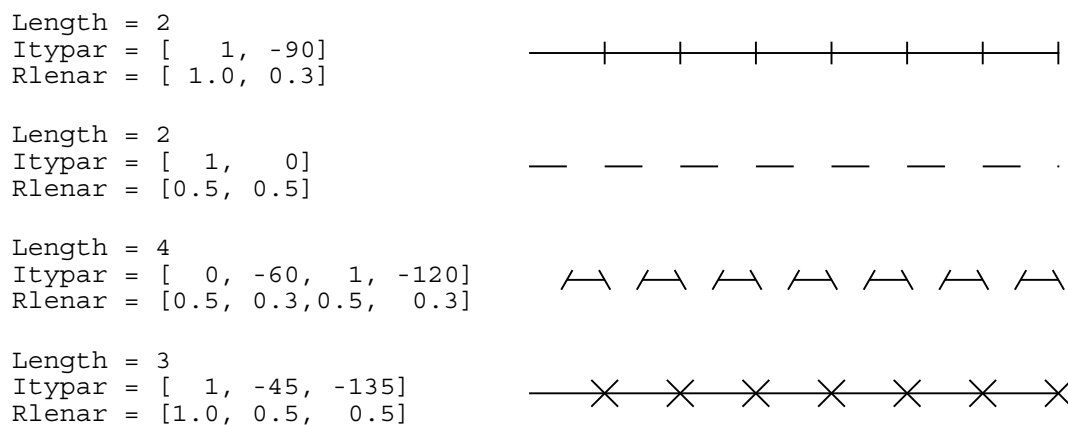
There is no limitation to the number of segments in a single pattern, but the total number of segments in all patterns defined is limited to 100.

The allowed values supplied through **Itypar**, and their meaning, are:

- 0: An invisible line segment, with length given by **Rlenar(i)**.
- 1: A visible solid line segment, with length given by **Rlenar(i)**.
- 1 to -179: A visible solid line segment, drawn with an angle of **-Itypar(i)** degrees between the line segment and the line as specified by its start and end coordinates. The length of the line segment is given by **Rlenar(i)**, and the centre of the line segment is placed on the specified line at the end of the previous line segment.

Some line patterns that may be defined by *LPPATT* are shown in the figure below.

**Figure 9.1** *User defined line patterns.*



The length of the line segments (**Rlenar**) are given in centimetres.

To delete the definition of a given linetype, call *LPPATT* with **Length** set to 1. If that linetype is later used for drawing, the linetype is just passed to the driver in the same way as if *LPPATT* is not used at all.



## 9.2 Defining Line Representation

Different representation of lines are selected by calling one of the routines described in this section. By line representation is meant how the line is drawn, by a single stroke or multiple strokes, and in the last case, the distance between strokes.

By default, lines are drawn by a single stroke line, in the same way as predefined linetypes. If some other representation has been used, this condition may be reset by

**CALL LPGPGS (lvis)**

where **lvis** is the linetype.

Drawing lines by the use of a number of parallel lines is selected by

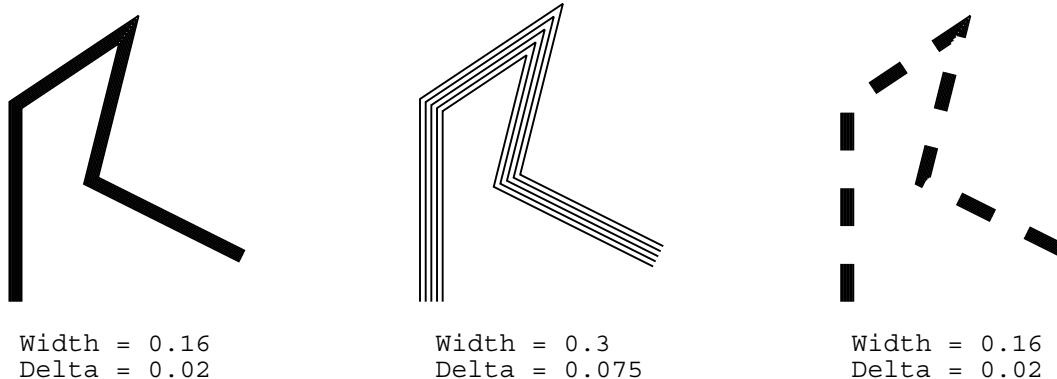
**CALL LPPARA (lvis, Width, Delta)**

where **lvis** is the linetype, **Width** is the total line width and **Delta** is the distance between each line drawn. If **Delta** is selected to be less than the width of a single stroke line, the line will appear thick and solid, even on line drawing devices like pen plotters.

Thick lines may also be selected by applying a linewidth scale factor to lines, by using the *LINWID* routine described on **page 13-1**. This feature is however device dependent, and is mainly provided by raster devices.

### **Figure 9.2** *Lines drawn using a number of parallel lines.*

The width and delta values are given in centimetres,  
the linetype of the right sequence is defined by *LPPATT*



As the angle between two lines to be drawn approaches 180 degrees, the outermost of the parallel lines will extend far beyond the specified line. To avoid this, a 'cut-off' distance is set. This is controlled by the *LPSET* routine described on **page 9-5**.

As an alternative, parallel lines may be drawn with rounded corners and endpoints. This is selected by

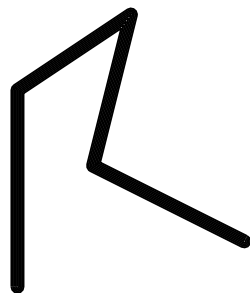
**CALL LPHOTD (Ivis, Width, Delta)**

with the same argument definitions as with *LPPARA*. This kind of lines are often referred to as hot dog lines.

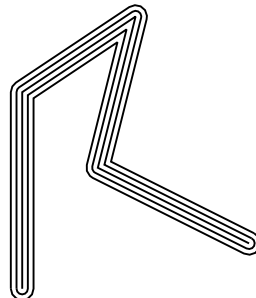
The rounding of corners and endpoints are done by drawing circular arcs approximated by line segments each covering a 30 degree arc. This value may be changed by using the *LPSET* routine described on page 9-5.

**Figure 9.3** *Examples of hot dog lines.*

The width and delta values are given in centimetres



Width = 0.16  
Delta = 0.02



Width = 0.3  
Delta = 0.075

A single line with a given offset from the specified line may be selected by

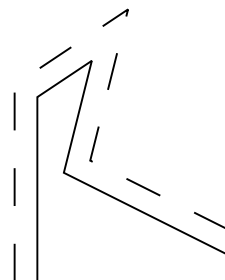
**CALL LPOFFS (Ivis, Offset)**

where **Ivis** is the linetype to define, and **Offset** specifies the distance from the specified line. A positive **Offset** is defined to be to the 'right' of the line specified, when looked at from the startpoint towards the endpoint.

**Figure 9.4** *Offset lines.*

The dashed lines are drawn with no offset.

The solid lines are drawn by using the same coordinates as the dashed lines, but the offset has been set to 0.3 (in this case centimetres).



### 9.2.1 Line Representation Parameters

Two parameters used when drawing parallel, hot dog and offset lines is controlled by the user. Their values are set by

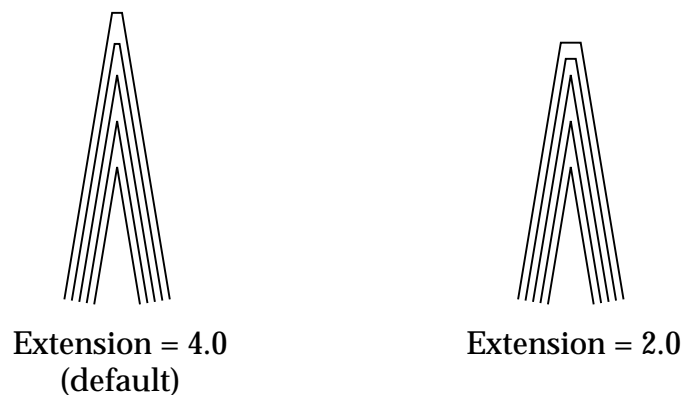
**CALL LPSET (Cid, Value)**

**Cid** is a character string specifying which parameter to set, and **Value** is the new parameter value.

**Cid** = 'Angle' (2 first letters significant) specifies the approximation angle when drawing circular arcs rounding hot dog lines at corners and endpoints. **Value** gives the angle in degrees. Default value is 30.

**Cid** = 'EXtend' (2 first letters significant) specifies the maximum extension of parallel and offset lines beyond their neighbour line (for offset lines the extension is measured from the specified line). **Value** gives this extension as a multiple of the distance between the line being drawn and its neighbour. Default value is 4.

**Figure 9.5** *Reducing the 'cut-off' distance of parallel lines.*





# Chapter 10

## Polylines and Curves

---

### 10.1 Polylines

To reduce the effort for both the user and the system, several lines may be passed to GPGS-F at a time by the use of the *TAB..* family of routines.

Different suffices are used to specify the type of coordinates:

- I* integer coordinates
- L* floating point coordinates
- R* coordinates relative to the previous point
- 3** 3-dimensional polylines

The following polyline routines are available:

**CALL TABL (Xarr(1), Yarr(1), Lthi, Ivis)**

**CALL TABLR (Dxarr(1), Dyarr(1), Lthi, Ivis)**

**CALL TABL3 (Xarr(1), Yarr(1), Zarr(1), Lthi, Ivis)**

**CALL TABLR3 (Dxarr(1), Dyarr(1), Dzarr(1), Lthi, Ivis)**

**CALL TABI (Ixarr(1), Iyarr(1), Lthi, Ivis)**

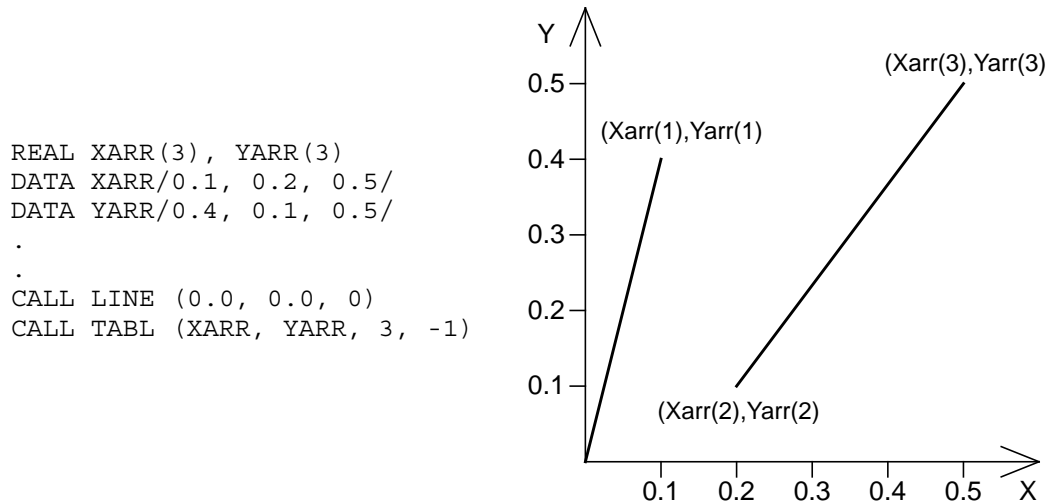
**CALL TABIR (Idxarr(1), Idyarr(1), Lthi, Ivis)**

**CALL TABI3 (Ixarr(1), Iyarr(1), Izarr(1), Lthi, Ivis)**

**CALL TABIR3 (Idxarr(1), Idyarr(1), Idzarr(1), Lthi, Ivis)**

**Xarr, Yarr, Zarr** and **Ixarr, Iyarr, Izarr** are arrays of absolute coordinates. **Dxarr, Dyarr, Dzarr** and **Idxarr, Idyarr, Idzarr** are arrays of relative coordinates. The number of entries in the coordinate arrays is given by **Lthi**.

**Ivis** specifies the linetype to use when drawing the polyline (see **page 4-1**). If the value of **Ivis** is negative every second line will be invisible, while the rest is drawn using linetype **-Ivis**, starting with the first line.

**Figure 10.1** *Using negative linetype with polyline drawing.*

The start position of a polyline is the current position, i.e. the first line is drawn from the current position to the point given by the first array elements (as shown by the figure above). If the polyline is to start at the point given by the first array elements, this is easily achieved by:

```

CALL LINE (XARR(1), YARR(1), 0)
CALL TABL (XARR(2), YARR(2), LTH-1, IVIS)

```

### 10.1.1 Automatic Value or Index Increment

If there is a fixed increment between values along one of the axes, building an array containing these values is unnecessary. Instead, the increment may be supplied to the *TABL.* routine. Before doing this, the *autoincrement indicators* must be set by

**CALL AUTOX (Mx, My)**

or

**CALL AUTOX3 (Mx, My, Mz)**

If one or more of the **Mx**, **My** or **Mz** arguments are set to zero, this implies that a increment value is supplied to the next *TABL.* routine instead of an array for the given axis.

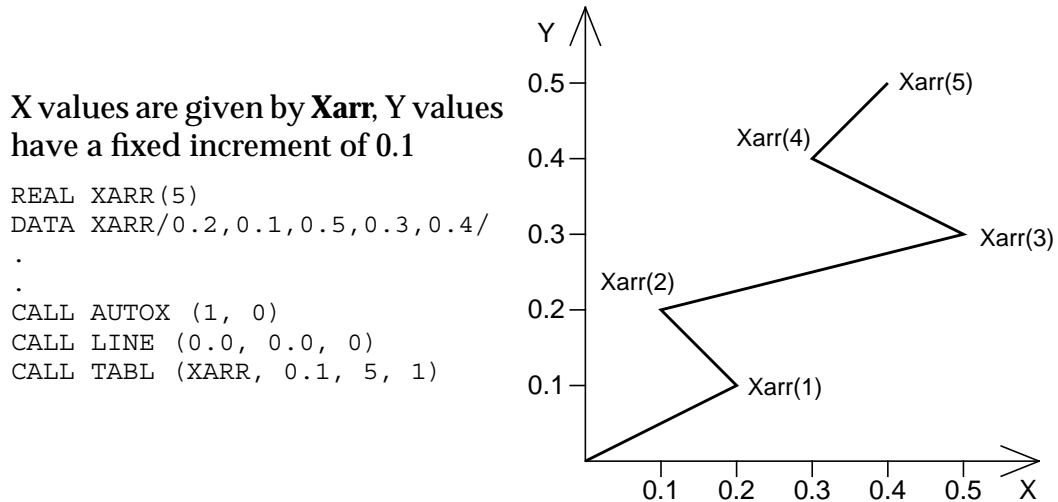
If the autoincrement indicators are set to a value unequal to zero, this marks the *index increment* to use for the corresponding array, e.g. if **Mx** is set to 2, every second element from the X array will be used.

The default values of the autoincrement indicators are 1, i.e. every element from the coordinate arrays are used.

Note that the increment values set by *AUTOX(3)* applies to the **next** call to a *TABL.* routine only. After the polyline is drawn, the indicators are reset to their default values.

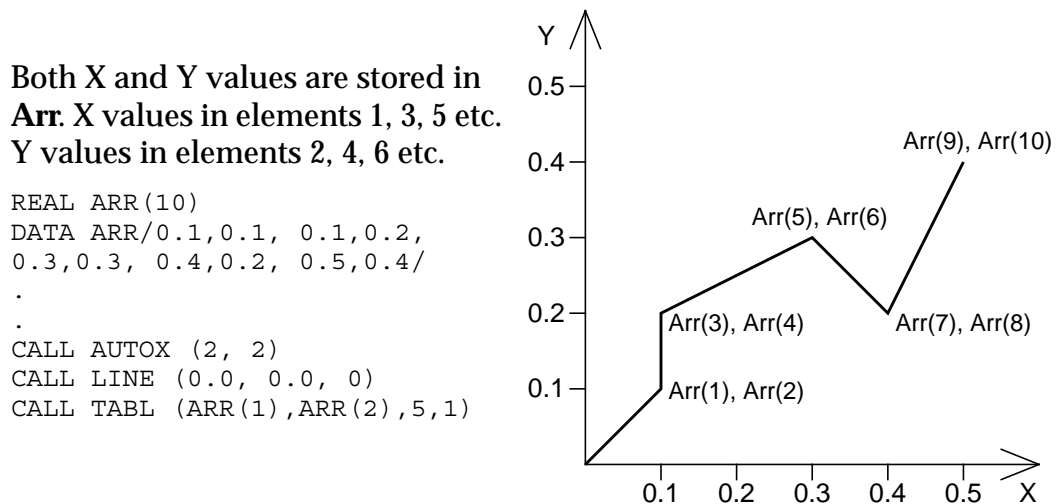
In addition to polylines, automatic *value* increment may be applied when drawing parameterized curves (see **page 10-4**), while automatic *index* increment may be applied when drawing polygons (see **page 12-3**).

**Figure 10.2** *Polyline with value increment.*



Index increment is not as commonly used as value increment. There are however some cases where this is useful, such as examining large arrays by drawing just some of the data values, or if drawing a polyline based on an array containing both the X and Y values stored in alternate entries.

**Figure 10.3** *Polyline with index increment.*



## 10.2 Parameterized Curves

If the coordinates of a curve are generated by external functions, GPGS-F provides routines that will draw the curve directly, without first storing the coordinates in arrays. There are however some limitations to these routines. There must be individual functions for generating the X, Y and Z value of each point, and these functions can not have more than one input argument.

The following routines are available:

**CALL CURV (Fx, Fy, Plowl, Uppl, Step, Ivis)**

**CALL CURVR (Dfx, Dfy, Plowl, Uppl, Step, Ivis)**

**CALL CURV3 (Fx, Fy, Fz, Plowl, Uppl, Step, Ivis)**

**CALL CURVR3 (Dfx, Dfy, Dfz, Plowl, Uppl, Step, Ivis)**

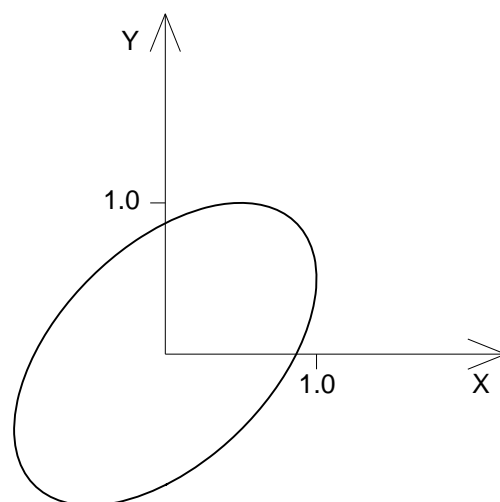
**Fx, Fy, Fz** are functions that will generate the X, Y, Z values, in absolute user coordinates, of a curve. For each point, the functions will be called in the sequence **Fx, Fy, Fz**. Relative values are generated by **Dfx, Dfy, Dfz**. As with the polyline routines, the 3-dimensional routines have a trailing **3**.

**Plowl, Uppl, Step** are the lower limit, upper limit and step size respectively of the argument passed to the functions. This means that the same argument values are passed to the functions. **Ivis** is the GPGS-F linetype to use for drawing.

**Figure 10.4** *Curve generated by intrinsic and external functions.*

The X values are generated by the intrinsic SIN function, while the Y values are generated by the application function SINY

```
EXTERNAL SINY
INTRINSIC SIN
.
.
CALL LINE (0.0, SINY(0.0), 0)
CALL CURV (SIN,SINY,0.0,6.3,0.1,1)
---
FUNCTION SINY (VAL)
SINY=SIN (VAL-3.14/3.0)
RETURN
END
```



As with polylines, the first line segment is drawn from the current position.



### 10.2.1 Automatic Value Increment

Similar to the polyline routines, automatic value increment may be used with curves as well. The *AUTOX*/*AUTOX3* routines are used, with the arguments defined as when used with polylines (see [page 10-2](#)).

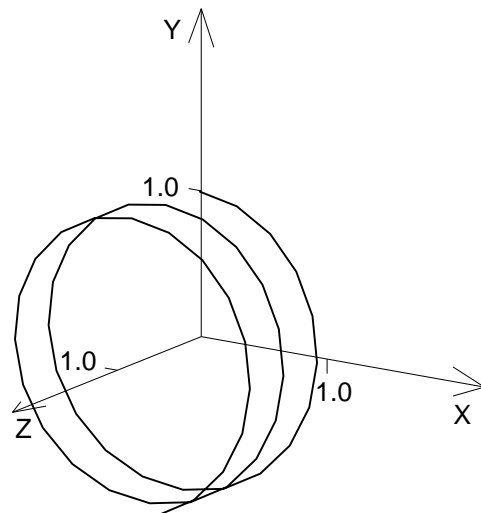
Note however that this is not guaranteed to work with all computers. Internally, GPGS-F will treat the input arguments to the *CURV.* routines either as a function reference or a real value, depending on the current settings of the autoincrement indicators. Some systems may not allow this.

There is no function corresponding to index increment available with the curve routines.

**Figure 10.5** *Curve with value increment in Z direction.*

The X and Y values are generated by the SIN and COS functions, while the Z values have a fixed increment of 0.02, starting at 0.0

```
INTRINSIC SIN, COS
.  
.  
CALL AUTOX3 (1, 1, 0)  
CALL LINE3 (0.0, 1.0, 0.0, 0)  
CALL CURV3 (SIN, COS, 0.02,  
            0.0, 15.7, 0.314, 1)
```





# Chapter 11

## Colour Specification

---

As the reader will have noticed the colour of graphic primitives is not specified through the subroutine calls generating the primitives.

The colour is instead specified by a separate routine, setting a *current colour index*, and this colour index will be used until changed by a new call to the same routine.

A colour index is not a direct colour, but an index into a table giving the actual colour. The reason why colour tables and colour indices are used instead of direct colours, is that this makes it possible to utilize the hardware colour facilities of raster devices, which is the most common device type used with GPGS-F.

The colour tables are kept in the device drivers, or if possible, in the graphic device. As a device is initialized by *NITDEV* (see **page 1-2**), the 8 first entries in the colour table are set to default values common to all devices. These default values are shown in **Table 11.1 (page 11-6)**. There are three different kinds of colour tables, allowing various degrees of manipulation. What kind of colour table is used by a given device depends on the capabilities of that device.

The different kinds of colour tables used by GPGS-F drivers are:

- Fixed:** The contents of the colour table may not be changed. This kind of colour table is used by pen plotters.
- Static:** Changing the definition of a colour index will affect subsequent primitives using the given index, while primitives already drawn will not be affected. This kind of colour table is used by most colour raster plotters.
- Dynamic:** Changing the colour table will change the colour of primitives already drawn, as well as subsequent primitives. This kind of colour table is used by colour raster terminals.

With devices using a fixed colour table, there is, as the name applies, a fixed relation between the colour index and the actual colour. For a pen plotter, GPGS-F defines this relation to be between the colour index and the physical pens of the plotter, i.e. colour index *n* means pen number *n*. Thus, to get the requested colour, the operator must ensure that the pens are mounted in the sequence given by the default colour table.

The length of the colour table is device dependant. Entries are numbered from zero upwards, with zero being the background colour. For raster terminals, the length of the colour table is the maximum number of colours that may be displayed on the screen at the same time. Most common values are 8, 16 and 256. For pen plotters, the length of the colour table equals the number of pens available. The length of the colour table of the device in use may be obtained by the *DATDEV* routine described on **page 23-4**.

GPGS-F can not handle colour tables with more than 32768 entries.

## 11.1 Colour Index Selection

The current colour index is selected by

**CALL COTIND (Ind)**

where **Ind** is the colour index to be used for drawing. Selecting a index higher than the maximum available for the active device will result in a device dependant colour index being used (normally 1).

As mentioned earlier, the colour index selected will remain *active* until changed by a new call to *COTIND*. The colour index is however reset to its default value of 1 when a new picture segment is opened by *BGNPIC* (see [page 3-1](#)).

The effect of using colour index 0 (the background colour) for drawing will depend on the device in use. With pen plotters, and other line drawing devices, using colour index 0 will have no visual effect. With raster devices, colour index 0 may be used to erase previously drawn primitives. This feature is further described in [Chapter 12](#).

For compatibility with previous versions of GPGS-F, a direct colour may be selected by

**CALL COLOUR (Icol)**

where **Icol** is a direct colour code. The allowed values are:

0	Default foreground	40	Yellow
10	Magenta	50	Cyan
20	Blue	60	Red
30	Green	100	Default background

With plotters, the default background is white, default foreground black. With most terminals, the default background is black, default foreground white.

## 11.2 Colour Models

The contents of the colour table may be changed at any time, giving the effect described on [page 11-1](#) for the different kinds of colour tables.

To change the colour table, GPGS-F allows the application programmer to choose among three of the colour models most commonly used with computer graphics. These colour models are **RGB** (Red, Green, Blue), **HLS** (Hue, Lightness, Saturation) and **HSV** (Hue, Saturation, Value). Detailed descriptions are given in the following sections.

Which colour model to use will mainly depend on how familiar the application programmer is with the different models.

**Table 11.1 (page 11-6)** shows how the default colour table is defined using each colour model.

### 11.2.1 RGB Colour Model

The most commonly used colour model, both with GPGS-F and other graphic systems, is the **RedGreenBlue** model, mainly because it is easiest to use.

With the RGB colour model, a colour is specified as a mixture of red, green and blue components by

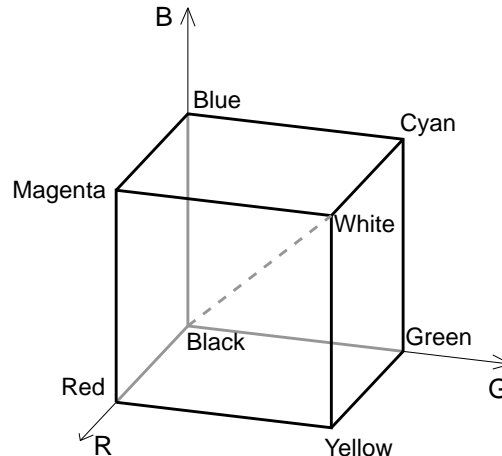
**CALL COTRGB (Ind1, Red(1), Green(1), Blue(1), Lth)**

**Ind1** is the first colour index, and **Lth** the number of indices to change.

The **Red**, **Green** and **Blue** arrays specifies the amount of red, green and blue for each colour index. The length of the arrays must then be at least equal to **Lth**. Each colour component is given as a real number ranging from 0.0 (no contribution) to 1.0 (full saturation).

The RGB colour model is often visualized as a unit cube, with the three components along the axes, as shown below.

**Figure 11.1** *The RGB colour model.*



## 11.2.2 The HLS Colour Model

The second colour model that may be used with GPGS-F, is the **HueLightnessSaturation** model. With this model, colours are specified by

**CALL COTHLS (Ind1, Hue(1), Rlight(1), Sat(1), Lth)**

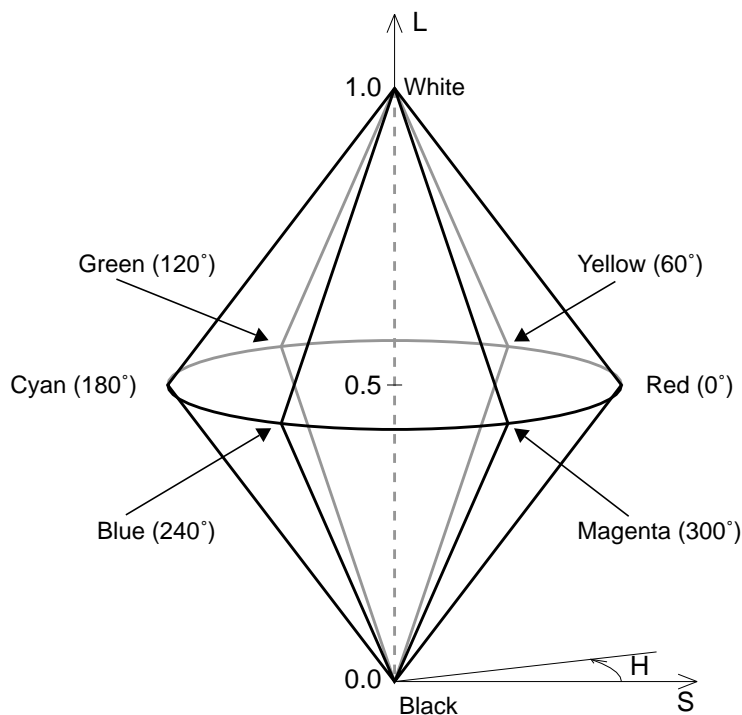
As with *COTRGB*, **Ind1** is the first colour index, and **Lth** the number of indices to change.

The colours are defined by the **Hue**, **Rlight** and **Sat** arrays, as visualized by **Figure 11.2**.

**Hue** specifies the different pure colours, given as degrees from 0.0 to 360.0, **Rlight** specifies the lightness (intensity), ranging from 0.0 (black) to 1.0 (white), and **Sat** specifies the saturation, measured radially from the vertical axis, with 0.0 at the axis (resulting in grey) and 1.0 for full saturation.

Pure colours are specified by setting **Rlight** to 0.5 and **Sat** to 1.0

**Figure 11.2** *The HLS colour model.*



### 11.2.3 The HSV Colour Model

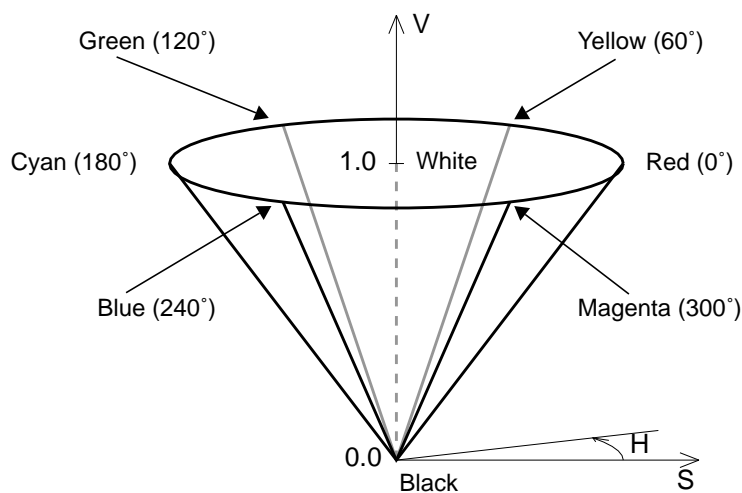
The third colour model that may be used with GPGS-F is the **HueSaturationValue** model. With this model, colours are specified by

**CALL COTHSV (Ind1, Hue(1), Sat(1), Val(1), Lth)**

As with the other colour models, **Ind1** is the first colour index, and **Lth** the number of indices to change.

The colour model is visualized in **Figure 11.3**. Pure colours are selected by **Hue**, with both **Sat** and **Val** set to 1.0. White pigment is added by decreasing **Sat**, while black pigment is added by decreasing **Val**.

**Figure 11.3** *The HSV colour model.*



**Table 11.1** *Default GPGS-F colour table.*

Index	Colour	R	G	B	H	L	S	H	S	V
0	Black <sup>1</sup>	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	White	1.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0
2	Red	1.0	0.0	0.0	0.0	0.5	1.0	0.0	1.0	1.0
3	Green	0.0	1.0	0.0	120.0	0.5	1.0	120.0	1.0	1.0
4	Blue	0.0	0.0	1.0	240.0	0.5	1.0	240.0	1.0	1.0
5	Cyan	0.0	1.0	1.0	180.0	0.5	1.0	180.0	1.0	1.0
6	Magenta	1.0	0.0	1.0	300.0	0.5	1.0	300.0	1.0	1.0
7	Yellow	1.0	1.0	0.0	60.0	0.5	1.0	60.0	1.0	1.0

1. With some devices (plotters and some displays), the default background is white and the default foreground is black.

## 11.3 Monochrome Devices

Internally, GPGS-F always converts colour specifications to RGB before these are sent to the device drivers. Monochrome devices will compute the resulting grey level based on the RGB values by:

$$\text{Grey} = 30\% \text{ Red} + 59\% \text{ Green} + 11\% \text{ Blue}$$

resulting in a value between 0.0 (black) and 1.0 (white).

Grey levels may be specified directly by using the colour definition routines as follows:

**RGB model:** Use equal amounts of red, green and blue.

**Red = Green = Blue = 0.0** gives black, **1.0** gives white.

**HLS model:** Set **Sat** to 0.0, and use **Rlight** to specify the grey level.

**Rlight = 0.0** gives black, **Rlight = 1.0** gives white.

The **Hue** value have no influence.

**HSV model:** Set **Sat** to 0.0, and use **Val** to specify the grey level.

**Val = 0.0** gives black, **Val = 1.0** gives white.

The **Hue** value have no influence.

Colour tables with different grey levels may of course also be defined for colour devices. This is especially useful if plots are to be copied in monochrome mode, as it is quite difficult to define colours that are well distinguishable when converted to grey levels.



# Chapter 12

## Raster Graphics

---

Originally, GPGS-F was a system for line drawing only. As the use of raster devices became more common, new routines were added to utilize the possibilities these devices offer for graphics programming.

### 12.1 Raster Graphics Programming

A program written for a line drawing device need not be changed to run on a raster device. However, raster devices offer features that need special programming techniques to be utilized.

As described in **Chapter 11**, colour table manipulation is possible with raster devices only. Raster devices allow area filling, using solid colours or patterns, and the visual effect of drawing overlapping primitives is quite different between raster and line drawing devices.

When a primitive is drawn on a raster device, this will obscure previously drawn primitives. This means that the drawing sequence may be significant for the final result, especially if filled areas are included. On line drawing devices, the colour of overlapping primitives will be a mixture of the colours used, i.e. changing the drawing sequence will have little or no visual effect.

The fact that a primitive obscures previously drawn primitives allows parts of a drawing to be erased by just redrawing these parts using the background colour. This effect may be used to introduce an element of dynamics into GPGS-F applications running on a raster terminal. By using *retained segments*, it is possible to control the visibility of individual picture segments, and also to physically move segments on the screen. The use of retained segments is described in **Chapter 14** and **Chapter 16**.

Even if retained segments are not used, some of the same effects may be achieved by the application program directly, by using the background colour to erase parts of the picture, or by manipulating the colour table if this is dynamic (see **page 11-1**).

**Example 12.1** *Selective erase on a raster terminal.*

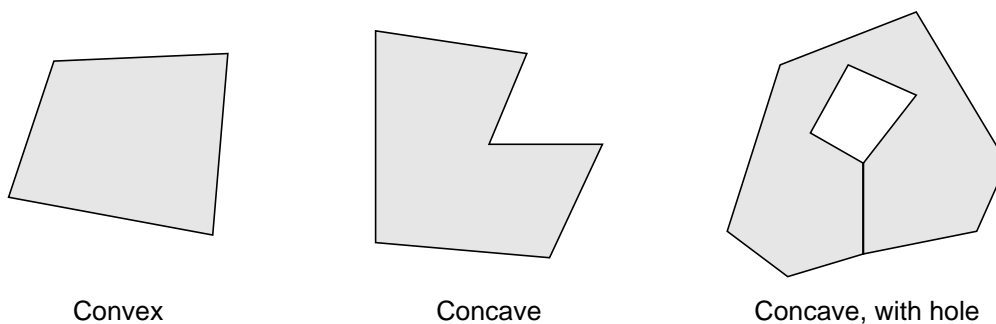
```
C
C Draw two parts in 2 different colours.
C (DRAW is a user-supplied routine.)
C
C     CALL COTIND(2)
C     CALL DRAW(PART1)
C     CALL COTIND(3)
C     CALL DRAW(PART2)
C
C Erase part 2 by drawing with the background colour.
C
C     CALL COTIND(0)
C     CALL DRAW(PART2)
```

**Example 12.2** *Manipulating the colour table.*

```
C
C Draw two parts in 2 different colours.
C
C     CALL COTIND(2)
C     CALL DRAW(PART1)
C     CALL COTIND(3)
C     CALL DRAW(PART2)
C
C 'Erase' PART2 by setting its colour equal to the
C background colour (assumed here to be black)
C
C     CALL COTRGB(3, 0.0, 0.0, 0.0, 1)
```

## 12.2 Polygons

A polygon is defined as an area enclosed by a number of edges. GPGS-F allows both convex (a polygon where no straight lines will cross more than two edges) and concave polygons, and even polygons with holes.

**Figure 12.1** *Polygon shapes.*

If the polygon is to be filled by hardware, either by default or as a result of user selection (routine **SOPOL** on **page 12-6**), some devices may have restrictions on polygon shape.

### 12.2.1 Polygon Drawing

A 2 dimensional polygon is drawn by

**CALL POLY (Xarr(1), Yarr(1), Lth, ltype)**

where the vertices are given by **Xarr** and **Yarr** as absolute coordinates, or by

**CALL POLYR (Dxarr(1), Dyarr(1), Lth, ltype)**

where the vertices are given by **Dxarr**, **Dyarr** as relative coordinates. The **Lth** argument is described below, the **ltype** argument is described on the next page.

2 dimensional polygons are drawn in the XY plane at the current Z value.

A 3 dimensional polygon is drawn by

**CALL POLY3 (Xarr(1), Yarr(1), Zarr(1), Lth, ltype)**

where the vertices are given by **Xarr**, **Yarr** and **Zarr** as absolute coordinates, or by

**CALL POLYR3 (Dxarr(1), Dyarr(1), Dzarr(1), Lth, ltype)**

where the vertices are given by **Dxarr**, **Dyarr** and **Dzarr** as relative coordinates.

GPGS-F restricts 3 dimensional polygons to be plane. To meet this restriction, most programmers will find it easier to define 2 dimensional polygons and transform these into the 3D space, using the routines described in **Chapter 6**, than to define 3 dimensional polygons. If a polygon is not plane, GPGS-F will still try to draw the polygon, but the result depends on the type of the polygon, and whether it is filled by hardware or software.

The **Lth** argument to the polygon routines gives the number of vertices of the polygon. Polygon closure is implicit, i.e. the edge from the last to the first vertex is added by GPGS-F. As the polygon during computation is kept in internal GPGS-F arrays, there is a limit on the number of vertices allowed. This limit is given by the system parameter **MPOLSZ** (see **page A-1**), which may be changed by the site responsible for GPGS-F.

When relative coordinates are given, the first vertex is specified relative to the current position, while the rest is specified relative to the previous vertex. When absolute coordinates are given, the current position is not used. After the polygon is drawn, the current position is set to the first vertex of the polygon in both cases.

Automatic index or value increment may be used with polygons in the same way as with polylines, using the *AUTOX* or *AUTOX3* routines described on **page 10-2**.

### 12.2.2 Interior Style

The **ltype** argument given through the polygon drawing routines, specifies the *interior style* of the polygon (also referred to as *polygon type*), i.e. how the polygon is rendered (filled).

**Table 12.1** *Interior styles available with GPGS-F*

<b>Itype value</b>	<b>Interior style</b>	<b>Description</b>
1	Solid	Uniform solid fill using the current colour index (see <b>page 11-2</b> ). This kind of filling is always left to the device hardware. If the device is not capable of performing solid fill, only the perimeter (outline) of the polygon will be drawn.
2	Hollow	The perimeter is drawn by solid lines, using the current colour index. No filling is applied.
3 to 8191	Textured	<p>The interior is rendered by a pattern or with hatch lines. In this case, <b>ltype</b> is an index into a pattern or hatch table, describing the pattern or hatch style. The allowable range of <b>ltype</b> depends on the texture quality selected by the <i>SOPPOL</i> routine described on <b>page 12-6</b>.</p> <p>How to select pattern or hatch rendering, how to define the pattern and hatch tables, and all other aspects of texture rendering is described in <b>section 12.2.4</b>.</p>

### 12.2.3 Perimeter Drawing

By default, when solid and textured polygons are generated, the perimeter (outline) is not drawn. This may however be selected by

**CALL PRCIND (Ind)**

where **Ind** is the colour index to use for the perimeter. If this is set to -1, the default condition is reset.

The visual effect of selecting perimeter drawing is the same as drawing the same polygon twice, first using solid or textured fill, secondly as a hollow polygon.

## 12.2.4 Texture Rendering

As described in **Table 12.1**, **ltype** values 3 to 8191 are used for both patterned and hatched polygons. Which texture type to use is selected by

**CALL PITYP (lptyp)**

where **lptyp**=1 selects hatch rendering and **lptyp**=2 selects pattern rendering. If **lptyp** is set to 0, the default texture type used by the current device is selected. This default may be requested by the *DATDEV* routine described on **page 23-4**.

Hatch and pattern texture is defined as follows:

**Hatching:** The polygon is filled by a number of parallel lines. The angle, and the distance between the lines, is selected by separate routines.

The lines are drawn using the current colour index.

**Patterning:** The polygon is filled by repeating a pattern horizontally and vertically until the polygon is completely filled.

A pattern is a two dimensional array of colour indices. This kind of rendering is available only with devices that are capable of setting the colour index of individual pixels.

### 12.2.4.1 Texture Quality

With GPGS-F, polygon texture rendering is designed to serve two purposes.

The most common purpose is just to get polygons with different appearance.

With raster devices this may be accomplished by using different colours for solid fill, or using different patterns. The pattern itself, as well as its size and transformation need not be important as the main goal is to get different looking areas.

With line drawing devices, different appearance is achieved by using hatch lines with different angles, density and colours. The actual values need not be important, as long as they are selected so that areas may be distinguished.

The other purpose for using texture rendering is to get an exact mapping of the pattern or hatch style onto polygons. E.g. if a brick wall is to be drawn, a brick may be defined as a pattern. The size and position of the brick must be defined relative to the wall, so that when the wall is transformed, the bricks are transformed with the wall.

With hatch line rendering, exact mapping means that the hatch angle and the distance between the hatch lines are transformed according to the transformation of the polygon that is to be rendered.

Depending on the requirements, the quality of the texture may be selected by

**CALL SOFPOL (lqual)**

where **lqual** specifies the quality as described by the table below.

**Table 12.2** *Texture quality.*

<b>Iqual</b>	<b>Quality</b>	<b>Description</b>
0	Low	The rendering is always left to the device driver or hardware. If the driver/hardware is not able to perform the selected rendering, only the perimeter of the polygon is drawn.
1	Medium	The device driver/hardware is used if possible. If not, the rendering is performed by GPGS-F software. In the first case, medium quality will be identical to low quality. In the second case, the texture is applied to the <i>transformed</i> polygon. With hatching, this means that the hatch angle and distance are not transformed. With patterns, each pattern cell is mapped to a pixel on the display surface, i.e. the pattern size (set by <i>PISZ</i> , page 12-8) is ignored. Medium quality is the default quality selected when GPGS-F is initialized.
2	High	Rendering is always performed by GPGS-F software. The texture is applied to the untransformed polygon, and subsequently transformed with the polygon.

Parts of the pattern and hatch style definitions are stored in tables accessed by the **ltype** argument through the polygon drawing routines. Additional attributes necessary to complete the definitions are given as global attributes independent of the table entry selected.

Note that when low or medium rendering quality is selected, only parts of the pattern or hatch style definitions will be used. The selected quality also determines the allowable range of the **ltype** argument given through the polygon drawing routines.

Details on these aspects are given in the following subsections, first describing the hatch and pattern tables, then the global attributes available.

A summary of pattern and hatch style definition is given on page 12-9, and a program example on page 12-10.









### 12.2.4.2 Pattern and Hatch Style Tables

The pattern and hatch style tables are divided into 3 logical parts, divided by the table index.

- 3 to 10:** Stored both in GPGS-F and in the device drivers. The entries are predefined as shown by tables **12.3** (patterns) and **12.4** (hatch styles), but may be redefined by the user. These entries may be used with any texture quality.
- 11 to 127:** Stored in the device drivers only. These entries have no predefined content, but may be defined by the user. They may be used with low and medium, but not high quality texture.
- 128 to 8191:** Also stored in the device drivers only. These entries are predefined, and may *not* be redefined. They may be used with low quality texture only.

Although all table entries from 3 to 8191 are said to be stored in the device drivers, the number of entries actually stored will depend on the device. The driver descriptions in **Appendix E** will show how many user definable (entries 3 and up) and predefined (entries 128 and up) patterns / hatch styles are available. shows the predefined patterns defined by the PostScript driver.

**Table 12.3** *Predefined patterns.*

Index	3	4	5	6	7	8	9	10
<b>Pattern definition</b>	<div> <div>01</div> <div>10</div> </div>	<div> <div>01</div> <div>01</div> </div>	<div> <div>12</div> <div>34</div> </div>	<div> <div>1000</div> <div>0101</div> <div>0010</div> <div>0000</div> </div>	<div> <div>1000</div> <div>0100</div> <div>0010</div> <div>0100</div> </div>	<div> <div>0010</div> <div>0010</div> <div>1111</div> <div>0010</div> </div>	<div> <div>0100</div> <div>0000</div> <div>1000</div> <div>0000</div> </div>	<div> <div>0000</div> <div>0100</div> <div>1110</div> <div>0100</div> </div>
<b>Applied to a polygon</b>								

Entries 3 to 127 in the pattern table may be (re)defined by

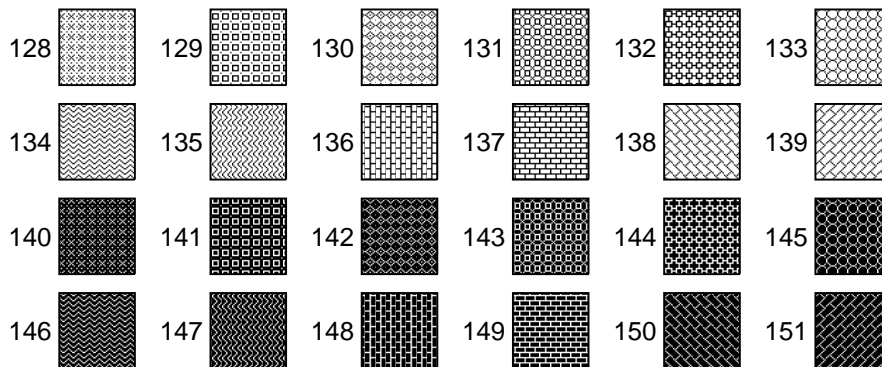
**CALL PATDEF (Index, lcarr(1,1), Nx, Ny)**

where **Index** is the pattern table index. **lcarr** is an array of colour indices, with **Nx** pattern cells in X direction and **Ny** pattern cells in Y direction.

If **Index** is in the range 3 to 10, the pattern is stored in GPGS-F, and sent to the driver when needed. If **Index** is in the range 11 to 127, the definition is sent directly to the driver, and if possible stored in the driver or in the device hardware.

The maximum number of cells in a pattern stored in GPGS-F is limited by the system parameter **MRPSIZ** (see **page A-1**), which may be changed by the site responsible for GPGS-F. For patterns in the range 11 to 127, GPGS-F will not limit the size, but just send the definition to the driver. Whether the driver/hardware will be able to use the definition will then be device dependant. Most devices will limit the number of cells in user defined patterns, common values are 8×8, 16×16 or 32×32.

**Figure 12.2** *Predefined hardware patterns, PostScript driver.*



**Table 12.4** *Predefined hatch styles.*

Index	3	4	5	6	7	8	9	10
Hatch angle	135°	90°	45°	0°	157.5°	112.5°	67.5°	22.5°
Applied to a polygon								

Entries 3 to 127 in the hatch style table may be (re) defined by

**CALL HTCDEF (Index, Angle)**

where **Index** is the table index, and **Angle** is the hatch angle *in radians*, measured counterclockwise from the X axis.

As with patterns, if **Index** is in the range 3 to 10, the definition is stored in GPGS-F, and sent to the driver when needed. If **Index** is in the range 11 to 127, the definition is sent directly to the driver, and if possible stored in the driver or in the device hardware.

### 12.2.4.3 Global Texture Attributes

To complete the definition of patterns and hatch styles, some attributes are needed in addition to the information stored in the pattern and hatch style tables. These attributes are set by two routines that are common to hatched and patterned polygons.

The pattern size and/or hatch density is defined by

**CALL PISIZ (Dx, Dy, Hdist)**

where **Hdist** is the distance between two individual hatch lines, **Dx** and **Dy** is the length of a pattern in X and Y direction respectively. The values are specified in user coordinates, with 0.01 as default value for all three arguments.



If only the pattern size or the hatch density is to be defined, the other arguments may be set to 0.0, which is defined to mean ‘no change’.

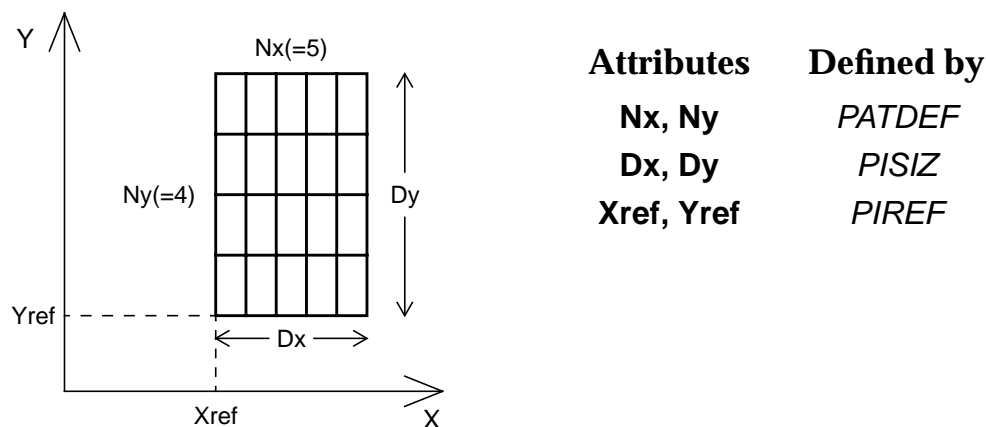
The *reference point* of textured polygons is set by

CALL   PIREF (Xref, Yref)

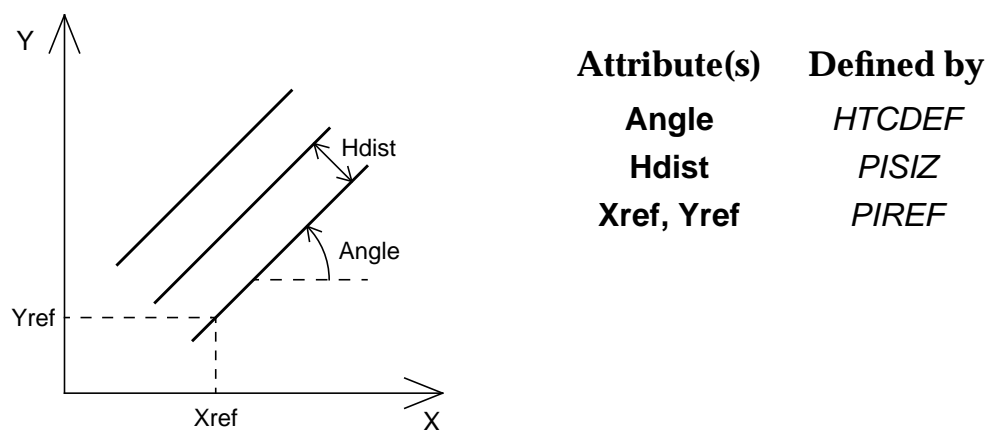
where **Xref** and **Yref** specify this point in user coordinates. For patterned polygons, the pattern is laid out so that the reference point will be at one of the corners of a pattern. For hatched polygons, the hatch lines will be drawn so that one of them passes through the reference point. Note that the reference point need *not* be inside the polygon.

The default values for both **Xref** and **Yref** are 0.0

**Figure 12.3** *Pattern definition, summary.*



**Figure 12.4** *Hatch style definition, summary.*



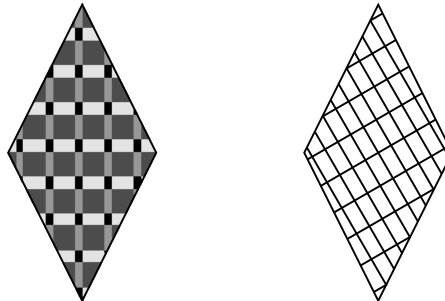
### **Example 12.3**    *User defined pattern and hatch styles.*

```

C *****
C COMPLETE WORKING EXAMPLE.
C *****
C NOTE! A raster device should be used when running this program.
C
      INTEGER IPARR(4,3)
      REAL XPOL(4), YPOL(4)
      DATA IPARR/2,2,2,3, 2,2,2,3, 7,7,7,1/
      DATA XPOL/0.20, 0.25, 0.20, 0.15/
      DATA YPOL/0.20, 0.30, 0.40, 0.30/
C
C Read device number, initialize driver.
C
      READ *, IDEV
      CALL GPGS
      CALL NITDEV(IDEV)
      CALL BGNPIC(1)
C
C Select high quality texture rendering, pattern texture.
C
      CALL SOFPOL(2)
      CALL PITYP(2)
C
C Redefine pattern no. 3
C Set pattern size. Set reference point to centre of polygon.
C
      CALL PATDEF(3, IPARR, 4, 3)
      CALL PISIZ(0.02, 0.025, 0.0)
      CALL PIREF(0.2, 0.3)
C
C Draw a patterned polygon, with perimeter.
C
      CALL PRCIND(1)
      CALL POLY(XPOL, YPOL, 4, 3)
C
C Redefine hatch index 3 to 30 degrees, index 4 to 120 degrees.
C
      PI=3.14159
      CALL HTCDEF(3, PI/6.0)
      CALL HTCDEF(4, PI*4.0/6.0)
C
C Set hatch density and draw hatched (30°)polygon.
C
      CALL PITYP(1)
      CALL PISIZ(0.0, 0.0, 0.02)
      CALL XLAT(0.2, 0.0)
      CALL POLY(XPOL, YPOL, 4, 3)
C
C Change hatch density and draw same polygon once more.
C Perimeter not needed this time.
C
      CALL PISIZ(0.0, 0.0, 0.01)
      CALL PRCIND(-1)
      CALL POLY(XPOL, YPOL, 4, 4)
      CALL ENDPIC
      CALL RLSDEV(IDEV)
      END

```

**Figure 12.5** *User defined pattern and hatch styles (generated by Example 12.3).*



#### **12.2.4.4 Applying Texture to 3D Polygons**

Though all geometric attributes for polygon texture are defined in 2 dimensions, GPGS-F is capable of texturing 3 dimensional polygons as well. This obviously applies to high quality texture only, as low and medium quality texture is applied to transformed polygons, i.e. the 2D projection of 3D polygons.

Texture rendering of a 3 dimensional polygon is performed as follows:

- 1) The polygon is projected into the users XY plane, by first applying an axonometric projection, then translating the first vertex of the polygon to the users origin.
- 2) The texture is applied to the polygon.
- 3) The polygon, with the texture, is transformed back into 3 dimensional space.

However, this does mean that accurate specification of the texture for 3D polygons is not possible. Thus, the recommended method is to define all polygons as 2 dimensional, and transform these into 3D space using the GPGS-F transformation routines.

## 12.3 Pixel Arrays

A pixel array is a drawing primitive that is defined the same way as a polygon pattern, i.e. a two dimensional array of colour indices.

Pixel arrays will be correctly drawn only by devices capable of setting the colour of individual pixels. If drawn with other devices, just the boundary is drawn using the current colour index.

Pixel arrays are drawn by

**CALL PIXARR (Width, Height, Idimx, Indarr(1,1), Nx, Ny)**

**Width** and **Height** defines a rectangular area in user coordinates, into which the pixel array is to be mapped. The lower left corner of the pixel array (**Indarr(1,Ny)**) is placed at the current position, and it is drawn in the user's XY plane at the current Z coordinate. The current position is left unchanged by *PIXARR*.

**Indarr** is the pixel array, **Nx** and **Ny** is the number of cells to draw in X and Y direction. **Idimx** must be equal to the first dimension of the pixel array *as defined*. The reason why this is given in addition to **Nx**, is that it makes it possible to draw subareas of large pixel arrays, as shown by **Example 12.5**.

### 12.3.1 Software / Hardware Generation

As with most GPGS-F primitives, the user may select whether pixel arrays are to be drawn by software or hardware. This is selected by

**CALL SOFPIX (lsw)**

where **lsw**=0 selects hardware, **lsw**=1 selects software. The default value is 0.

A software pixel array is subject to all GPGS-F transformations. The pixel array is scaled according to the width and height specified, then sampled to find the colour index for each pixel. Thus, the number of colour index values that is transferred to the driver depends on the height and width of the pixel array, and the resolution of the device.

When hardware is selected, the height, width and the pixel array is sent to the device driver. Some drivers are capable of scaling the pixel array to the given size, others will just map one cell of the pixel array to one pixel on the display surface. In any case, the lower left corner will be transformed to the specified position, but no other transformations are applied.

If clipping is switched on, by using the *CLICTL* routine described on **page 2-4**, exact clipping of software pixel arrays will be performed.

Hardware pixel arrays are not clipped, but are *ignored* if parts of the destination area is outside the window.

### Example 12.4 *Hardware and software pixel array drawing.*

```

C *****
C COMPLETE WORKING EXAMPLE
C *****
C NOTE! A raster device should be used when running this program
C
      INTEGER IPIX(8,8)
      DATA IPIX/ 1,1,1,2,2,3,3,3, 1,1,1,2,2,3,3,3
+                , 1,1,1,2,2,3,3,3, 2,2,2,2,2,2,2,2
+                , 2,2,2,2,2,2,2,2, 5,5,5,2,2,7,7,7
+                , 5,5,5,2,2,7,7,7, 5,5,5,2,2,7,7,7/
C
C Initialize device
C
      READ *, IDEV
      CALL GPGS
      CALL NITDEV(IDEV)
      CALL BGNPIC(1)
C
C Apply some transformations. Draw a hardware pixel array,
C and its defined outline.
      CALL XLAT(0.3, 0.2)
      CALL ROTAD(30.0, 1)
      CALL ROTAD(30.0, 3)
      CALL LINE(0.0, 0.0, 0)
      WIDTH = 0.075
      HEIGHT= 0.1
      CALL PIXARR(WIDTH, HEIGHT, 8, IPIX, 8, 8)
C
      CALL LINER(0.0, HEIGHT, 1)
      CALL LINER(WIDTH, 0.0, 1)
      CALL LINER(0.0, -HEIGHT, 1)
      CALL LINER(-WIDTH, 0.0, 1)
      CALL IDEN
C
C Draw same pixel array, to the right, using software.
      CALL SOFPIX(1)
      CALL XLAT(0.6, 0.2)
      CALL ROTAD(30.0, 1)
      CALL ROTAD(30.0, 3)
      CALL LINE(0.0, 0.0, 0)
      CALL PIXARR(WIDTH, HEIGHT, 8, IPIX, 8, 8)
      CALL LINER(0.0, HEIGHT, 1)
      CALL LINER(WIDTH, 0.0, 1)
      CALL LINER(0.0, -HEIGHT, 1)
      CALL LINER(-WIDTH, 0.0, 1)
C
      CALL ENDPIC
      CALL RLSDEV(IDEV)
      END

```

**Figure 12.6** *Pixel arrays (generated by the example above).*



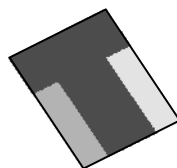
### **Example 12.5    *Pixel array subarea.***

```

C *****
C COMPLETE WORKING EXAMPLE (mainly same code as previous example)
C *****
C NOTE! A raster device be used when running this program
C
      INTEGER IPIX(8,8)
      DATA IPIX/ 1,1,1,2,2,3,3,3, 1,1,1,2,2,3,3,3
+                , 1,1,1,2,2,3,3,3, 2,2,2,2,2,2,2,2
+                , 2,2,2,2,2,2,2,2, 5,5,5,2,2,7,7,7
+                , 5,5,5,2,2,7,7,7, 5,5,5,2,2,7,7,7/
C
C Initialize device
C
      READ *, IDEV
      CALL GPGS
      CALL NITDEV(IDEV)
      CALL BGNPIC(1)
      WIDTH = 0.075
      HEIGHT= 0.1
C
C Apply some transformations.
C
      CALL XLAT(0.3, 0.2)
      CALL ROTAD(30.0, 1)
      CALL ROTAD(30.0, 3)
      CALL LINE(0.0, 0.0, 0)
C
C Draw a 4 by 3 cell subarea of IPIX,
C with cell (3,5) as upper left corner
C
      CALL SOFPIX(1)
      CALL PIXARR(WIDTH, HEIGHT, 8, IPIX(3,5), 4, 3)
C
C Draw the outline of the specified pixel array area.
C
      CALL LINER(0.0, HEIGHT, 1)
      CALL LINER(WIDTH, 0.0, 1)
      CALL LINER(0.0, -HEIGHT, 1)
      CALL LINER(-WIDTH, 0.0, 1)
C
      CALL ENDPIC
      CALL RLSDEV(IDEV)
      END

```

**Figure 12.7    *Pixel array subarea (generated by the example above).***



### 12.3.2 Inquiring Pixel Values From the Display

A routine is available to inquire the contents of the display of a raster device. It works the opposite way of *PIXARR*, but in a simplified way. This function is often not available due to device hardware limitations.

The contents of a rectangular area of the display is read by

**CALL DATPIX (Xwlow, Ywlow, Idimx, Indarr(1,1),  
Nx, Ny, Istat)**

where **Xwlow** and **Ywlow** is the lower left corner of the rectangle in window coordinates. If the position is known in NDC or user coordinates, it may be converted to window coordinates using the routines described on **page 8-14**.

**Nx** and **Ny** are the number of pixel values, in X and Y direction, to return. The pixel values, i.e. colour indices, are returned through **Indarr**. **Idimx** is the first dimension of this array *as defined*, allowing parts of **Indarr** to be read.

**Istat** is a status variable returning 0 if all pixels were returned OK, -1 if the function was not available with the device in use. If the arguments are given so that pixels outside the display surface are referenced, a value of 1 is returned through **Istat**. In that case, pixel readback is aborted.





# Chapter 13

## Picture Element Attributes

---

The appearance of all picture elements (graphic primitives) depends on a number of attributes. Some of these are specified with the drawing routines, others are set by global routines. The most commonly used of the latter kind, the colour index, was described in **Chapter 11**.

Picture elements also have some other attributes. As with the colour index, these are set to a current value, which is applied to all primitives subsequently generated. All these attributes are sent to the current device driver, and interpreted by this. Whether the attributes actually give any visual effect will then be device dependent. Thus, the routines described in this chapter should be avoided if device independency is a major goal.

There is an additional attribute, detectability, that is not described here. As this is used only in connection with pick input, it is described in **Chapter 20** with the other aspects of pick input.

A common property of the global picture element attributes, is that they are all reset to their default values when a new picture segment is opened by *BGNPIC* (see **page 3-1**). The current value of the attributes may be requested by the *DATATR* routine described on **page 23-2**.

### 13.1 Linewidth

Different colours are the preferred attribute for distinguishing between different parts of a plot. For monochrome devices, different linewidths may be used instead.

In **Chapter 9**, routines for defining different linewidths by drawing a number of parallel lines were described. Although device independent and very flexible, the method will reduce drawing speed as the amount of data to transfer to the device increases.

As an alternative when using raster devices, different linewidths may be achieved by

**CALL LINWID (Wscal)**

where **Wscal** is a scaling factor, in the range 0.0 to 25.5, to be applied to the default linewidth. With most devices, the default linewidth is 1 pixel wide, i.e. the fractional part of **Wscal** is ignored, and it is not possible to specify lines thinner than the default.

Note that the specified linewidth will be applied to all lines, including software text and circles. Whether the linewidth is applied to hardware generated primitives will be device dependent.

The linewidth scaling factor is reset to its default value (1.0) when *BGNPIC* is called.

## 13.2 Depth Modulation

Depth modulation is a hardware feature provided by 3 dimensional devices for visualizing the Z coordinate, often by adjusting the intensity depending on the depth.

This feature is controlled by

CALL DEPCTL (lswtch)

where **lswtch**=1 enables, and **lswtch**=0 disables depth modulation. The default value is 0 (disabled).

The modulation range is specified by the Z range of 3 dimensional viewports (see **page 2-3**). That is, if intensity modulation is available, primitives at the lower Z limit will be drawn with lowest intensity, primitives at the upper limit will be drawn with highest intensity.

## 13.3 Blinking

An additional hardware feature that is available with some devices, is the ability to apply blinking to primitives. This is controlled by

CALL BLICTL (lswtch)

where **lswtch**=1 enables, and **lswtch**=0 disables picture element blinking. The default value is 0.

# Chapter 14

## Picture Segment Storing

---

The most common use of GPGS-F is to present results of some computations or data collections in a quite straightforward way. If the data is changed within the application, the new data is presented after clearing the display surface, using *CLRDEV* (page 1-2), i.e. the complete picture is redrawn. If the changes affect only parts of the picture, this means that a lot of unnecessary redrawing is performed.

In addition to its drawing and interaction facilities described in previous chapters, GPGS-F does however provide methods for storing and reusing pictures or parts of pictures, allowing dynamic picture building and manipulation.

The graphic unit used in storage and manipulation operations is the **picture segment**. As described in **Chapter 3**, a picture segment is a collection of graphic primitives and attributes. There is no limit to the size of a picture segment. With some applications, there may be a large number of segments, each containing very few primitives, with other applications there may be just one large segment containing the complete picture.

It is not possible to change the contents of a picture segment after the definition is completed, i.e. after *ENDPIC* is called.

### 14.1 Segment Classes

There are two classes of stored segments, called **Pseudo Segments** and **Retained Segments**. These are stored in different ways, and they are used for quite different purposes. An introductory description of the two classes, and the routines for defining segment storage, are given in the following sections, details on how to use stored segments are given by following chapters.

#### 14.1.1 Pseudo Picture Segments

Pseudo segments are used for modelling purposes. The primitives are stored as transformed user coordinates in a device independent format. The segments may later be inserted when drawing on a terminal or plotter. When inserted, the primitives are transformed once more, by the current transformation matrix.

Pseudo segments may be stored in buffers (arrays) supplied by the user, or in permanent picture libraries (disc files). When using the second kind of storage method, segments may be reused not only by the application creating the segments, but by other applications as well.

### 14.1.2 Retained Picture Segments

Retained segments are used for dynamic picture manipulation, and for simulating pick input. Such segments may be used with terminals only, and are device dependant in the sense that they belong to the device they have been created for.

With some terminals retained segments are stored in the terminal itself, giving fast update of picture changes. For compatibility, GPGS-F provides a module for storing segments for terminals that does not have local display files. This means that the same visual effect may be achieved on all terminals, the difference being the speed with which the visual changes are performed.

Retained segments may be manipulated by turning the visibility on/off, transformed using image transformations, deleted or copied to a background device.

## 14.2 Picture Segment Identifiers

Picture segments that are to be stored must be given a unique identifier (supplied with the *BGNPIC* routine). This identifier is stored in a Fortran integer, together with a 4-bit status word. Thus, if a Fortran integer is 16 bits, the range is 1 to 4095, if a Fortran integer is 32 bits, the range is 1 to 268435455.

Each device driver does however keep a separate list of segments belonging to it (pseudo segments 'belong' to device 0, the pseudo device). Thus, the same identifier may be used for segments belonging to different drivers.

If an already used segment identifier is given, an error message is given, and the old segment replaced by the new one.

## 14.3 Defining Picture Storage

Picture segments may be stored in **primary buffers** or in **picture libraries**. Pseudo segments always require storage to be defined, either in buffer or library.

Retained segments are by some devices allowed to be stored in the device itself, in which case no storage need to be allocated by the application program. There is however one exception. If the segments are to be copied to a background device (described in **Chapter 19**), the segments *must* be stored in GPGS-F buffers.

If the device is not capable of storing segments by itself, *simulated* retained segments are stored in GPGS-F buffers. Such segments may later be saved in library files, but all segments that are visible on the screen must reside in buffers.

Whether a device is capable of storing retained segments in hardware is shown in the driver descriptions in **Appendix E**, and may also be requested by using the *DATDEV* routine described on **page 23-4**.

A program example using the different storage methods is given at the end of the chapter.

### 14.3.1 Primary Buffers

A primary buffer is a Fortran integer array or equivalent, defined as a GPGS-F buffer by

```
CALL NITBUF (IARR(1), Length)
```

where **IARR** is the name of the array. **Length** is the number of integers in the array, and must be less or equal to the size of the array as declared.

An array defined as a GPGS-F buffer must not be used for other purposes by the application program, and must reside in permanent storage. With Fortran, this is ensured by using the `SAVE` statement, placing the array in a `COMMON` area, or declaring the array in the main program.

#### **Example 14.1** *Defining a GPGS-F buffer.*

```
C
      PARAMETER (LENGTH=2000)
      INTEGER IARR(LENGTH)
      SAVE IARR
C
      CALL NITBUF (IARR, LENGTH)
```

Several primary buffers may be defined by a single application. GPGS-F keeps a list of the buffers, where new buffers are put at the head when defined. When a new segment is created, it will be stored in the buffer at the head of this list, i.e. the *current* buffer.

An already defined buffer may be moved to the head of the buffer list by

```
CALL SELBUF (IARR(1))
```

where **IARR** is the array defined by *NITBUF*. New segments will then be stored in this buffer, other effects are discussed in **section 14.3.1.1**

*SELBUF* is also used to switch back to primary buffer storage in cases where both primary buffers and picture libraries are used (see the description of *SELLIB*, **page 14-5**).

If a buffer is no longer needed by GPGS-F, it may be released by

```
CALL RLSEBUF (IARR(1))
```

All picture segments in the buffer are then deleted. If the buffer contains retained segments, these will also be deleted from the screen. No error will occur if buffers are not released at program termination.

If the current buffer is released, the next one in the buffer list is set to current.

### 14.3.1.1 Programming Guidelines

Selecting the size of a buffer is not easy. The size will of course depend on how many segments are to be stored, and the number of graphic primitives in each segment. The problem is that GPGS-F does not provide any method for finding how much space is needed for the different primitives and attribute settings.

The *DATBUF* routine described on **page 23-8** may be used to find the amount of free and used space in a buffer.

A segment is stored in a continuous part of the buffer, starting at the 'top'. Thus, free space is always at the end of the buffer. When a segment is deleted, its space is marked as released, and all following segments are moved forward to compact the buffer when a new segment is created. With a large buffer containing many segments, this may be quite time-consuming. Using several buffers will reduce the time needed, as each buffer will then contain fewer segments.

Using several buffers also has other effects. Whenever a segment operation is to be performed, the given segment is searched for in all buffers in the sequence given by the buffer list. *SELBUF* may be used to optimize this searching, but requires that the application keeps track of which segments are stored in which buffer to have any effect.

When retained segments are redrawn (by using the *REDRAW* routine described on **page 16-4**), segments with the same priority are redrawn in the sequence given by the buffer list. The same applies to pick input (described in **Chapter 20**), the buffer list determines the sequence to be used by GPGS-F when scanning for hit.

Segment priority is described on **page 17-3**.

### 14.3.2 Picture Libraries

A major disadvantage of using primary buffers for segment storage is that the size of the buffers may not be changed during program execution. In many cases it is not known how many segments are to be stored. If the size is too small, the only way to change this is to first change the application program, and then recompile and reload the program.

A second major disadvantage is that the segments stored in primary buffers are local to the application in which they are created. There is no way to use the same segments by other applications.

To help solve these problems, GPGS-F provides routines for storing picture segments on disc files. This kind of segment storage is called picture libraries. When using picture libraries, there is no limit to the size of the storage, as disc files normally are dynamically extendable. Segments stored in picture libraries will remain in the file on program termination, so that other applications may use the same segments.

Pseudo picture segments may be stored directly in picture libraries, and copied between picture libraries and primary buffers (see **page 14-6**).

Retained segments may be copied to and from picture libraries, but can be displayed only when stored in a buffer. Obviously, if retained segments are stored in the device itself, they may not be copied.

Files to be used as GPGS-F picture libraries must be opened by the application program. As the Fortran `OPEN` statement will be different for different computers, GPGS-F provides utility routines for opening and closing such files.

A file is opened by

**CALL GUF SOP (lunit, Fname, Nrec, Fstat, Istat)**

where **lunit** is the Fortran file number to use, in the range 1 to 99. **Fname** is the filename (text string), **Nrec** is the maximum number of records to be stored in the file (this is ignored by computers that are able to dynamically extend the file size) and **Fstat** is the file status (text string, one of 'new', 'old', 'unknown'). **Istat** is the returned status from the Fortran `OPEN` statement, 0 (zero) if OK.

Once a file is opened, it may be defined as a picture library by

**CALL NITLIB (lunit)**

where **lunit** is the Fortran file number, which must be the same number as used with *GUF SOP*. *NITLIB* will not change to contents of the file, i.e. it may already contain picture segments created by another application, or it may be an empty file to be used for storing new segments.

A picture library is cleared, i.e. all picture segments within the file is deleted, by

**CALL CLRLIB (lunit)**

where **lunit** is the Fortran file number.

Up to 4 files may be used as picture libraries at the same time. The last one defined will then be the *current* library. An already open library may be set to be the current one by

**CALL SELLIB (lunit)**

New segments will be stored in the current library. However, if a buffer is selected after *SELLIB*, new segments will be stored in the selected buffer. Thus, even if only one buffer and one picture library is used, *SELBUF* and *SELLIB* may still have to be used to specify where segments are to be stored. The same is true when segments are to be deleted, by using the *DELPIC* routine described on **page 14-7**.

Other segment operations are executed by separate routines for referring to segments in buffers and libraries, or the operations are available with primary buffers only.

When referring to segments, there is an important difference between primary buffers and picture libraries. As described previously in this chapter, when referring to a segment stored in a buffer, *all* buffers are searched until the segment is found. When a segment in a library is referred, only the *current* library is searched.

This last fact implies an additional difference between buffers and picture libraries. The picture segment identifier must be unique with respect to all primary buffers, while with libraries, the identifier must be unique within each library.

When a picture library is no longer to be used by an application, it *must* be released by

**CALL RLSLIB (lunit)**

If a picture library is not released, its contents may be inaccessible the next time it is opened. This is because a file header is copied to GPGS-F when the file is defined as a library by *NITLIB*. If some segments are deleted from, or added to the file, the copy of the header is changed, but the modified header is not copied back until *RLSLIB* is called.

If the current library is released, there will be no current library, even if there are other open libraries. A new current library must be explicitly set by *SELLIB*.

Finally, after releasing a library, the file must be closed by

**CALL GUFSC (lunit, lstat)**

where **lunit** still is the Fortran file number. **lstat** is the returned status from the Fortran `CLOSE` statement, 0 (zero) if OK.

## 14.4 Copying Segments

A picture segment may be copied from a primary buffer to a picture library by

**CALL SAVPIC (ldold, ldnew)**

where **ldold** is the segment identifier in the buffer, and **ldnew** is the identifier to be used in the library. The segment is copied to the current library, selected by the last *NITLIB* or *SELLIB*. The source segment need not be in the current buffer, as all buffers are searched.

Both pseudo and retained segments may be copied to a picture library.

Similarly, a segment may be copied from a library to a buffer by

**CALL RESPIC (ldold, ldnew)**

where **ldold** and **ldnew** is the identifier of the source and destination segment. The segment will be copied to the current buffer. Only the current library will be searched for the segment. If not found there, an error message is given, but other libraries will not be searched.



As mentioned earlier in this chapter, all segments belong to the device they were created for. When copying a segment from a library to a buffer, the device this segment belong to must have been initialized.

If the segment copied is a retained segment, it is marked as invisible, i.e. it will *not* be drawn on the device it belong to. To make it visible, the *VISPIC* routine described on **page 17-1** must be used.

## 14.5 Deleting Segments

A picture segment may be deleted from a buffer or picture library by

**CALL DELPIC (Ident)**

where **Ident** is the picture segment identifier.

The segment is searched for in *either* the current library or in the primary buffers, depending on whether *NITLIB / SELLIB* or *NITBUF / SELBUF* was last called.

As GPGS-F does not allow picture segments to be changed after they have been created, the only way to 'change' a segment is to delete it and create a new segment with the same identifier.

The effect of deleting retained segments is explained on **page 16-4**.

### **Example 14.2    *Picture segment storing.***

```

      .
      CALL NITBUF(IARR1,10000)
C   New segments are stored in buffer 'IARR1'
C
      .
      CALL NITBUF(IARR2,5000)
C   New segments are stored in 'IARR2'
C
      .
      CALL GUF SOP(IUNIT1, 'gpgs-01.dat', nrec, 'new', ISTAT)
      IF (ISTAT .NE. 0) CALL Error-Handler
      CALL NITLIB(IUNIT1)
C   New segments are stored in library 'IUNIT1'
C
      .
C
C   Copy a segment from 'IUNIT1' to 'IARR2'
C
      CALL RESPIC(ID1, ID2)
      CALL SELBUF(IARR1)
C   New segments are stored in 'IARR1'
C
      .
      CALL GUF SOP(IUNIT2, 'gpgs-02.dat', nrec, 'new', ISTAT)
      IF (ISTAT .NE. 0) CALL Error-Handler
      CALL NITLIB(IUNIT2)
C
C   New segments are stored in library 'IUNIT2'
C
      .
C
C   Copy a segment from buffer IARR1 or IARR2
C   to library 'IUNIT2'
C
      CALL SAVPIC(ID3, ID4)
C
C   Copy a segment from buffer IARR1 or IARR2
C   to library 'IUNIT1'
C
      CALL SELLIB(IUNIT1)
      CALL SAVPIC(ID5, ID6)
C
C   Delete a segment from library 'IUNIT1'.
      CALL DELPIC(ID1)
C
C   Delete a segment from one of the buffers
      CALL SELBUF(IARR1)
      CALL DELPIC(ID3)
C
C   Release buffers and libraries
      CALL RLSBUF(IARR1)
      CALL RLSBUF(IARR2)
      CALL RLSLIB(IUNIT1)
      CALL GUF SCL(IUNIT1, ISTAT)
      CALL RLSLIB(IUNIT2)
      CALL GUF SCL(IUNIT2, ISTAT)

```

# Chapter 15

## Pseudo Picture Segments

---

Pseudo picture segments contain graphic and non graphic (attribute) elements stored in a device independent form. The segments are created and stored by the **PSEUDO** driver, which has GPGS-F device number 0 (zero).

Although this driver is special, the application program controls the driver as other drivers, using the device control routines described in **Chapter 1**, i.e. *NITDEV* is used to initiate, and *SELDEV* to set it to the current device.

*RLSDEV* is used to release the driver, but has the additional effect of deleting all pseudo segments in primary buffers. Thus, even if the application first creates all pseudo segments needed, then switches to a second device that are to use those segments, the pseudo driver must not be released until the pseudo segments are not to be used any more.

*CLRDEV* will normally not be used with the pseudo driver, as there is no display surface to clear. If called, the effect will be the same as with *RLSDEV*, i.e. all pseudo segments are deleted from primary buffers.

### 15.1 Inserting Pseudo Segments

When the pseudo driver is active, picture segments are stored in the current buffer or library, depending on which was last selected. Coordinates are stored after being transformed by the current transformation matrix, but are not passed through the window-to-viewport mapping.

At a later time the *contents* of these picture segments may be inserted into new segments for any device. Inserted coordinates are then transformed once more, by the current transformation at insertion time.

If a pseudo segment is stored on a primary buffer, it is inserted into the currently open segment by

**CALL INSERT (Ident)**

where **Ident** is the pseudo segment identifier. If several primary buffers are in use, all buffers are searched, in the sequence given by the buffer list (see the description of *NITBUF* and *SELBUF* on page 14-3).

If a pseudo segment is stored in a picture library, it is inserted by

**CALL INSLIB (lunit, Ident)**

where **lunit** is the unit number of the library and **Ident** is the pseudo segment identifier. Other libraries are not searched if the segment is not found in the library selected by **lunit**.

As a pseudo segment is inserted into the open segment, some attribute setting functions are also inserted. These will however not affect the settings of the open segment. Attribute settings in the open segment will in the same way not affect the attributes of primitives from the inserted segment.

As might be expected, inserting a segment from a buffer is faster than inserting from a file. If the same segment is to be inserted more than once by an application, some time will be saved by first copying the segment from file to buffer, using the *RESPIC* routine described on **page 14-6**, and then inserting the segment from buffer each time it is needed.

### 15.1.1 Colour of Inserted Primitives

When *INSERT* or *INSLIB* is called, the current colour index (see **page 11-2**) will be set to its default value of one. This will be used until a colour index command is found in the inserted segment, i.e. all primitives will be drawn with the colour that was specified when the pseudo segment was stored.

As pseudo segments are often used to define (small) building blocks that are later to be put together, there will be cases where it is desirable to specify the colour at the time of insertion.

This is possible by

**CALL INSCOL (lmod)**

where **lmod**=1 means that the colour index will *not* be set to default when *INSERT* / *INSLIB* is called. The effect of this, is that the *current* colour index will be used for all primitives inserted, until a colour index command is inserted. From then on, colours will appear as if *INSCOL* was not used.

If *COTIND* was not used at all when a pseudo segment was created, the complete segment will be drawn using the colour index that is current at insertion time.

If **lmod** is set to 0, the default condition is reset.

**Example 15.1    *Basic use of pseudo picture segments.***

```
INTEGER IARR(1000)
.
.
CALL GPGS
CALL NITBUF(IARR,1000)
C
C Create 2 pseudo picture segments in buffer,
C one red, one with no colour setting.
C
CALL NITDEV(0)
CALL BGNPIC(1)
CALL COTIND(2)
.
CALL ENDPIC
CALL BGNPIC(2)
.
CALL ENDPIC
C
C Select a colour device, open a picture segment and
C draw some primitives using colour index 3 (green).
C
CALL NITDEV(72)
CALL BGNPIC(7)
CALL COTIND(3)
.
C
C Insert both pseudo segments. Segment 1 will be red,
C segment 2 black (default)
C
CALL INSERT(1)
CALL INSERT(2)
CALL ENDPIC
C
C Apply some transformations, open a new segment and
C draw some primitives using colour index 4 (blue)
C
CALL TRANS(...)
CALL BGNPIC(8)
CALL COTIND(4)
.
C
C Insert both pseudo segments once more,
C after calling INSCOL.
C
CALL INSCOL(1)
CALL INSERT(1)
CALL INSERT(2)
C
C Segment 1 will still be red, as defined when created,
C while segment 2 will be blue.
C
```

### 15.1.2 Areas of Application

Pseudo segments may be used for a wide range of different purposes. Perhaps the most common one is to store frequently used subpictures in picture libraries. Other applications may then use these segments to build more complex pictures, without having to create the subpictures each time.

An additional advantage of using this method, is that if a subpicture is to be changed, only the application *creating* the pseudo segments has to be changed. The applications *using* the segments may be left unchanged, and need not even be recompiled.

Using pseudo segments is also useful when viewing complex 3D models. Quite often, defining the transformations that give the best view of such models is not straightforward. If the model is stored as a pseudo segment, experimenting with different transformations does not require regeneration of the model each time. The same is true when different projections of 3D models are to be drawn.

## 15.2 Clipping

The coordinates of pseudo picture segments are stored as transformed user coordinates. If clipping is enabled (see **page 2-4**) during generation, coordinates are clipped before being stored.

If clipping is enabled when the pseudo segment is later inserted, it will be clipped once more, this time by the current window setting at insertion time.

Enabling clipping when creating pseudo segments is not very common. Using clipping at insertion time, however, will often be most useful. By using different window and viewport settings (or scaling), details of inserted segments may be viewed, i.e. a zooming effect is achieved. 3D clipping may in addition be used to view 'slices' of 3D models at different Z levels.

## 15.3 Pseudo Segment Reference

As described on **page 15-1**, *INSERT* and *INSLIB* will insert the contents of a pseudo segment into the currently open segment. This is allowed even if the open segment is also a pseudo segment. This may be used to create pseudo segments at different complexity levels, first some basic building blocks, then compound segments containing these basic segments.

However, this means that the code of the basic segments is duplicated. An additional drawback is that if a basic segment is changed (deleted and recreated), all compound segments using that segment must also be recreated to reflect the changes.

To avoid these disadvantages, GPGS-F allows pseudo segments to contain symbolic references to other pseudo segments. Such a reference is inserted into the currently open segment by

**CALL REFER (Ident)**

where **Ident** is the pseudo segment referenced. GPGS-F does *not* require the referenced segment to exist at the time the reference is inserted, i.e. the segments may be created in any sequence.

When inserting a segment containing a reference from a primary buffer, the referenced segment must also reside in a primary buffer, but not necessarily the same.

When inserting segments from a picture library, referenced segments must reside in the same library as the segment containing the reference.

Symbolic references may be nested to a maximum of 10 levels. The number of references at each level is however not limited.

When *REFER* is called, the current transformation matrix is inserted into the open segment in addition to the reference. The (visual) effect of this will be the same as if the referenced segment was inserted instead of referenced.

The mode set by *INSCOL* (**page 15-2**) will have the same effect for referenced segments as inserted segments. That is, if *INSCOL(1)* is called, primitives generated ahead of the first colour setting command will be drawn using the colour index that is current at the time the reference is encountered.

### **Example 15.2**    *Creating compound pseudo segments.*

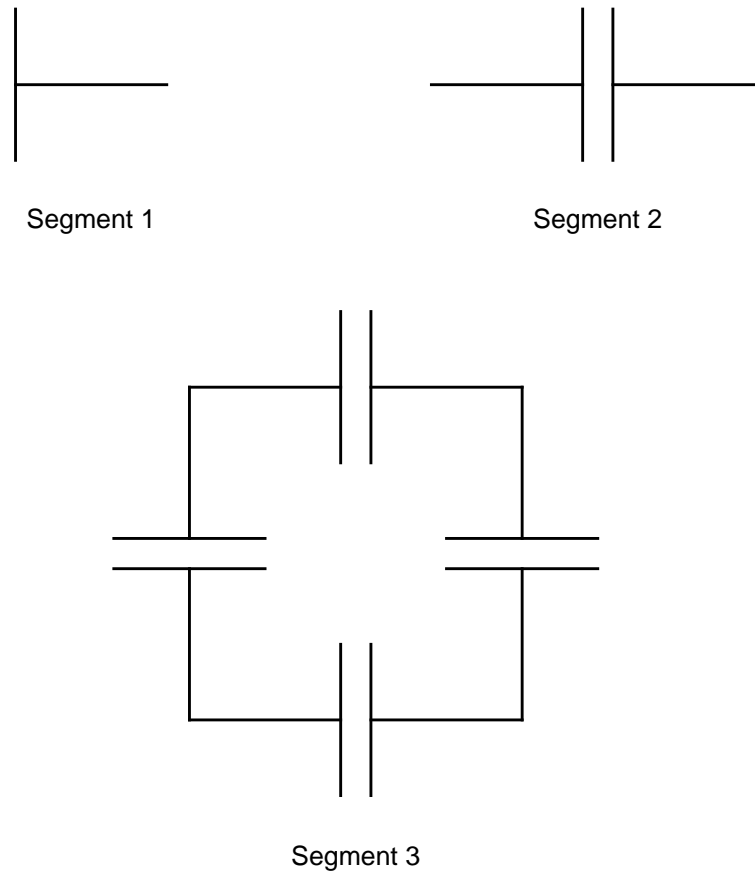
```

      INTEGER Ibuff(1000)
      .
C
C   Define primary buffer for segment storage
C
      CALL GPGS
      CALL NITBUF(IBUFF,1000)
C
C   Initialize pseudo driver.
C
      CALL NITDEV(0)
C
C   Create a basic picture part.
C
      CALL BGNPIC(1)
      CALL LINE(2.0, 0.0, 0)
      CALL LINE(0.0, 0.0, 1)
      CALL LINE(0.0, 1.0,0)
      CALL LINE(0.0,-1.0,1)
      CALL ENDPIC
C
C   Create a compound segment containing two basic parts.
C
      CALL BGNPIC(2)
      CALL XLAT(0.2,0.0)
      CALL REFER(1)
      CALL IDEN
      CALL XLAT(-0.2,0.0)
      CALL ROTAD(180.0,3)
      CALL REFER(1)
      CALL ENDPIC
      CALL IDEN
C
C   Create a third segment, containing four instances of
C   segment 2 put together.
C
      CALL BGNPIC(3)
      CALL XLAT(0.0, 2.2)
      CALL REFER(2)
      CALL XLAT(0.0,-4.4)
      CALL REFER(2)
      CALL IDEN
      CALL ROTAD(90.0,3)
      CALL XLAT(0.0, 2.2)
      CALL REFER(2)
      CALL XLAT(0.0,-4.4)
      CALL REFER(2)
      CALL IDEN
      CALL ENDPIC

```



**Figure 15.1** *Pseudo segments (defined by Example 15.2).*





# Chapter 16

## Retained Picture Segments

---

Graphic information generated by GPGS-F is passed to the active device driver. Some drivers will store the image in internal device display files, while most devices will just draw the image on the display surface and discard the picture code.

Interactive programming with devices without picture storage is not without problems. In order to make minor changes to the screen image, the screen has to be cleared, and the complete changed image must be regenerated.

By introducing retained segments in this kind of programs, picture changes will be much easier to handle.

Specifying that a picture segment is to be retained means that the contents of the segment should be stored for later use. Some devices store the segments in device hardware, while others require storage in GPGS-F buffers. In the last case, *NITBUF* (see **page 14-3**) must be used to define these buffers.

**Appendix E** shows what drivers need GPGS-F buffers, and the *DATDEV* routine described on **page 23-4** may be used to get the same information from an application program. If storage in GPGS-F buffers is required, all retained segment operations are handled by a module called **GPGS-F buffer/pick simulation module**, or **SIMU driver** (described on **page E-4**). If a device provides hardware storage, all operations on retained segments are handled by the hardware or device driver.

### 16.1 Storage Mode

Whether picture segments will be retained or not, is controlled by a global storage mode set by

**CALL   RETAIN (lswtch)**

**lswtch** set to 1 will turn storage mode on, i.e. picture segments will be retained. **lswtch** set to 0 will turn storage mode off. By default the storage mode is 0 (off).

As mentioned, a major reason for using retained segments is that it makes it easy for the programmer to make changes in a picture on the screen. Such changes include setting visibility of segments (**Chapter 17**) and moving segments on the screen (**Chapter 18**).

The action performed when a retained segment is changed, depends on the terminal type and how the segment is stored.

If retained segments are stored in device hardware, the visual change is controlled by the hardware. With raster terminals using GPGS-F buffers for storage, the visual change is controlled by the GPGS-F **SIMU** driver (described on **page E-4**). Deleting a segment or making it invisible is done by redrawing the segment using the background colour (selective erase). This will give the unwanted effect that 'holes' will appear in other segments if primitives overlap.

With devices not capable of performing selective erase, the complete image must be redrawn each time a segment is made invisible or deleted. Whether this redrawing is done automatically by GPGS-F is controlled by the deferral mode described next.

## 16.2 Deferral Mode

In cases where several segments are to be changed, it may be faster to defer the changes until all segments are changed, and then redraw the complete image, than to perform each change in sequence. The reason for this is that the time needed to make a segment invisible will be the same as needed to draw the segment.

Deferral of picture changes is controlled by

**CALL DEFER (ldefer, laldev)**

**ldefer** selects when the display is to be updated:

- 0: Update the display as soon as possible.
- 1: Update before next interaction. (i.e. when a request input routine or *AWAIT* is called)
- 2: Do not update until requested by user.

Deferral mode is said to be *set* when **ldefer** is 1 or 2. By default deferral mode is not set.

The action necessary, and time needed, to keep the screen updated is quite different depending on whether the terminal is capable of performing selective erase or not.

If deferral mode is set, **laldev** is used to select which devices this applies to.

- 0: Defer only when for devices without selective erase.
- 1: Defer for all devices.

When deferral mode is set, GPGS-F will do some optimization to avoid unnecessary redrawing. When the visibility of a segment is switched on, or a new segment is created, and nothing is yet deferred, the segment will be drawn at once. If a segment is made invisible, the action will be deferred, and so are all subsequent changes.

When picture changes have been deferred, and the screen is to be updated (by user request or internal GPGS-F action), this is always done by first clearing the screen and then redrawing all segments currently set visible. This means that if there are segments on the screen that are *not* retained, these will disappear.

If the user wants the display to be updated without changing the deferral mode, this is done by

**CALL UPDAT (Iregen)**

**Iregen** set to 1 means that the picture should be redrawn if there are deferred changes. **Iregen** set to 0 will *not* redraw the picture, just ensure that the internal driver buffer is emptied. In both cases the device will be set in non graphic mode. There is no need to use *UPDAT(0)* if there is no picture segment open, as *ENDPIC* will already have performed the same actions.

Calling *UPDAT(1)* when deferral mode is set, has the same effect as:

```
CALL DEFER(0, 0)
CALL DEFER(Previous-Idefer, Previous-Ialdev)
```

### Example 16.1 Using *DEFER* and *UPDAT*.

CALL DEFER (1, 0)	Set deferral mode for devices with no selective erase.
make-seg-1 visible	Segment always made visible as nothing is yet deferred.
delete-seg-2	Deleted if selective erase possible. If not, deferred.
make-seg-3 visible	Made visible if selective erase possible. If not, deferred.
CALL UPDAT (1)	Picture redrawn if selective erase not available. If selective erase available, display is up to date, i.e. no action necessary.

On page 17-4 there is an example of a complete program using *DEFER* and *UPDAT*.

## 16.2.1 Compatibility Routines

In previous versions of GPGS-F, deferring picture changes was done by collecting the changes in batches. To provide backwards compatibility these routines are still available.

A batch of updates starts with

**CALL BGNBTC**

and ends with

**CALL ENDBTC**

Calling *BGNBTC* has the same effect as calling *DEFER(2,1)*, while *ENDBTC* equals *DEFER(0,0)*.

When writing new programs, *DEFER / UPDAT / REDRAW* should be used instead of these compatibility routines.

If updating old programs, either continue using *BGNBTC / ENDBTC* throughout the program, or replace *all* occurrences of *BGNBTC / ENDBTC* with the new routines.

## 16.3 Redrawing

When doing changes on a terminal using selective erase, there will be 'holes' in the picture if deleted segments overlap other segments. This is especially noticeable if deleted segments contain polygons.

To 'repair' these holes, all visible segments may be redrawn by

**CALL REDRAW**

The difference between using *REDRAW* and *UPDAT(1)* is that *UPDAT(1)* will redraw the picture only if there are deferred actions, while *REDRAW* will redraw the picture in any case.

Redrawing is done by first clearing the display, then all visible segments are redrawn in the sequence given by the segment priority (set by the *PRIPIC* routine described on **page 17-3**). Thus, if there are any segments that are *not* retained, these will disappear when calling *REDRAW*.

When using a multi window device (**Chapter 21**), there is a separate routine available for redrawing the segments belonging to a given window.

## 16.4 Deleting Segments

As described in **Chapter 14**, a segment may be deleted by

**CALL DELPIC (Ident)**

where **Ident** is the picture segment identifier. If this is a retained segment, it is first made invisible (if currently visible), and then deleted from internal hardware storage or GPGS-F buffer. Only segments belonging to the current active device may be deleted, i.e. to delete segments from another device, *SELDEV* (**page 1-2**) must be called to set the device to current before calling *DELPIC*.

Calling *CLRDEV* will, in addition to clearing the display surface, delete all retained segments belonging to the device given.

When a picture buffer is released by *RLSBUF* (**page 14-3**), any retained segments in that buffer is made invisible before being deleted. This applies to all devices currently in use, not only the current active device.

# Chapter 17

## Retained Segment Attributes

---

Each retained segment is stored with four different attributes. These are visibility, highlighting status, priority and detectability. The last one is used in connection with pick input and is described in **Chapter 20**. The other attributes are described in the following subsections.

All routines controlling segment attributes apply to the current active device. If multiple devices are in use, the actual device must be set to current, by using the *SELDEV* routine described on **page 1-2**, before changing attributes of segments belonging to it.

As stated in **Chapter 16**, retained segments are handled either by device hardware or by the **SIMU** driver (described on **page E-4**). In the last case, the segments are often referred to as 'software simulated'.

### 17.1 Visibility

The visibility of a retained picture segment is set by

**CALL VISPIC (Ident, lsw)**

where **Ident** is the picture segment identifier, and **lsw** is the visibility (0=invisible, 1=visible). By default, new segments are visible.

A software simulated segment is made invisible by redrawing it using the background colour. Thus, if segments overlap, parts of other segments will be erased as well. When the segments are handled by device hardware, such unwanted effects will normally not occur.

'Destroyed' segments may be fixed by using the *REDRAW* routine described on **page 16-4**. An individual segment may also be redrawn, by calling *VISPIC* with **lsw** set to 1 even if the segment is already visible. This works because *GPGS-F* *does not* check the current visibility when a segment is set to be visible.

When non retained segments are created, the graphic primitives will be sent directly to the device driver and displayed immediately, with a possible delay due to driver buffering.

When creating a (software simulated) retained segment however, the graphic primitives are just stored by the **SIMU** driver, and sent to the current device driver when the segment is closed by *ENDPIC*. This default behaviour may be overruled by specifying the visibility of the segment while it is still open.

Setting an open segment visible has the effect that all primitives generated so far will be displayed, and subsequent primitives will be displayed as soon as possible. This is useful if a retained segment is to be interactively generated.

Setting an open segment invisible will have no visual effect when *VISPIC* is called, but when the segment is closed, it will *not* be displayed.

### Example 17.1 *Visibility of new segments.*

Program code	Comments
CALL RETAIN (0) CALL BGNPIC (1) . CALL LINE (X, Y, IVIS) . CALL ENDPIC	Segment not to be retained  Displayed as soon as driver buffer is full.  Empties driver buffer
CALL RETAIN (1) CALL BGNPIC (2) . CALL LINE (X, Y, IVIS) . CALL ENDPIC	Segment to be retained  Not displayed yet  Complete segment is displayed
CALL RETAIN (1) CALL BGNPIC (3) . CALL LINE (X, Y, IVIS) CALL VISPIC (3, 1) . CALL LINE (X, Y, IVIS) . CALL ENDPIC	Segment to be retained  Not displayed yet All primitives generated so far is displayed  Displayed as soon as driver buffer is full  Empties driver buffer
CALL RETAIN (1) CALL BGNPIC (4) . CALL LINE (X, Y, IVIS) CALL VISPIC (4, 0) . CALL ENDPIC	Segment to be retained  Not displayed yet No <b>visual</b> effect  Segment is not displayed

Note that the description above applies to software simulated segments only. When segments are stored by device hardware, the effect is device dependent (some devices do not allow visibility setting of an open segment).



## 17.2 Highlighting

A second attribute of retained segments is the highlighting or blinking mode. This is controlled by

**CALL BLIPIC (Ident, lsw)**

where **Ident** is the segment identifier, and **lsw** is the highlighting/blinking mode (0=off, 1=on).

Note that this feature is *not* software simulated, i.e. it is available only with devices storing segments in hardware. The method used for highlighting/blinking a segment is device dependent.

By default the highlighting/blinking mode is off.

## 17.3 Priority

As segments are displayed, new segments will be drawn 'on top of' previous segments. Making a segment visible will have the same effect.

In some cases this will not give the wanted result, especially if segments contain polygons. If, for example, there are two segments, one containing a solid polygon, and the other containing a text supposed to be written inside the polygon, the text will not be visible if the polygon is displayed last.

To allow the application to control the sequence by which segments are displayed, there is a segment priority attached to each segment. This is set by

**CALL PRIPIC (Ident, lpri)**

where **Ident** is the segment identifier, and **lpri** is the new priority of the segment. The lowest priority is 0 (zero), while the upper limit is device dependant, when segments are stored by device hardware. With software simulated segments, the upper limit is 32767.

Default priority for new segments is 0. Setting the priority of the current open segment will have no effect, it will still be drawn on top of previously drawn segments.

The priority is used when segments are redrawn using the *REDRAW* routine described on **page 16-4**. The segments will be drawn in the sequence given by the segment priority, starting with the lowest priority. If there are two or more segments with the same priority, the segment buffers are searched in the sequence given by the buffer list (see the description of the *NITBUF* and *SELBUF* routines on **page 14-3**). Within each buffer, the sequence is given by the segment identifiers, starting with the lowest one.

In addition to the redrawing sequence, the segment priority also determines the sequence to use when searching for pick input (described **Chapter 20**), and when copying segments to a possible background device (described in **Chapter 19**).

### Example 17.2 *Segment visibility.*

```

C *****
C COMPLETE WORKING EXAMPLE
C *****
      INTEGER IARR(1000)
      PRINT *, ' Give device number'
      READ *, IDEV
      CALL GPGS
      CALL NITDEV(IDEV)
      CALL NITBUF(IARR, 1000)

C
C Generate a non retained p.s.
      CALL BGNPIC(1)
      CALL LINE(0.1, 0.9, 0)
      CALL CHARC('PS.1 - not retained')
      CALL ENDPIC

C
C Generate three retained p.s.
      CALL RETAIN(1)
      CALL BGNPIC(2)
      CALL LINE(0.1, 0.7, 0)
      CALL CHARC('PS.2 - retained')
      CALL ENDPIC

C
      CALL BGNPIC(3)
      CALL LINE(0.1, 0.5, 0)
      CALL CHARC('PS.3 - retained')
      CALL ENDPIC

C
      CALL BGNPIC(4)
      CALL LINE(0.1, 0.3, 0)
      CALL CHARC('PS.4 - retained')
      CALL ENDPIC

C
C Set deferral mode and make p.s. 2 and 3 invisible.
C Screen is cleared at UPDAT and only p.s. 4 is redrawn.
      CALL DEFER(2, 1)
      CALL VISPIC(2, 0)
      CALL VISPIC(3, 0)
      CALL UPDAT(1)

C
C Turn deferral mode off and make p.s. 2 visible again.
      CALL DEFER(0, 0)
      CALL VISPIC(2, 1)

C
C Try to make p.s. 1 visible.
C Will give error message - p.s. 1 is not retained
      CALL VISPIC(1, 1)
      CALL RLSDEV(IDEV)
      END

```

The example above must be run with a device allowing retained segments. To see the effect of the different subroutine calls, an input routine may be called at various points.

# Chapter 18

## Image Transformations

---

As described in **Chapter 6**, GPGS-F provides a complete set of routines for modelling transformations. These are used to set up a transformation matrix to transform all user coordinates before displayed on the display surface. This is a powerful tool for building complex objects from simpler parts, possibly defined in their own local coordinate systems.

Modelling transformations may however not be used to transform a picture, or parts of a picture, after it is displayed. To do this, another class of transformations, called *image transformations*, are available. Image transformations apply to retained picture segments belonging to the current active device.

Coordinates used with this kind of transformations are given in Normalized Device Coordinates (see **page 2-2**). As opposed to modelling transformations, image transformations are *not* cumulative, i.e. they always apply to the segments as originally created.

With the current version of GPGS-F the only image transformation available is translation.

The image transformation routines may not be called if a picture segment is open.

A retained picture segment is translated by

**CALL VXLAT (Ident, Xdisp, Ydisp)**

or

**CALL VXLAT3 (Ident, Xdisp, Ydisp, Zdisp)**

where **Ident** is the picture segment identifier, and **(Xdisp, Ydisp [,Zdisp])** is the new position of the segment relative to its original position.

If **VXLAT3** is used with a 2D device, the **Zdisp** value will be ignored.

The image transformation matrix of a segment is reset to the identity matrix by

**CALL VIDEN (Ident)**

where **Ident** is the picture segment identifier.

As long as the only image transformation available is translation, **CALL VIDEN(Ident)** will give the same visual effect as **CALL VXLAT(Ident, 0.0, 0.0)**. If/when additional image transformation routines are added, this will not longer be true.

**Example 18.1    *Interactively moving a segment on screen.***

```
C  ****
C  COMPLETE WORKING PROGRAM
C  ****
      INTEGER IBUFF(1000)
C
C  Initialise driver and picture buffer
C
      READ *,IDEV
      CALL GPGS
      CALL NITDEV(IDEV)
      CALL NITBUF(IBUFF, 1000)
      CALL RETAIN(1)
C
C  Create a retained picture segment.
C
      IPIC=3
      CALL BGNPIC(IPIC)
      CALL LINE(0.5, 0.5, 0)
      CALL CHARC('Retained p.s.')
      CALL ENDPIC
C
      1000 CONTINUE
C
C  Wait for interrupt from locator device
C
      CALL REQLOC(201, XNDC, YNDC)
C
C  Use given position as new start position of string
C  until pointing far right.
C
      IF (XNDC.LE.1.0) THEN
          CALL VXLAT(IPIC,XNDC-0.5, YNDC-0.5)
          GOTO 1000
      ENDIF
C
      CALL RLSDEV(IDEV)
      END
```

# Chapter 19

## Background Device

---

As stated in **Chapter 1**, graphic output is created for only one device at a time. To make the same picture on two devices, the same sequence of GPGS-F calls must be executed for the two devices.

In cases where a picture is interactively built on a terminal screen, this means that it is not straightforward to get a copy of the picture on a second device, in most cases a plotter or printer.

In order to ease this, it is possible to tell GPGS-F that the picture build is to be kept for later copying to another device. This second device is called the **background device**, while the device on which the picture is build, is often referred to as the **source device**.

Any device may be used as a background device, while only devices able to handle retained segments may be used as a source device.

A device is selected as a background device by

**CALL BACDEV (ldev)**

where **ldev** is the GPGS-F device number. This device must have been initialized by *NITDEV* (see **page 1-2**).

Throughout an application program, several different background devices may be used, but only one at a time. The last one selected by *BACDEV* will be current.

The picture on the source device is copied by

**CALL BACDRW**

which will copy all *retained* segments (see **Chapter 16**) that are currently visible to the background device. The segments will be copied in the sequence given by the segment priority (set by *PRIPIC*, **page 17-3**), starting with the lowest priority.

Storage space for these segments *must* be defined by *NITBUF* (see **page 14-3**), even if the source device is able to store picture segments in hardware, i.e. the segments will be stored both in hardware and in the buffer. Thus, *BACDEV* must be called *before* generating any segments on the source device, to tell GPGS-F that the segments need to be stored.

When the source device stores segments in hardware, and the background device is no longer to be used, the additional buffer storing may be switched off by **BACDEV(-1)**. If retained segments are software simulated, this will have no effect, as the segments then are stored in buffers in any case.

If the application program initially does not know what device is to be used as background device, driver number 1 (see **Appendix E**) may be initialized and specified as the background device, just to ensure that segments are stored. Then, at some later point, the *actual* background device may be selected by just calling *BACDEV* once more.

*BACDRW* may be called several times to get different hardcopies as the picture on the source device is changed, or new segments are added. Each copy may be placed on a separate page, by clearing the background device between each call to *BACDRW*, or several copies may be placed on a single page by using the *BACVPT* routine described in **section 19.1**.

The background device may also be used for direct output, by just selecting this as the current active device using *SELDEV* (see **page 1-2**). When *BACDRW* is called, the source device must however be the active device.

## 19.1 Background Viewport

When making a copy on the background device, the default viewport of the source device will be mapped onto the default viewport of the background device.

The viewport of the background device may however be specified by

**CALL BACVPT (Bvarr(1) )**

where **Bvarr** is a viewport array specified as with *VPORT* (see **page 2-3**). This will define the mapping to use when copying a picture from the source device, but will have no influence when using the background device for direct output.

Specifying the background viewport is especially useful when the hardcopy is to have a given physical size. To find the values to use for **Bvarr**, the method used in the example on **page 5-2** may be used. *BACVPT* may also be used to position several hardcopies in a single page on the background device.

## 19.2 Limitations

All graphic primitives will be copied from the source to the background device. Hardware generated primitives may however appear quite different when copied. Circles may not be drawn at all, and text may appear with a different font, size and rotation than on the source device. Pixel related primitives, i.e. patterned polygons and pixel arrays, will for sure not be identical, as there are no two devices with the exact same resolution.

In general, hardware generated primitives should be avoided if the major goal is to get hardcopies that are as identical as possible to the source picture, or if several different background devices are to be used. If, on the other hand, the major goal is to get high quality hardcopies on a given background device, this is achieved by using primitives that not necessarily give the best picture quality on the source device, but are known to be give the best result on the selected background device.

**Example 19.1    *Using a background device.***

```
INTEGER Ibuff(5000)
REAL BVARR(4)

C
DATA BVARR/0.0, 0.5, 0.0, 0.5/
C
CALL GPGS
C
C Initialize the PostScript and X11 drivers.
CALL NITDEV(90)
CALL NITDEV(72)
C
C Specify that PostScript is to be used as background device.
CALL BACDEV(90)
C
C Specify storage space for segments, turn retain mode on.
CALL NITBUF(IBUFF, 5000)
CALL RETAIN(1)
C
C Draw some segments on the screen
C (current device as it was last initialized).
CALL Draw_Segments
C
C Make a PostScript copy.
CALL BACDRW
C
C Do some picture changes on the screen.
CALL Change_Picture
C
C Make a new hardcopy, in the lower left quadrant of a new page.
CALL CLRDEV(90, 0)
CALL BACVPT(BVARR)
CALL BACDRW
C
C Do more picture changes.
CALL Change_Picture
C
C New hardcopy, on same page as previous, lower right quadrant.
BVARR(1)=0.5
BVARR(2)=1.0
CALL BACVPT(BVARR)
CALL BACDRW
C
C Add a heading and frame to the last PostScript page.
CALL SELDEV(90)
CALL Heading_and_Frame
C
C Release devices and exit.
C
CALL RLSDEV(90)
CALL RLSDEV(72)
C
END
```





# Chapter 20

## Pick Input

---

As described in **Chapter 8**, the interaction tools supported by GPGS-F are divided into classes according to the information the tools return.

The most advanced input class is pick input. Pick input means that the user may use some kind of pointing device to identify graphic elements in the display. This kind of interaction was in the early days of computer graphics available by using a lightpen connected to a refresh terminal with a local display file. Today, the lightpen is simulated by a locator device, such as a graphic cursor, and the display file is replaced by segment storage, either in hardware or in GPGS-F primary buffers.

To be identified by pick input, a graphic element must meet the following requirements:

- It must be part of a *retained* picture segment (see **Chapter 16**). If the device in use requires segment storage in GPGS-F buffers, storage space must be defined by *NITBUF* (described on **page 14-3**).
- It must have one or more *names* (integer identifiers) attached to it.
- It must be *detectable*.
- The segment which the element is part of must be detectable.

(Detectability is also referred to as *lightpen sensitivity*, for historical reasons.)

### 20.1 Element Namestack

The pick input routines will return the *namestack* of the graphic element pointed at. The namestack is an array of identifiers, where the first element is the identifier of the picture segment. The rest of the array is a list of *names* attached to the element. As the length of the total namestack is returned, the number of names will then be one less than this length.

If no detectable graphic element was pointed at, the input routines will return 0 (zero) as namestack length. The length will never be returned as 1, as detectable elements *must* have at least one name.

A name is attached to a single graphic element by

**CALL NAME (Iname)**

where **Iname** is an integer in the range 1 to 4095. This name is attached to the element generated by the *next* GPGS-F routine called.

Element names need not be unique, i.e. the same name may be attached to several elements within the same segment, and to elements within different segments.

**Example 20.1** *Picture element naming.*

```
.
C  Assign name 1 to a line.
    CALL NAME(1)
    CALL LINE(X, Y, 1)
.
C  Assign name 2 to a polyline.
    CALL NAME(2)
    CALL TABL(XARR, YARR, 10, 1)
.
C  Assign name 3 to a string.
    CALL NAME(3)
    CALL CHARC('Named string')
```

Note that names will also be assigned to invisible lines, i.e. moves. Hence, the sequence

```
CALL NAME(1)
CALL LINE(X, Y, 0)
CALL CHARC('String')
```

will name the move, not the string, i.e. *NAME* must be called immediately ahead of the routine generating the graphic element to be named.

If several graphic elements are to have the same name, it is not necessary to set the name of each element. Instead a whole group of elements may be named by

**CALL BGNNAM (Iname)**

before generating the first element of the group, and

**CALL ENDNAM**

after generating the last element of the group.

*BGNNAM* / *ENDNAM* may be nested to give more than one name to elements, i.e. instead of a single name, a list of names (a namestack) is stored with the elements. The number of nested names is device dependent, and may be requested by the *DATDEV* routine described on **page 23-4**. When retained picture segments are stored in GPGS-F buffers, names may be nested to a maximum of 9 levels.

Each *BGNNAM* call will push the given name on the namestack, while *ENDNAM* will remove the last added name from the stack. The pick input routines return the namestack with the highest level name first, as shown by **Example 20.2**.

### Example 20.2    *Nested element names.*

Program code	Namestack
CALL BGNPIC(1001)	
.	
CALL LINE(X, Y, 1)	Empty
CALL BGNNAM(100)	
CALL LINE(X, Y, 1)	1001 100
CALL BGNNAM(10)	
CALL LINE(X, Y, 1)	1001 100 10
CALL NAME(1)	
CALL LINE(X, Y, 1)	1001 100 10 1
CALL LINE(X, Y, 1)	1001 100 10
CALL BGNNAM(2)	
CALL LINE(X, Y, 1)	1001 100 10 2
<b>xx</b> CALL LINE(X, Y, 1)	1001 100 10 2
CALL ENDNAM	
CALL LINE(X, Y, 1)	1001 100 10
CALL ENDNAM	
CALL BGNNAM(20)	
CALL LINE(X, Y, 1)	1001 100 20
CALL NAME(1)	
CALL LINE(X, Y, 1)	1001 100 20 1
CALL LINE(X, Y, 1)	1001 100 20
CALL ENDNAM	
CALL LINE(X, Y, 1)	1001 100
CALL ENDNAM	
CALL LINE(X, Y, 1)	Empty

## 20.2 Element Detectability

In addition to having a name, elements must be detectable to be identified by the pick input routines. Detectability is controlled by a global switch set by

**CALL LPSCTL (lsw)**

where **lsw**=1 means **on**, **lsw**=0 means **off**. The current value of this switch is stored with graphic elements when created.

By default, i.e. when a new segment is opened, the detectability switch is **on**. However, the value is *not* stored with elements created before *LPSCTL* is called the first time. When working with retained segments only, this is of no significance, but it may be used to achieve special effects when pseudo segments are inserted into retained segments, as described on [page 20-7](#).

The only case where *LPSCTL* is useful if pseudo segments are not involved, is if some undetectable elements are to be created within a sequence of detectable elements. If the line marked **xx** in **Example 20.2** should not be detectable, the easiest way to achieve this would be to call *LPSCTL(0)* ahead of the *LINE* call, and *LPSCTL(1)* after the call. The alternative would be to insert a number of *ENDNAM* calls to empty the namestack, and then a sequence of *BGNNAM* calls to restore the namestack afterwards.

## 20.3 Segment Detectability

As mentioned, the element namestack and detectability are stored with the graphic elements, and may thus not be changed after the elements have been created.

The application program will still be able to control, at runtime, which parts of a picture that are detectable, as detectability is also a *segment* attribute.

The segment detectability is set by

**CALL LPSPIC (Ident, lsw)**

where **Ident** is the picture segment identifier, and **lsw** is the detectability switch. A value of 0 means **off**, 1 means **on**. By default, the switch is off.

As stated on page **page 20-1**, *both* the element and segment detectability must be set for a graphic element to be identified by pick input.

The example below shows all the routines necessary to build a segment containing detectable graphic elements.

### **Example 20.3** *Creating detectable graphic elements.*

```

      INTEGER IBUFF(5000)
      .
      .
C   Define segment storage, set retained mode on.
      CALL NITBUF(IBUFF, 5000)
      CALL RETAIN(1)
C
C   Open a picture segment, and draw elements that are not
C   to be detectable
      CALL BGNPIC(1)
      CALL LINE(X, Y, 0)
      CALL CHARC('Not detectable')
      .
      .
C
C   Draw the elements that are to be detectable,
C   using the same name for all.
      CALL BGNNAM(100)
      CALL LINE(X, Y, 0)
      CALL CHARC('Detectable elements')
      .
      .
      CALL ENDNAM
C
C   Following elements will not be detectable.
      CALL ENDNAM
      .
      .
C
C   Close the segment and make it detectable.
      CALL ENDPIC
      CALL LPSPIC(1, 1)

```

## 20.4 Scanning for Hit

When the application program calls one of the pick input routines, *REQHIT* (page 8-3) or *SMPHIT* (page 8-4), GPGS-F will return the namestack of the element, or a namestack length of 0 if no detectable element was pointed at.

Pointing at an element means that at least a part of the element must be within the *hit rectangle*. This is defined by the position of the pick tool and the pick aperture size (see page 8-9). The size of the hit rectangle thus determines how accurate the user must be when pointing.

If segments are stored in hardware, the information is returned directly by the device driver. If segments are stored in GPGS-F buffers, the information is found by software, using the **SIMU driver** (described on page E-4). Although the application programmer does not *need* to know the details of this procedure, it may give some hints on how to speed up execution time, which should be a major goal for any interactive program.

The SIMU driver will receive the hit rectangle from the device driver. Then, all retained segments that are currently visible and detectable are found, and sorted in priority order (the priority is set by *PRIPIC* described on page 17-3), with the highest priority first. Segments with the same priority are ordered according to the buffer list (see *NITBUF* and *SELBUF* on page 14-3). Within each buffer, the segments are sorted by the segment identifier, with the lowest number first.

Once the ordered segment list is completed, the actual scanning starts. Taking one segment at a time, each graphic element is checked if it has a name and is detectable, and if so, the rules given below are used to find whether it was hit. The scanning is aborted as soon as a element is found, i.e. the SIMU driver will not attempt to find the element that is closest to the pick tool position.

The following rules are used to find whether an element is hit:

- A line is hit if it intersects the hit rectangle.
- Software characters / circles, and hollow / hatched polygons are represented as lines, and are checked as such.
- A hardware circle arc is hit if it passes through the hit rectangle.
- Hardware text is hit if the surrounding box intersects the hit rectangle. If the device driver is not able to return the correct box, the box surrounding the given text if it was drawn using software font 0 is used.
- A marker is hit if its centre is within the hit rectangle.
- A solid or patterned polygon, or a pixel array, is hit if it overlaps the hit rectangle.

Following from the description above, there are several aspects that influence the time needed to detect a hit. The most important is the number of detectable segments, and their priority. How the graphic elements are grouped into segments will however also be essential. As all elements has to be scanned, undetectable elements should, if possible, not be mixed with detectable elements. To further improve response time, software text should not be used as detectable elements, such as menu items (**Example 20.4** shows how a text menu may be build using undetectable text strings).

### Example 20.4 *Menu building.*

```

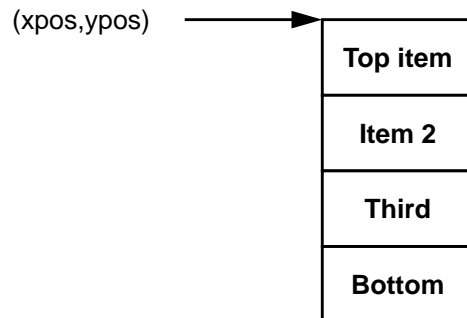
SUBROUTINE MENU(XPOS, YPOS)
C
  INTEGER IHIT(2), IMENL(4)
  REAL XPOL(4), YPOL(4)
  CHARACTER*8 CMENU(4)
  DATA XPOL/0.0, 0.1, 0.1, 0.0/
  DATA YPOL/0.0, 0.0, 0.05, 0.05/
  DATA CMENU/'Top item','Item 2','Third','Bottom'/
  DATA IMENL/ 8, 6, 5, 6/
C
C Create a segment containing 4 named polygons.
C The polygons are drawn using the background colour, with
C the perimeter drawn using the default colour.
C The input arguments give the upper left corner of the menu.
  CALL XLAT(XPOS, YPOS-YPOL(4))
  CALL BGNTRN
  CALL BGNPIC(1)
  CALL COTIND(0)
  CALL PRCIND(1)
  DO 1000 INAME=1,4
    CALL NAME(INAME)
    CALL POLY(XPOL, YPOL, 4, 1)
    CALL XLAT(0.0, -YPOL(4))
1000 CONTINUE
  CALL ENDTRN
  CALL ENDPIC
C
C Add the menu text, without names, centred in each polygon.
  CALL BGNPIC(2)
  CALL CJUST(0.5, 0.25)
  DO 2000 IMENU=1,4
    CALL LINE(XPOL(2)/2.0, YPOL(4)/2.0, 0)
    CALL CHARC(CMENU(IMENU)(1:IMENL(IMENU)))
    CALL XLAT(0.0, -YPOL(4))
2000 CONTINUE
  CALL IDEN
  CALL ENDPIC
C
C Make segment 1 detectable.
  CALL LPSPIC(1, 1)
  RETURN

C Main program
C
C Enter loop reading menu hits. If the menu is pointed at,
C the first element of the namstack will be 1 (segment id.),
C the second element will be 1 to 4, depending on which
C menu item was hit.
1000 CONTINUE
  CALL REQHIT(3, 2, IHIT, IHLEN)
  IF (IHLEN .EQ. 2 .AND. IHIT(1) .EQ. 1) THEN
    CALL Branch(IHIT(2))
    GO TO 1000
  ENDIF

```

Scanning for menu hit in this example involves checking the 4 polygons only, the text strings will not be scanned as they belong to an undetectable segment.

**Figure 20.1** *Text menu (defined by Example 20.4).*



## 20.5 Using Pseudo Segments

Names and detectability are also stored with elements within pseudo segments. If the segments are to be used for display only, this will obviously have no significance.

However, pseudo segments may well be inserted into detectable retained segments. In that case, the resulting namestack and detectability of elements will be a combination of the state at the time a pseudo segment is inserted, and the values inserted.

If the detectability switch is set on/off within the pseudo segment, the following elements will always/never be detectable, regardless of the state when inserted. Elements generated before *LPCTL* is called the first time, will be detectable if the switch is on at the time the segment is inserted.

The names of inserted elements will be added to the namestack, but the segment identifier will not.

**Example 20.5** *Pseudo segment inserted into detectable segments.*

	Code of 'inserting' routine							
	CALL BGNPIC(101) CALL BGNNAM(10) CALL LPSCTL(1) CALL INSERT(1)				CALL BGNPIC(102) CALL BGNNAM(20) . CALL BGNNAM(30) CALL LPSCTL(0) CALL INSERT(1)			
Pseudo code	Namestack of inserted elements							
CALL BGNPIC(1) CALL NAME(1) CALL CHARC('Inherit') CALL LPSCTL(1) CALL NAME(2) CALL CHARC('Always') CALL LPSCTL(0) CALL CHARC('Never') CALL ENDPIC	101	10	1	Empty				
	101	10	2	102	20	30	2	
	Empty			Empty				





# Chapter 21

## Multi Window Devices

---

With all routine descriptions so far, it has been assumed that each device has a single image area, equal to the display surface. Today, the most commonly used graphic devices are running some sort of window system, such as X11, allowing the user to work with several image areas, i.e. windows, mapped onto the same display surface.

Existing GPGS-F application may be run without changes on such devices. The traditional display surface is then represented by a single window. However, this will obviously not utilize any of the advantages of window systems compared to conventional graphic terminals.

To utilize (some of) the possibilities of modern window systems, GPGS-F has been extended with special purpose routines to create and manipulate windows, which in this respect is referred to as **Device Windows** (DWI). This must not be confused with the kind of window described in **Chapter 2**, which is used to specify the area in user coordinate space that is to be displayed.

In addition to allow graphics to be created in several device windows, these routines simplify the use of GPGS-F in applications using the window system directly. A typical example of this, is an application using X11 toolkit routines for defining and handling the user interface, while GPGS-F is used as a output only system to create the graphics, either in windows created by GPGS-F itself or in windows created by the application.

The maximum number of simultaneously active device windows is device dependent, and may be requested by the *DATDEV* routine described on **page 23-4**. The windows are referred to by a number ranging from 1 to the allowed maximum.

*NITDEV* (**page 1-2**) must be used to initialize multi window devices in the same way as other devices, and will create a device window that is given number 1 (this is why 'old' applications may be run with such devices). Applications wanting to have full control of the window management may still achieve this, as it is possible to specify through *DEVOPT* (**page 1-4**) that the initial window is not to be displayed.

Note that drivers for multi window devices use the same colour table for all windows, i.e. changing colour definitions (see **page 11-2**) will affect the primitives of all windows.

The first multi window driver developed, was the driver for the X11 window system. This has to some degree influenced the selection of what routines were added to GPGS-F, as well as some terms and references used throughout this chapter. The routines should however have the same effect with other window systems.

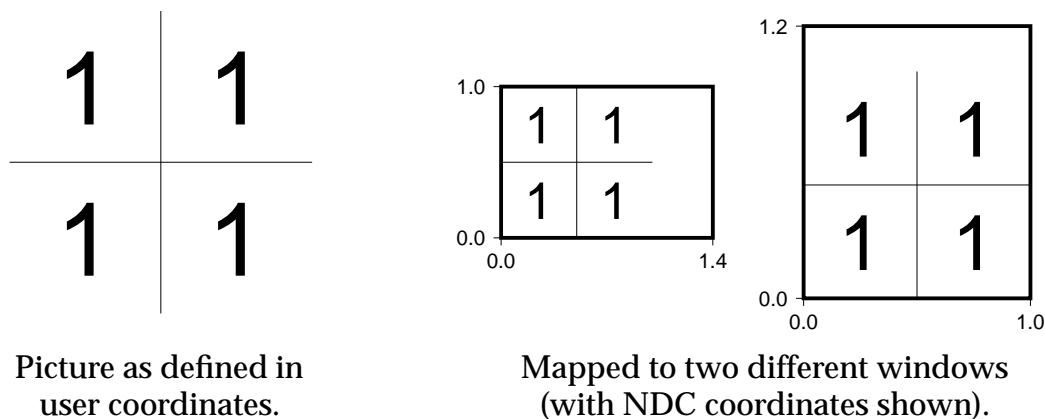
## 21.1 Window to Viewport Mapping

As described in **Chapter 2**, user coordinates are mapped to device coordinates by defining a window and a viewport. The viewport is specified in NDC coordinates, where the range 0.0 to 1.0, the default viewport, is defined to be the largest available square within the display surface.

With multi window devices, the default viewport of a given window will be the largest available square within that window. Thus, the physical size of a NDC unit will depend on the window size.

GPGS-F will store just the last window and viewport definition, not one pair for each window. If the application program uses different mappings for each window, the application itself must keep track of the values used, and call *WINDW* and *VPORT* with the actual values each time when switching between windows.

**Figure 21.1** *Window to viewport mapping.*



## 21.2 Window Management

A device window is created (opened) by

**CALL NITDWI (ldwi, ltyp, lref)**

where **ldwi** is the window number, ranging from 2 to the device dependent maximum. Value 1 may not be used, as that window is created by *NITDEV*.

**ltyp** specifies the window type, and may take one of the following values:

- 1: A new top level window. **lref** is not used.
- 2: A new subwindow (child), with GPGS-F window **lref** as parent.
- 3: A new subwindow, with the application created window **lref** as parent. **lref** must in this case be the internal window number used by the window system.
- 4: Connect to the application created window **lref**, i.e. GPGS-F will not create a new window, but will use window **lref** for drawing.

When using a window system, there is always a *window manager* running. Normally, when GPGS-F creates a top level window, the window manager will be notified, and decorate the window as any other top level window, before it is displayed. Top level windows may also be moved and/or resized by the operator, using a mouse or similar tool. How window resizing affects the drawing process is described on **page 21-10**.

Subwindows, on the other hand, will not be available to the window manager, i.e. they will not be decorated, and may not be manipulated by user actions.

The parent of a subwindow need not be a top level window, as GPGS-F does not limit the number of subwindow levels.

*DEVOPT* is used to specify the size and position of new windows to be created. In addition, options are defined for giving background colour, window and icon name, border width and colour, and window start condition, i.e. whether the window is to be displayed or just defined. The options recognized by a given device are described in **Appendix E**, or in additional driver descriptions available.

A device window is cleared by

**CALL CLRDWI (ldwi)**

The *CLRDEV* routine (**page 1-2**) will clear *all* windows.

When a device window is no longer to be used, it may be released (deleted) by

**CALL RLSDWI (ldwi)**

The given window number **ldwi** may then be reused by *NITDWI*.

Window number 1 may *not* be released by *RLSDWI*, as this has to be open until the driver is released by *RLSDEV* (**page 1-2**). *RLSDEV* will also release other windows that are still open.

If **ldwi** specifies a window that was created by the application program, i.e. **ltyp**=4 through *NITDWI*, the window will not be deleted by *RLSDWI*, but it will no longer be accessible through GPGS-F.

When a window is deleted, GPGS-F will check if it has any descendants, and if so, delete these as well.

*CLRDWI* and *RLSDWI* will delete all retained segments belonging to the given window and possible subwindows. More information on using retained segments with multi window devices are given on **page 21-10**.

### **Example 21.1    *Window creation.***

```

C  ****
C  COMPLETE WORKING PROGRAM
C  ****
C
C  Define options for device driver 72 (X11).
C  (in all figures in this chapter, 100 pixels is 1 cm)
      INTEGER I72OPT(11, 5)
      DATA I72OPT/0, 0, 100, 700, 500,1000, 11, 0, 0, 0, 3,
+           0, 7, 550,1050, 150, 650, 11, 0, 0, 0, 3,
+           0, 5, 200, 550, 100, 450, 1, 0, 0, 0, 2,
+           0, 0, 200, 450, 100, 300, 1, 0, 0, 0, 2,
+           0, 0, 50, 200, 20, 220, 1, 0, 0, 0, 1/
C
      CALL GPGS
C
C  Set options for the device driver and window 1.
      CALL DEVOPT(I72OPT(1,1), 11, RDUM, 0, CDUM, 0)
      CALL DEVOPT(IDUM, 0, RDUM, 0, 'Window 1', -1)
C
C  Initialize device, implies creation of window 1.
      CALL NITDEV(72)
C
C  Specify character size and alignment, and draw a picture
C  segment in window 1.
      CALL CSIZES(0.25, 0.5)
      CALL CJUST(0.5, 0.25)
      CALL DRWSEG(1)
C
C  Set options for window 2, a new top level window.
C  Create the window and draw a picture segment.
      CALL DEVOPT(I72OPT(1,2), 11, RDUM, 0, CDUM, 0)
      CALL DEVOPT(IDUM, 0, RDUM, 0, 'Window 2', -1)
      CALL NITDWI(2, 1, IDUM)
      CALL DRWSEG(2)
C
C  Window 3, a subwindow of window 1.
      CALL DEVOPT(I72OPT(1,3), 11, RDUM, 0, CDUM, 0)
      CALL NITDWI(3, 2, 1)
      CALL DRWSEG(3)
C
C  Window 4, a subwindow of window 2.
      CALL DEVOPT(I72OPT(1,4), 11, RDUM, 0, CDUM, 0)
      CALL NITDWI(4, 2, 2)
      CALL DRWSEG(4)
C
C  Window 5, a subwindow of window 3.
      CALL DEVOPT(I72OPT(1,5), 11, RDUM, 0, CDUM, 0)
      CALL NITDWI(5, 2, 3)
      CALL DRWSEG(5)
C
C  Wait for operator to push a key or mouse button, just to
C  be able to inspect the result before releasing the driver.
      CALL REQLOC(201, XDUM, YDUM)
C
      CALL RLSDEV(72)
      END

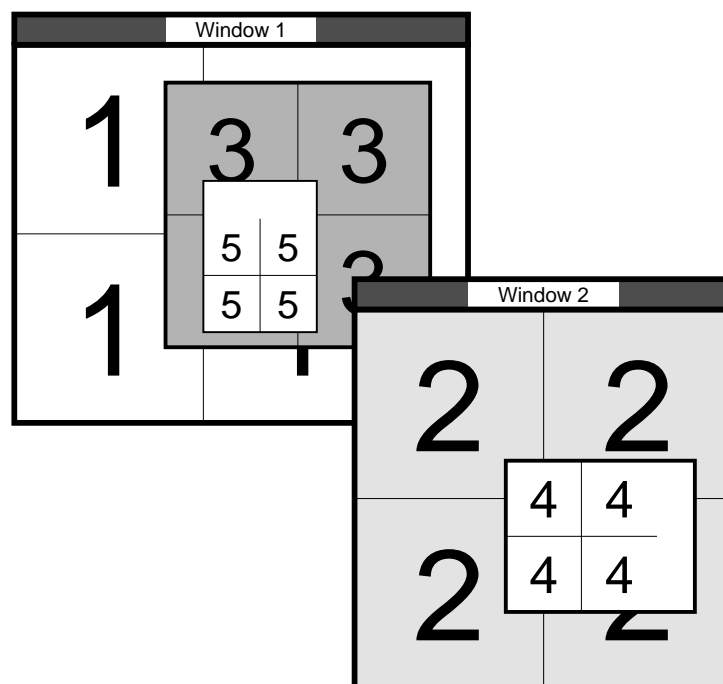
```

```

SUBROUTINE DRWSEG(ISEG)
C
C Draw a picture segment in current device window.
C
REAL XPOS(4), YPOS(4)
DATA XPOS/0.25, 0.75, 0.25, 0.75/
DATA YPOS/0.25, 0.25, 0.75, 0.75/
C
CALL BGNPIC(ISEG)
CALL LINE(0.0, 0.5, 0)
CALL LINE(1.0, 0.5, 1)
CALL LINE(0.5, 0.0, 0)
CALL LINE(0.5, 1.0, 1)
DO 1000 IPOS=1,4
CALL LINE(XPOS(IPOS), YPOS(IPOS), 0)
CALL CHARI(ISEG, 1)
1000 CONTINUE
CALL ENDPIC
RETURN
END

```

**Figure 21.2 Window creation (result of Example 21.1).**



Although there may be several open device windows, output is generated for just one window, the *current* output window, at a time. This window is selected by

**CALL SELDWI (ldwi)**

where **ldwi** is an existing window. *NITDWI* implies *SELDWI*, i.e. the window that will receive output is the one selected by the last *NITDWI* or *SELDWI*.

If the current window is released by *RLSDWI*, window number 1 is set to current, no matter if it is visible or not.

## 21.3 Window Operations

GPGS-F allows several operations to be applied to device windows. For top level windows, the same operations may normally be performed by the operator through the window manager, while subwindows may only be manipulated by program control.

The visibility of a device window is controlled by

**CALL VISDWI (ldwi, lsw)**

where **lsw**=0 means invisible ('unmapped') and **lsw**=1 means visible ('mapped').

If a top level window is made invisible, it will normally be made into an icon.

GPGS-F does not check if the current window is visible when generating output. Whether the generated primitives will appear when the window is later made visible depends on the window system.

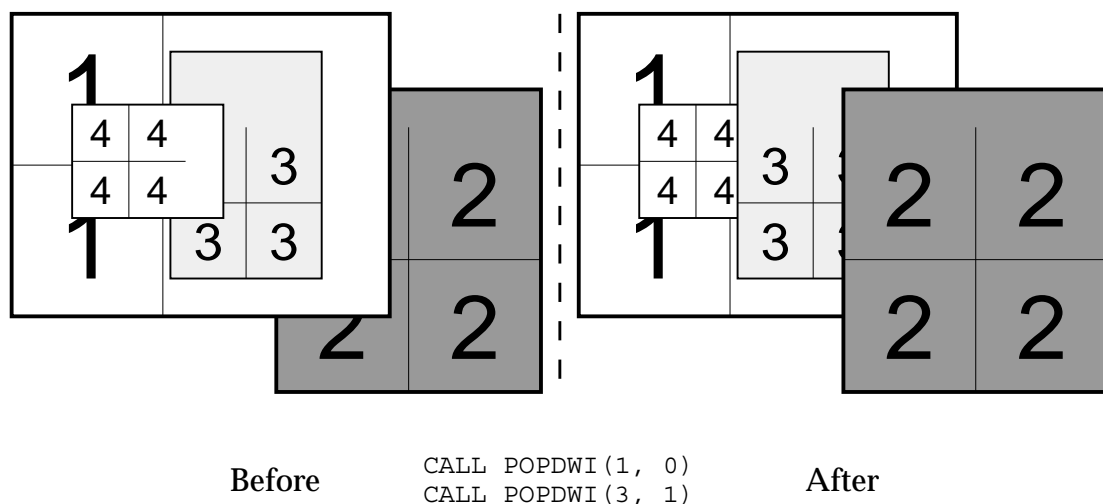
The stacking order among siblings (=windows with the same parent) may be changed by

**CALL POPDWI (ldwi, lsw)**

where **lsw**=1 will move window **ldwi** to the front of all its siblings, and **lsw**=0 will move it behind all its siblings.

When a new window is created, it is always put in front of its siblings.

**Figure 21.3** *Changing stacking order.*



Window 3 is not moved in front of window 2, as they are not siblings.

The position of a window may be changed by

**CALL MOVDWI (ldwi, lxpos, lypos, lmod, lcorn)**

**lxpos** and **lypos** specify the *distance* between a given corner of the window and the corresponding corner of its parent window. The unit used to specify this distance is selected by **lmod**. If **lmod** is 0, the distance is given as a percentage of the parent window size. If **lmod** is 1, the distance is given as pixels.

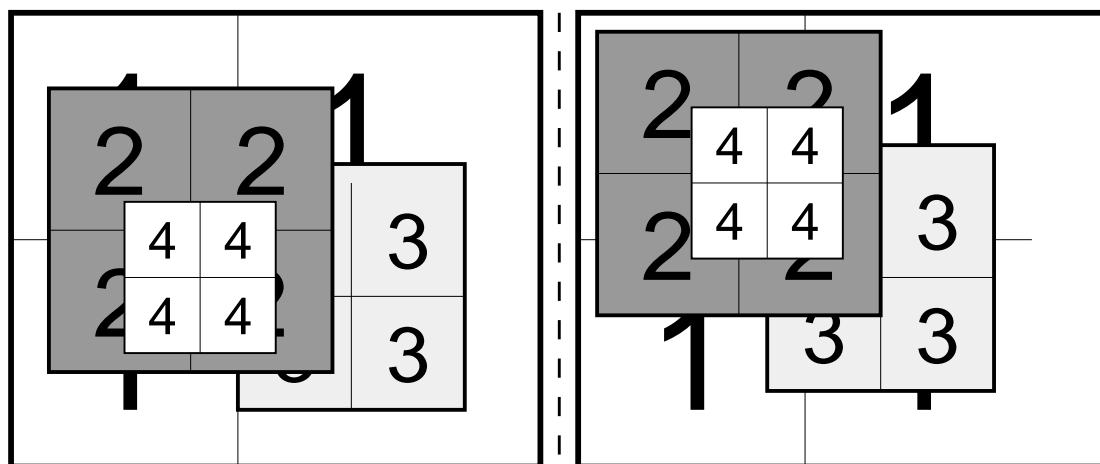
**lcorn** specifies which corner the distance is specified for (1=lower left, 2=lower right, 3=upper right, 4=upper left).

When a percentage value are used, GPGS-F will compute the distance in pixels based on the largest possible square of the parent window.

GPGS-F does *not* limit a subwindow to be within its parent. Whether the graphics in invisible parts of a window become visible when the window is later moved into its parent, or the parent is enlarged, depends on the properties of the window system (see [section 21.6](#) on [page 21-10](#)).

Note that the parent of top level windows is the root window, i.e. the complete display surface.

**Figure 21.4** *Moving windows.*



Before

```
CALL MOVDWI (2, 25, 25, 1, 4)
CALL MOVDWI (3, 150, 100, 1, 2)
CALL MOVDWI (4, 125, 75, 1, 1)
```

After

The size of a window is changed by

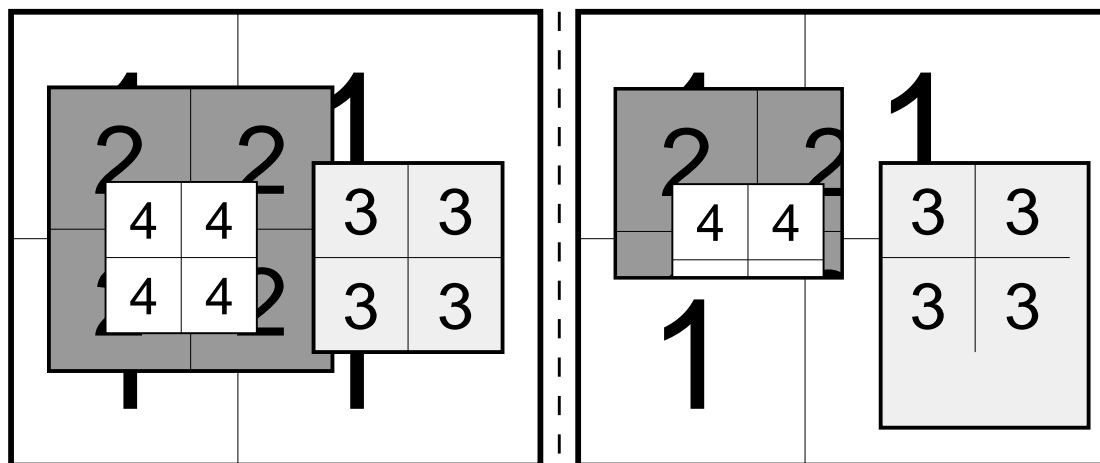
**CALL RSZDWI (ldwi, lwid, lhei, lmod)**

where the width (**lwid**) and height (**lhei**) may also be specified as a percentage of the parent window (**lmod=0**) or as pixels (**lmod=1**). When a percentage value is given, the size will be computed the same way as with *MOVDWI*, i.e. based on the largest possible square in the parent window.

The position of the upper left corner, relative to its parent, will not move when a window is resized. The same is true for subwindows of the changed window, and also the graphics within the window.

Note that the graphics within the window is not resized with the window. How to adjust the graphics to the new size is discussed in **section 21.6** on **page 21-10**.

**Figure 21.5** *Resizing windows.*



Before

```
CALL RSZDWI (2, 300, 250, 1)
CALL RSZDWI (3, 275, 350, 1)
```

After

The last window operation available through GPGS-F, is to reparent a window, i.e. move a window from one parent to another. This is done by

**CALL RPADWI (ldwi, lref, lreftp)**

where **lref** is the GPGS-F number of the new parent if **lreftp=0**, or the internal window number if **lreftp=1**. Thus, a window with a GPGS-F window as parent, may well be changed to have an application generated window as parent, and vice versa.

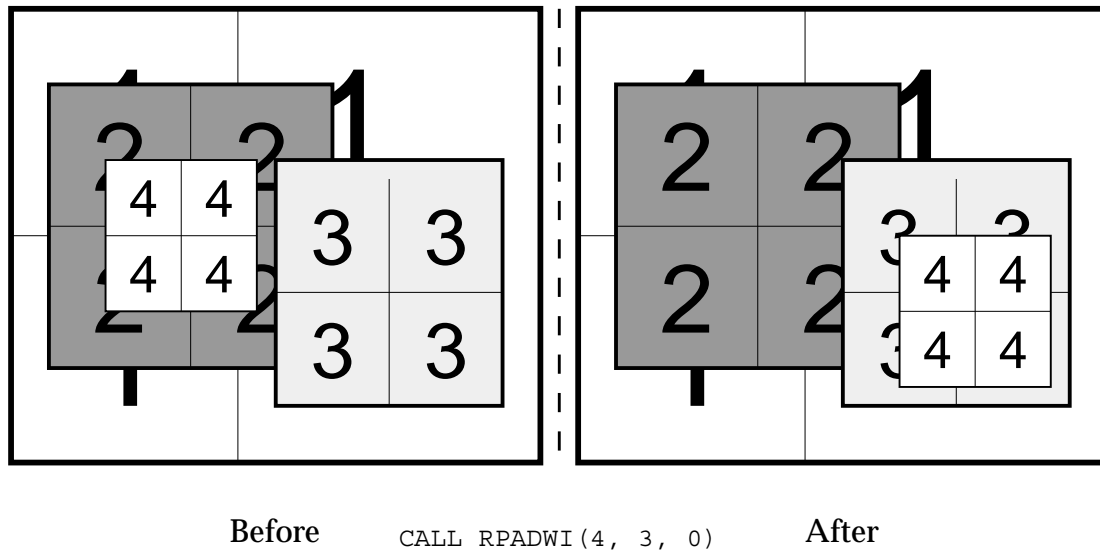
When a window is reparented, the position of the upper left corner will be the same in the new parent as in the old one. If the new parent is smaller than the old one, the reparented window may then be partly or completely invisible.



If the window that is reparented contains subwindows, these will of course be moved with the window.

*Note!* A top level window may *not* be reparented, and a subwindow may not be converted to a top level window by giving the root window as the new parent.

**Figure 21.6** *Reparenting a window.*



## 21.4 Window Numbers

As stated at the start of this chapter, device windows are referred to by a number from 1 and up. This numbering is internal to GPGS-F, and has no relevance to the window identifiers used by the window system running.

When the application program uses window system routines directly, the application may however need to know the internal window number of a window generated by GPGS-F. This may be requested by

**CALL INQDWI (ldwi, lwsid, lchg)**

where **ldwi** is the GPGS-F window number and **lwsid** is the returned internal number. If GPGS-F is not able to find the internal number, **lwsid** will be set to a value that is not a legal window identifier.

(The **lchg** argument is described on [page 21-10](#)).

## 21.5 Retained Segments

Retained segments are handled much the same way by multi window devices as by other devices. The only difference is that the segments belong to a given device window in addition to the device they were created for.

The window number is however *not* stored with the segment identifier, as the device number is. Thus, the same segment identifier may not be used by two segments belonging to the same device, even if they belong to two different device windows.

When a retained segment is created, it will be drawn in the current device window. A segment may however be deleted (by *DELPIC*, [page 16-4](#)), and its attributes changed ([Chapter 17](#)), without first selecting the window the segment belongs to.

The *REDRAW* routine will, as described on [page 16-4](#), redraw all visible segments belonging to the current device. With multi window devices, this means that all segments in all windows are redrawn.

It is also possible to redraw the segments belonging to a single window by

**CALL RDRDWI (ldwi)**

where **ldwi** is the window to redraw.

## 21.6 Updating Window Contents

If a window containing GPGS-F graphics is resized, either by *RSZDWI* or an operator action, GPGS-F will not do any automatic redrawing, and if adding more graphics to the window, the 'old' size will still be used when mapping user coordinates to device coordinates.

This 'old' size is the window size as it was when the window was last empty. The *actual* window size is recorded by GPGS-F when *BGNPIC* is called and the current output window is empty, or when the segments of the window is redrawn by *RDRDWI* or *REDRAW*.

As the application program itself must handle redrawing, it is possible to ask GPGS-F whether the window size is changed since it was last empty.

**CALL INQDWI (ldwi, lwsid, lchg)**

will return 1 through **lchg** if the window size is changed, 0 if not.

If the window is empty when *INQDWI* is called, 0 is returned true **lchg**.

(The **lwsid** argument was described on [page 21-9](#)).

**Example 21.2    *Updating window contents.***

```
      SUBROUTINE CHKDWI (IDWLST, ILTH)
      INTEGER IDWLST (ILTH)
C
C   A (user supplied) routine to check if any of the device
C   windows in use need redrawing.
C
C   It is assumed that all segments are retained.
C
C   Input:
C       IDWLST:   List of length ILTH, containing a 1 for each
C                 window that is currently in use.
C
      DO 1000 IDWI=1,ILTH
        IF (IDWLST(IDWI) .EQ. 1) THEN
          CALL INQDWI (IDWI, IDUM, ICHG)
          IF (ICHG .EQ. 1) THEN
            CALL RDRDWI (IDWI)
          ENDIF
        ENDIF
      1000 CONTINUE
C
      RETURN
      END
```

There are also additional cases where a window needs redrawing, such as when an invisible window is made visible by *VISDWI* (page 21-6), or when obscured parts of a window is exposed.

With some window systems, redrawing in such cases are taken care of by the window system itself (with X11 there is a window property called *Backing\_Store* that indicates whether a window should be automatically redrawn or not).

If a window containing graphics is shrunk, parts of the graphics may fall outside the new size (as in **Figure 21.5** on page 21-8). If the window is later resized to its original size or larger, these cut-off parts will not become visible again, even if the window system handles window redrawing. Using GPGS-F is then the only way to redraw the picture.

GPGS-F does not provide any routines that indicates whether redrawing is necessary because of other reasons than resizing. Thus, if redrawing is not done by the window system, the application itself must include methods for detecting such events.

## 21.7 Requesting Window Size

The size of a window may be requested through *DATDEV* (page 23-4), which returns data for the current output window, or by

**CALL DATDWI (ldwi, larr(1), Lthi, Farr(1), Lthf)**

which returns data for window **ldwi**. **Farr** is defined as for *DATDEV*, but the maximum length is 4, i.e. maximum viewport size in X, Y and Z, and default viewport size in meters. **larr** will return the position of the left, right, bottom and top window borders in pixels, measured from the lower left corner of the parent window.

**Lthi** and **Lthf** gives the number of values to return through **larr** and **Farr**.

When *DATDEV* or *DATDWI* is called, GPGS-F will return the window size as it was when the window was last empty, i.e. not necessarily the actual size (see previous section). If the window is currently empty, the actual size will however always be returned.

### Example 21.3 Requesting current window size.

The window as originally defined.

3	3
3	3

```
CALL RSZDWI (3, 200, 250, 1)
CALL DATDWI (3, IARR, 4, RDUM, 0)
```

will return [100, 450, 100, 400] trough *IARR*, i.e. the 'old' size.

3	
3	

```
CALL RDRDWI (3)
CALL DATDWI (3, IARR, 4, RDUM, 0)
```

will return [100, 300, 100, 350] trough *IARR*.

3	3
3	3

## 21.8 Interaction

All interaction facilities described in **Chapter 8**, as well as the pick input method described in **Chapter 20**, may be used with multi window devices.

By default, input is read from the current output window. This is however not very convenient, as applications using multiple windows are commonly designed to receive input from some windows, while other windows are used for output only.

To give the application programmer full control of which windows are to be used for input and not, an input mode may be set for each window, by

**CALL INPDWI (ldwi, lsw)**

where **lsw**=1 means that window **ldwi** may receive input, and **lsw**=0 means that input no longer is possible. If **ldwi** is set to 0, **lsw**=1 means that *all* windows may receive input, while **lsw**=0 means that *no* windows may receive input.

Note that if all windows are set to receive input, this applies to the *currently* open windows. If a new window is created, this has to be explicitly set to receive input by calling *INPDWI* once more.

If **ldwi** is set to -1, the default condition is reset.

If more than one window is defined to receive input, the user must of course have a method for finding what window actually received the input. Instead of defining a complete new set of input routines with an extra argument, a single utility routine has been defined as

**CALL READWI (ldwi)**

which will return, through **ldwi**, the number of the window in which the *last* input operation was performed. This routine should be called immediately after the actual input routine is called.

Following a *SMPLOC/SMPHIT* call (**page 8-4**), 0 will be returned through **ldwi** if the cursor was not in a GPGS-F window, or in a GPGS-F window that was not set up to receive input by *INPDWI*.

## 21.9 Background Device

Generally, all GPGS-F routines will give the same effect with multi window devices as with traditional graphic terminals.

When using a background device (described in **Chapter 19**), there is however one exception. The *BACDRW* routine will for a traditional driver copy all visible segments on the screen to the specified background device. Used with multi window drivers, *BACDRW* will copy the visible segments in the *current* window only.



## Chapter 22

# Hidden Lines and Surfaces Removal

---

When drawing 3D objects using the basic routines of GPGS-F, the resulting drawing will display all primitives generated for the object. In some cases this is what the user wants, but in most cases the preferred result is one that does not display primitives that is hidden by primitives in front of it.

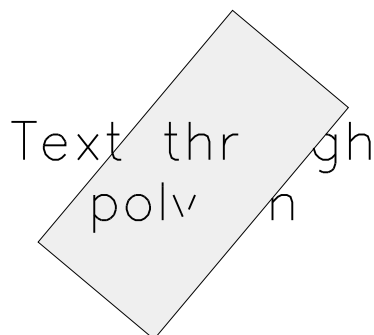
To make this possible, GPGS-F has been extended with a module for performing hidden lines and surfaces removal (HLHS module for short). A surface is in this connection equivalent to a polygon generated by one of the GPGS-F polygon routines described in **Chapter 12**. It is not possible to define surfaces by other means.

Removal of hidden lines means that parts of lines that are hidden by surfaces are removed (in some other systems, the term is used when the visual parts of polygon edges are displayed as lines).

When using the HLHS module, a *copy* of the polygons and lines generated are stored. That is, the primitives are still passed to the active device driver, in the same way as when the module is not used (more on this on **page 22-6**). Upon request from the user, the result of the HLHS calculation is inserted into the currently open picture segment.

Not only lines defined by the user are stored, but also lines generated by internal GPGS-F routines. That is, software circles and characters are stored as a number of lines, and invisible parts are removed when displayed. Hardware characters, circles and markers are *not* stored.

**Figure 22.1** *Hidden lines removal.*



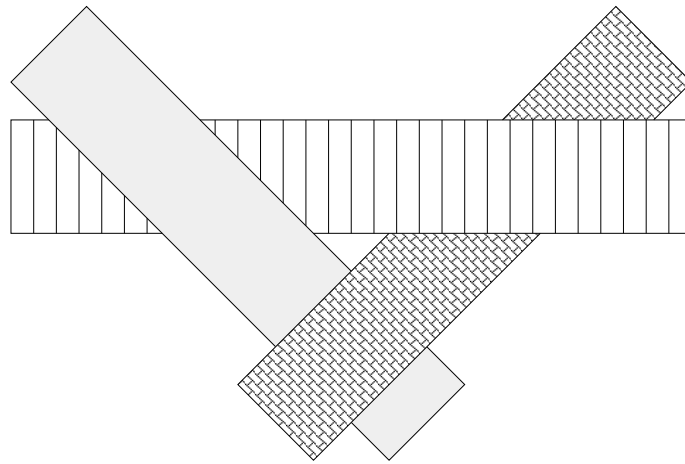
Software text partly in front of, partly behind the polygon.  
Each line is checked for visibility.

As visible parts of lines are drawn, original lines may be split into several line segments. If using GPGS-F linetypes 3 to 5, this means that the line pattern is not maintained as each line segment will start a new pattern. To fully control the linepattern, user defined linetypes (described in **Chapter 9**) should be used.

The HLHS module of GPGS-F is capable of handling all polygon shapes, including polygons with holes. The polygons *must* however be plane.

The module is very general in that it handles cyclic overlap (**Figure 22.2**) and intersecting polygons (**Figure 22.4**), in addition to 'normal' overlap.

**Figure 22.2** *Cyclic overlap.*



## 22.1 HLHS Module Control

The HLHS module is initialized by

**CALL NITHID**

which will (re)set the module to its defined initial state.

*NITHID* will set the *storage switch* of the module on, i.e. copies of all polygons and lines subsequently created will be stored by the HLHS module.

The state of the storing switch is controlled by

**CALL HIDCTL (lsw)**

where **lsw**=0 will turn the switch off, **lsw**=1 will turn it on.

Changing the state of the storing switch will not affect the polygons and lines already stored by the HLHS module.



## 22.2 Inserting the Result

When all polygons and lines that are to take part in the HLHS calculation are stored, the result is inserted into the currently open segment by

**CALL INSHID (lopts(1), Length)**

where **lopts** is an option array of length **Length**. With the current version 3 options are defined.

**lopts(1)** controls what to draw, and the drawing sequence.

**0:** Draw polygons front to back, and draw all lines afterwards (**default**).

**1:** Draw polygons back to front, and draw all lines afterwards.

**2:** Draw polygons and lines back to front *with no HLHS removal*.

This may be used when all polygons are solid or patterned, and it is known that there is no overlap in Z direction between the polygons, and no lines passes through polygons. The reason why the result will be correct is the fact that the polygons when drawn will obscure what has previously been drawn (assuming a raster device is used).

**10-12:** As **0-2**, except that only the polygons are drawn, not the lines.

**98:** Draw only lines, and remove the lines from storage afterwards.

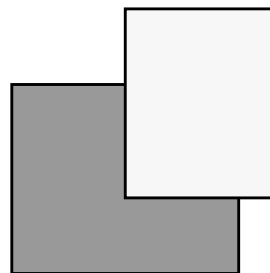
**99:** Draw only lines, and leave the lines in storage.

Drawing polygons back to front will be slightly slower than front to back, but with colour or grey scale raster devices, the result will in most cases look more correct (as shown by **Figure 22.3**). With pen plotters, or if not using colours, the visual effect will be the same.

**Figure 22.3** *Drawing sequence.*

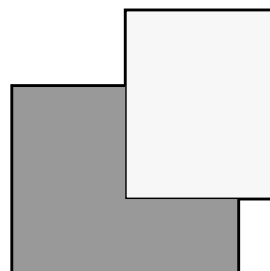
**lopts(1)=1**

Polygons drawn back to front.



**lopts(1)=0**

Polygons drawn front to back. The perimeter of the front polygon is partly erased when drawing the polygon behind it.



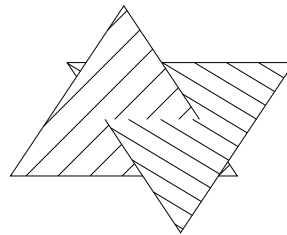
**lopts(2)** controls what polygon edges to draw.

- 0: Draw original edges only (**default**).
- 1: Draw original edges, and intersection lines between polygons.
- 2: Draw all polygon edges, including new edges resulting from the HLHS calculation.

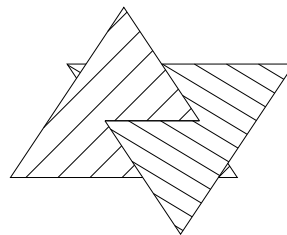
This option applies to the edges of hollow polygons, and the edges of filled polygons if perimeter drawing has been selected by *PRCIND* (see **page 12-4**).

**Figure 22.4** *Drawing polygon edges.*

**lopts(2)=0**  
Only original edges drawn.



**lopts(2)=1**  
The intersection line between the two polygons is drawn in addition.



*PRCIND* should always be used if the edges of filled polygons are to be drawn. Drawing the same polygon twice, first as a filled polygon and then as a hollow polygon will not be correct, as only one of the two will be visible, the other one will be completely hidden.

**lopts(3)** may be used to control how filled polygons are drawn on line drawing devices.

- 0: Just pass the polygons to the device, and hope for the best (**default**).
- > 0: Draw the perimeter of solid and patterned polygons.
  - 1: Use the interior colour index for the perimeter.
  - 2: Use the perimeter colour index if set, if not, use the interior colour index.
  - 3: Use colour index 1 for drawing all perimeters.

The interior colour index means the current index at the time a polygon was created.

As *INSHID* is called to display the result of the HLHS calculation, the data will *not* be deleted from the module. It is possible to add new polygons and lines at a later stage, and call *INSHID* again to display a new result. With the current version, it is not possible to delete individual polygons and/or lines from the module.

If several independent drawings are to be made by the same application program, *NITHID* must be used to reset the HLHS module to its initial state.

**Example 22.1    *Basic use of the HLHS module***

```
C
C  Initialize a device and the HLHS module.
C
      CALL GPGS
      CALL NITDEV(idev)
      CALL NITHID
C
C  Draw some polygons (and maybe lines).
C
      CALL BGNPIC(1)
      CALL User-routine-to-draw
      CALL ENDPIC
C
C  Clear the screen, or in the case of a multi window device,
C  create a new window.
C
      IF (Multi_Window_Device) THEN
        CALL NITDWI(2, 1, 0)
      ELSE
        CALL CLRDEV(idev, 0)
      ENDIF
C
C  Draw some explanatory text. This is not to be part
C  of the HLHS calculation, so storing is turned off.
C
      CALL HIDCTL(0)
      CALL BGNPIC(2)
      CALL User-routine-to-draw-text
C
C  Then insert the result of the HLHS calculation.
C
      CALL INSHID(idum, 0)
      CALL ENDPIC
      CALL RLSDEV(idev)
      END
```

## 22.3 Using the Dummy Device

As mentioned at the start of this chapter, what is stored by the HLHS module is a *copy* of the polygons and lines generated.

The polygons and lines are at the same time sent to the current device driver and displayed. In most cases, however, this display is of no interest. To suppress displaying the input data, GPGS-F driver number 1, the dummy device, may be used during the storing stage of the application program.

### Example 22.2 *Using the dummy device with the HLHS module.*

```
C
C  Initialize the dummy device and the HLHS module
C
      CALL GPGS
      CALL NITDEV(1)
      CALL NITHID
C
C  Draw some polygons (and maybe lines).
C
      CALL BGNPIC(1)
      CALL User-routine-to-draw
      CALL ENDPIC
C
C  Initialize the device to use, and insert
C  the result of the HLHS calculation.
C
      CALL NITDEV(iddev)
      CALL BGNPIC(2)
      CALL INSHID(idum,0)
      CALL ENDPIC
C
C  Add some more polygons to the HLHS module.
C
      CALL SELDEV(1)
      CALL BGNPIC(3)
      CALL User-routine-to-draw
      CALL ENDPIC
C
C  Clear the display, and insert the new result of
C  the HLHS calculation.
C
      CALL SELDEV(iddev)
      CALL CLRDEV(iddev, 0)
      CALL BGNPIC(4)
      CALL INSHID(idum, 0)
      CALL ENDPIC
```

## 22.4 Front- and Back-Facing Polygons

When a polygon (copy) is passed to the HLHS module, a front and back face is defined. The reason for this is that the application program may choose to store only the polygons whose front face is towards the viewer (front-facing polygons).

The front face of a polygon is defined as follows:

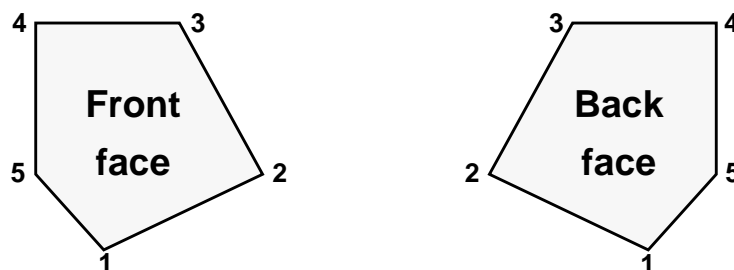
A local right-hand coordinate system is set up. One of the edges is defined to be the X axis, with vertex  $n$  as startpoint and vertex  $n+1$  as endpoint.

The Y axis is in the polygon plane, with the positive direction towards the interior of the polygon.

The front face of the polygon is then defined to be the face that is seen from the positive part of the Z axis.

Following from this, a front-facing polygon will have its local Z axis pointing from the display surface towards the viewer, while a back-facing polygon will have its local Z axis pointing 'into' the display surface. Thus, if the vertices are defined in the opposite sequence, the front and back face will be swapped.

**Figure 22.5** *Front and back face of polygons.*



The numbers show the sequence of the vertices as supplied to the polygon drawing routine.

In many cases, especially when the polygons define solids, it is known that back-facing polygons will in no case be visible. These polygons need therefore not take part in the HLHS calculation, and the CPU time needed will be reduced.

Whether back-facing polygons are to be stored is selected by

**CALL BFACEV (lsw)**

where **lsw**=0 means back-facing polygons will not be stored, 1 means they will be stored.

The default value, set by *NITHID*, is 0.

## 22.5 Polygon Attributes

As the result of HLHS removal is drawn, the polygons are to have the same attributes as when generated, i.e. the attributes must be stored as well. The table below shows what attributes are stored with the different polygon types. For a detailed description of polygon types and attributes, refer to **Chapter 12**.

**Table 22.1** *Polygon attributes stored by the HLHS module.*

Polygon type	Attributes stored
Solid	The colour index of the interior, and perimeter colour index if set (see the <i>PRCIND</i> routine, <b>page 12-4</b> ).
Hollow	The colour index.
Hatched	If low quality rendering is selected (see the <i>SOFPOL</i> routine, <b>page 12-6</b> ), the hatch index. If medium or high quality rendering, the hatch angle and distance. In both cases, the colour index of the hatch lines, and of the perimeter if set.
Patterned	Pattern index, and perimeter colour index if set. If user defined patterns are used, this means that if the pattern is redefined between polygon generation and display of the HLHS result, the <i>new</i> definition will be used.

## 22.6 Limitations

The current version of the HLHS module stores the lines and polygons in internal GPGS-F arrays. As the size of these arrays are defined at compilation time in Fortran, there is a limit to the number of lines and polygons that may be stored.

The array sizes are installation dependent (see **Appendix A**), i.e. they may be changed by the site responsible for GPGS-F. The current setting may be requested by the *DATHID* routine described on **page 23-7**.

# Chapter 23

## Fetching System Status Data

---

GPGS-F contains a family of routines returning various system status information to the application program. These are most useful when writing utility routines that are to be used by several applications. By inquiring GPGS-F status information in such a utility routine, it is possible to make the routine itself quite independent of the calling routine.

An example of such a utility routine is one that is to write a text in a given size and with a given font. As the routine returns control to the calling routine, all settings should be as when the routine was entered. To make this possible the routine must know the current window and viewport setting, current character size and character font. All this information is available through inquiry routines, making it unnecessary for the calling routine to pass it through the call.

This chapter describes the routines returning GPGS-F system status information. If relevant, the routine(s) setting the status is given with the description. If the setting routine has not yet been used, the system default values are returned. Note that there are cases where the returned value will not be the last one set, as GPGS-F may have changed the value. An example of this is the current colour index, which will always be reset to its default value when a new picture segment is opened.

When values are returned through arrays, the user must in most cases specify the number of values to be returned. It is not possible to specify the array index of the first element, i.e. if the user wants **N** values to be returned, array elements **1** through **N** are always returned.

In a few cases, array elements are marked as *Not Used*. The reason why these are not deleted, is that the array ordering must be kept for compatibility reasons.

### Device number

**CALL DATDNO (ldev)**

**ldev:** Device number of the current active device driver, selected by *NITDEV* or *SELDEV* (page 1-2). If there is no active driver, zero is returned.

### Picture segment number

**CALL DATPNO (lident)**

**lident:** Identifier of the current open picture segment, set by *BGNPIC* (page 3-1). If there is no open picture segment, zero is returned.

## Window

**CALL DATWIN (Warr3(1))**

**Warr3:** Current window limits, set by *WINDW* / *WINDW3* (page 2-2). **Warr3** must be a 6 element array as the Z limits are always returned, even if the user has set a 2 dimensional window.

The window limits are returned in the same sequence as set by *WINDW* / *WINDW3*, i.e. [**Xlow**, **Xhigh**, **Ylow**, **Yhigh** (**Zlow**, **Zhigh**)].

## Viewport

**CALL DATVP (Varr3(1))**

**Varr3:** Current viewport limits, set by *VPORT* / *VPORT3* (page 2-3).

The same rules apply as for *DATWIN* described above.

## Current position

**CALL DATUSR (Xusr, Yusr, Zusr)**

**Xusr, Yusr, Zusr:** Current position in user coordinates.

**CALL DATPOS (Xpos, Ypos, Zpos)**

**Xpos, Ypos, Zpos:** Current position transformed by the system transformation matrix.

## Picture element attributes

**CALL DATATR (larr(1), Lthi, Farr(1), Lthf)**

The number of values to be returned must be specified by **Lthi** (number of integers) and **Lthf** (number of real numbers).

**larr(1):** Blinking status, set by *BLICTL* (page 13-2). 0 = disabled, 1 = enabled.

**larr(2):** *Not Used*.

**larr(3):** Depth modulation, set by *DEPCTL* (page 13-2). 0 = disabled, 1 = enabled.

**larr(4):** Element detectability, set by *LPSCCTL* (page 20-3). 0 = disabled, 1 = enabled.

**larr(5):** Current colour index, set by *COTIND* (page 11-2).

**Farr(1):** *Not Used*.

**Farr(2):** Linewidth scaling factor, set by *LINWID* (page 13-1).



### Clipping mode

**CALL DATCLI (lswtch)**

**lswtch:** Current clipping mode, set by *CLICTL* (page 2-4).  
0 = clipping off, 1 = clipping on.

### Transformation mode

**CALL SAVMOD (lswtch)**

**lswtch:** Current transformation mode, set by *MODTRN* (page 6-16).  
0 = space mode, 1 = picture mode.

### Circle generation mode

**CALL DATCIR (lmod, Dist)**

**lmod:** Circle generation mode, set by *SOFCIR* (page 4-10).  
0: Hardware generation selected.  
1: Software generation selected, with number of line segments to approximate a circle to be computed by GPGS-F.  
>1: Software generation selected. **lmod** returns the number of line segments to approximate a circle, as specified by the user.  
**Dist:** Circle approximation value, set by *CIRAPR* (page 4-10).

### Character attributes

**CALL DATCHR (larr(1), Lthi, Farr(1), Lthf)**

The number of values to be returned must be specified by **Lthi** (number of integers) and **Lthf** (number of real numbers).

**larr(1):** Character generation mode, set by *SOFCHA* (page 7-6).  
0: Hardware generation selected.  
1: Software generation selected.  
**larr(2):** System escape character in A1 format, set by *CESCAP* (page 7-2).  
**larr(3):** Character language, set by *CLANG* (page 7-10).  
**larr(4):** Character font, set by *CFONT* (page 7-8).  
**Farr(1):** Character space dimension, X direction. Set by *CSIZES* (page 7-4).  
**Farr(2):** Character space dimension, Y direction. Set by *CSIZES*.  
**Farr(3):** Character box fraction, X direction. Set by *CSIZEL* (page 7-4).

- Farr(4):** Character box fraction, Y direction. Set by *CSIZEL*.
- Farr(5):** Sine of character rotation angle. Computed by GPGS-F.
- Farr(6):** Cosine of character rotation angle. Computed by GPGS-F.
- Farr(7):** Character shearing factor. Set by *CSHEA* (page 7-5).
- Farr(8):** Character rotation angle in radians. Set by *CROTA(D)* (page 7-6).
- Farr(9):** Character alignment factor, X direction. Set by *CJUST* (page 7-7).
- Farr(10):** Character alignment factor, Y direction. Set by *CJUST*.

### Marker size

**CALL DATMAR (Size)**

**Size:** Marker size in NDC. Set by *MSIZE* (page 4-12).  
If *MSIZE* has not been called, 0.0 is returned.

### Device data

**CALL DATDEV (larr(1), Lthi, Farr(1), Lthf)**

The number of values to be returned must be specified by **Lthi** (number of integers) and **Lthf** (number of real numbers).

The data are returned for the current active device driver.

- Farr(1):** Maximum available viewport, X direction (NDC).
- Farr(2):** Maximum available viewport, Y direction (NDC).
- Farr(3):** Maximum available viewport, Z direction (NDC). Two dimensional devices will return a value of 10000.0
- Farr(4):** Default viewport size in meters.
- Farr(5):** Spotsize in X direction (NDC), i.e. the inverse of the number of pixels within the default viewport.
- Farr(6):** Maximum line thickness in millimeters (approximate value).
- Farr(7):** Spotsize in Y direction (NDC). Will be equal to **Farr(5)** if device pixels are square.
- larr(1):** *Not used.*
- larr(2):** Number of keyboard input devices (input tool 2). Always 0 or 1.
- larr(3):** Maximum length of namestack for pick input device (input tool 3), including the segment identifier. 0 means tool not supported, 1 means only segment name etc.
- larr(4):** Number of valuator input devices (input tools 101 to 199).
- larr(5):** Number of locator input devices (input tools 201 to 299).

**larr(6):** *Not used.*

**larr(7):** Number of button (function key) input devices (input tools 401 to 499).

**larr(8):** Storage method used for retained segments.

- 0: The device does not support retained segments.  
All printers/plotters return this value.
- 1: All segments *must* be retained, with storage in GPGS-F buffers, i.e. **RETAIN** (page 16-1) and **NITBUF** (page 14-3) must be used.
- 2: Optional retained segment, selected by **RETAIN**. Segments are stored by the device itself, i.e. **NITBUF** is not necessary.
- 3: Optional retained segments, selected by **RETAIN**. Segments are stored in GPGS-F buffers, defined by **NITBUF**, while segment operations are handled by the device or driver.
- 4: As 3, except that segment operations are handled by the **SIMU** driver.  
This is the most common value returned by graphic terminals.

**larr(9):** Image transformation (**Chapter 18**) capabilities.

- 0: No image transformations available.
- 1: 2D translation.
- 2: All 2D transformations except shearing.
- 3: All 2D transformations.
- 4: All 3D transformations except perspective.
- 5: Full 4×4 transformation matrix.

**larr(10):** Hardware (or driver) clipping.

- 0: No clipping, i.e. primitives specified outside the display surface may wrap around.
- 1: Clipping at display surface boundaries.
- 2: Clipping at viewport boundaries.

**larr(11):** Default background colour. 0 = black (dark), 1 = white (light)

Not relevant for devices with static or dynamic colour table (see **larr(19)**).

**larr(12):** Default foreground (drawing) colour. 0 = black (dark), 1 = white (light).

Not relevant for devices with static or dynamic colour table.

**larr(13):** Hardware text capability. Returned as a sum of size and rotation.

Size:

- 0: No hardware text available.
- 1: One size only.
- 2: Limited number of different sizes.
- 3: Any size available.

Rotation:

- 10: Horizontal text only.
- 20: 4 directions (rotation angle 0, 90, 180 and 270 degrees).
- 30: Any rotation available.

**larr(14):** Number of hardware fonts available.

**larr(15):** Hardware circle capability.

- 0: No hardware circles available.
- 1: 2D circles.
- 2: 3D circles.

**larr(16):** Device type.

- 0: Vector terminal.
- 1: Vector plotter.
- 2: Raster terminal.
- 3: Raster plotter.
- 4: Device independent file format.
- 5: Other, e.g. pseudo.

**larr(17):** Colour / monochrome device.

0 = colour, 1 = monochrome.

**larr(18):** Maximum colour index (=length of colour table excluding index 0).

See also **larr(27)**.

**larr(19):** Colour table type (see **page 11-1** for details).

- 0: Fixed (no modification possible).
- 1: Static (may be modified, will not affect previously drawn primitives).
- 2: Dynamic. See also **larr(27)**.

**larr(20):** Default polygon texture type (see **page 12-5**).

- 0: No texture available.
- 1: Hatch.
- 2: Pattern.

**larr(21):** Number of user definable patterns available (see **page 12-7**).

**larr(22):** Number of user definable hatch styles available (see **page 12-8**).

**larr(23):** Pixel arrays capability (see **page 12-12**).

- 0: Pixel arrays not available.
- 1: Pixel arrays may be drawn.
- 2: Pixel arrays may be drawn and inquired.

**larr(24):** Number of picture segment priorities supported (see **page 17-3**).

**larr(25):** Automatic update when changing segment priority.

- 0: No, **REDRAW** (**page 16-4**) or **RDRDWI** (**page 21-10**) must be used.
- 1: Yes.

**larr(26):** Number of available device windows (**Chapter 21**).

**larr(27):** For dynamic colour tables, the highest colour index that may be dynamically changed. Will be equal to or lower than **larr(18)**.

If no colour indices may be dynamically changed (will be the case if all colours have been allocated by other applications), a value of -1 will be returned.

If zero is returned, all entries may be changed.

### Installation dependent parameters for polygon drawing

**CALL DATPAR (lpolsz, lcros, lpatsz)**

Installation dependant parameters are described in **Appendix A**.

**lpolsz:** Maximum number of polygon vertices. Parameter **MPOLSZ**.

**lcros:** Maximum number of crossings between an arbitrary line and a polygon, used for software hatching and pattern filling. Parameter **MPCROS**.

**lpatsz:** Maximum number of colour cells in user defined patterns. Parameter **MRPSIZ**.

### Fortran unit numbers used by GPGS-F

**CALL DATFNU (lfontu, lerru)**

**lfontu:** Fortran unit number used when reading font data. Either installation dependent parameter **MFUNIT**, or the value set by **SETFNU** (page 7-9).

**lerru:** Fortran unit number used when reading error information. Either installation dependent parameter **MEUNIT**, or the value set by **SETFNU** (page 24-4).

### HLHS module limits and utilization

**CALL DATHID (larr(1), Lthi)**

The number of values to be returned must be specified by **Lthi**.

Installation dependant parameters are described in **Appendix A**.

**larr(1):** Maximum number of polygons that may be stored by the module. Installation dependent parameter **MXUPOL**.

**larr(2):** Maximum number of vertices of a polygon to be stored. Parameter **MXUPTS**.

**larr(3):** Expected average number of vertices of polygons to be stored. The total number of vertices that may be stored is computed as this value multiplied with **larr(1)**. Parameter **MAUPTS**.

**larr(4):** Maximum number of lines that may be stored. Parameter **MXULIN**.

**larr(5):** Number of polygons stored so far.

**larr(6):** Number of vertices stored so far.

**larr(7):** Number of lines stored so far.

### Primary segment storage

**CALL DATBUF (larr(1), Lthi)**

The number of values to be returned must be specified by **Lthi**.

The values refer to the current buffer, selected by *NITBUF* or *SELBUF* (page 14-3)

**larr(1):** Free space in buffer, in integer words.

**larr(2):** Minimum free space in buffer since *DATBUF* was last called. If no segments have been deleted from the buffer, this will be the same as **larr(1)**. If segments *have* been deleted, the value gives the free space as it was before garbage collection.

**larr(3):** Total space in buffer, as specified by *NITBUF*.

### Picture library

**CALL DATLIB (larr(1), Lthi)**

The number of values to be returned must be specified by **Lthi**.

The values refer to the current library, selected by *NITLIB* or *SELLIB* (page 14-5).

**larr(1):** Unit number of current picture library.

**larr(2):** Number of picture segments in library.

**larr(3..):** Identifiers of stored segments, in random order.

# Chapter 24

## Errors and Messages

---

GPGS-F will check the input arguments to see if they fit the definition of the subroutine called, and the restrictions defined for some arguments (e.g. linetype codes must not be negative).

In addition, the system will check if the subroutine in question may be called at that time (e.g. *ENDPIC* may not be called if there is no segment open).

Note that GPGS-F will not detect if the number of arguments passed to a subroutine is incorrect. Passing too few arguments will in most cases make the Fortran runtime system abort the program.

### 24.1 GPGS-F Error Vector

If an error condition is detected, GPGS-F will build an integer array, called the **GPGS-F Error Vector**, containing a description of the error condition. This Error Vector is then passed to a routine for printing the error, or to a routine supplied by the application program (see [page 24-4](#)).

**Table 24.1** *GPGS-F Error Vector.*

Element number	Description
1	Error number. Listed in <b>Appendix I</b> .
2	Subroutine code. Identifier of the GPGS-F routine causing the error. This is an integer number, listed in <b>Appendix G</b> .
3	Argument number causing the error, if relevant. Used only with integer arguments. Arguments are counted left to right in the argument list.
4	Error severity code. See <b>Table 24.2</b>
5	Number of GPGS-F subroutine calls made since the application program was started.
6	Bad value. The value of the argument, given by element 3, that caused the error.
7	GPGS-F version number (see <a href="#">page 1-6</a> ).

**Table 24.2** *GPGS-F severity codes.*

Severity code	Description.
1	Warning. Program execution will continue, no action necessary. Example: Trying to open a library that is already open.
2	Recoverable error. Program execution will continue after some error dependent recovery action has been taken. Example: Trying to open a picture segment when a segment is already open. In this case, the recovery action taken is to close the previous segment.
3	Severe error. Because of an illegal argument value, GPGS-F can not perform the requested action, unless the bad value is changed by an application supplied error handling routine (see <b>page 24-4</b> ).
4	Fatal error. Further program execution is not possible.

## 24.2 Default Error Handling

The default error handling performed by GPGS-F, is to print an error message before taking the action indicated by the severity code. If this is 1 or 2, execution will continue, if it is 3 or 4, the program will be terminated.

The format of the error message may be chosen by

**CALL MSGLEV (llevel)**

where **llevel** = 0 selects **short** format, **llevel** = 1 selects **extended** format.

When short format is used, the values of the Error Vector is just printed, and the user have to look up the actual error message (**Appendix I**) and the name of the subroutine causing the error (**Appendix G**) in this manual.

When extended format is selected, GPGS-F will instead print the error message text, and the name of the subroutine causing the error. This information is read from text files supplied with the GPGS-F system. If GPGS-F is not able to open or read these files, the user will be notified, and short format will be used.



### Example 24.1 Error messages.

Short format	Extended format
<pre> **GPGS ERROR          7 * SEVERITY             2 * CALL NO.            6 * ROUTINE NO.         60 * ARGUMENT NO.        0 * VALUE               1 **GPGS VERSION 9503 </pre>	<pre> ***** *** GPGS-F Error message *** *****  ** Picture segment open.    The previous picture segment    should be closed.  * Routine giving error : BGNPIC * Severity code       :      2 * Call no             :      6 * GPGS-F version      :    9503 </pre>
<pre> **GPGS ERROR          11 * SEVERITY             3 * CALL NO.            6 * ROUTINE NO.         106 * ARGUMENT NO.        3 * VALUE              -1 **GPGS VERSION 9503 </pre>	<pre> ***** *** GPGS-F Error message *** *****  ** Illegal value of argument.    Argument outside allowable range.  * Routine giving error :    LINE * Severity code       :      3 * Call no             :      6 * Argument no.        :      3 * Value of argument   :     -1 * GPGS-F version      :    9503 </pre>

Note that when short format is used, the argument number and value is always printed, even when this information is not relevant with the given error.

## 24.3 Error File

By default, error messages are printed on the standard output device, i.e. the terminal the program is run from. When extended message format is selected, it is however possible to get error messages written to a file instead of, or in addition to, the standard output.

Such an error file must be a sequential file opened by the application program. The Fortran unit number of the file is passed to GPGS-F by

CALL ERFILE (Ifile)

If **Ifile** is positive, error messages are written to that file *in addition to* standard output. If **Ifile** is negative, messages are written to unit (**-Ifile**) *instead of* standard output.

By giving 0, the default condition is reset, i.e. error messages are written to standard output only.

The information written by the extended message format is fetched from two text files. When these are opened, GPGS-F will by default use the Fortran unit number set by the installation dependent parameter **MEUNIT** (see **Appendix A**).

If this number is used by the application for other purposes, GPGS-F may be told to use a different unit number by

**CALL SETFNU (lfontu, lerru)**

where **lerru** is the new Fortran unit number to use when opening the error files. **lfontu** is described on **page 7-9**.

If -1 is given for **lerru** and/or **lfontu**, the default value is reset, while 0 means no change.

## 24.4 Application Supplied Error Routine

Using the default GPGS-F error handling method informs the user of the errors that are detected, and if severity code is 3 or 4, terminates the program.

In some cases, the application should perform some shutdown actions before leaving the program (closing databases, resetting terminal conditions, etc.), and getting the program aborted by GPGS-F will make this a problem.

To avoid such problems, GPGS-F allows the user to write his own error handling routine, replacing the default routine from the GPGS-F library.

When GPGS-F detects an error, the system will call a routine called *ERROUT*, with the Error Vector (see **page 24-1**) as argument. This routine is defined as

**SUBROUTINE ERROUT (larr)**  
**DIMENSION IARR(7)**

What the application programmer has to do, is then just to write his own routine defined the same way, and make sure this is linked/loaded instead of the GPGS-F routine.

The application supplied *ERROUT* routine may attempt to recover from an error condition in cases where an argument is given an incorrect value. This is done by changing the argument value (**larr(6)**) to a legal value, and the severity code (**larr(4)**) to a value less than 3.

If the error condition reported indicates that it is impossible to recover, the routine should perform the necessary application dependant shutdown action and exit. Note that *ERROUT* **must not** call any other GPGS-F routine, except *LOGERR* described next.

The application supplied error routine may print the error message by

**CALL LOGERR (larr(1))**

where **larr** is the Error Vector. When printing the message, *LOGERR* will use the values set by *MSGLEV* and *ERFILE*.

### 24.4.1 Closing Down GPGS-F

The main reason why an application selects to provide its own error routine, is to perform the necessary application dependent shutdown procedures.

However, if a program is aborted due to some GPGS-F error condition, GPGS-F itself may be left in an inconsistent state, e.g. plotter files are not terminated correctly as devices are not released, picture library files are not correctly updated and closed, etc.

To perform these kinds of shutdown actions,

**CALL GPGSOF**

should be included in any application supplied error routine.

Note that the default *ERROUT* routine supplied by GPGS-F does *not* call *GPGSOF*.

#### **Example 24.2 Application supplied error routine.**

```

SUBROUTINE ERROUT(IVEC)
  DIMENSION IVEC(7)
  C
  C Print error message by using GPGS-F routine.
  C
  CALL LOGERR(IVEC)
  C
  C If severity 1 or 2, no action necessary as the program
  C will continue.
  C
  IF (IVEC(4) .GT. 2) THEN
  C
  C The next test is not straightforward. Most obvious error
  C condition making it possible to recover, is error 11,
  C 'Illegal value of argument'
  C
    IF (possible-to-recover) THEN
      PRINT *, ' Give new argument value:'
      READ *, IVEC(6)
      IVEC(4)=1
    ELSE
      CALL Users-shutdown-routine
      CALL GPGSOF
    ENDIF
  ENDIF
  RETURN
END
```



# Appendix A

## Installation Dependent Parameters

---

The information given in this appendix is mainly aimed at the site responsables for GPGS-F. All GPGS-F users should however know which limitations of GPGS-F are installation dependent, and which are fixed.

The following is a list of all installation dependent parameters, with a short description on what they are used for. For some of the parameters it is possible to request the current value. If so, the routine to be used is mentioned.

As the GPGS-F system is delivered, the parameters are set to a default value, which is mentioned in the list below. Some parameters should not be set to a value lower than the default, i.e. the default value is the lower limit.

The parameters are found in the include file **IDPARA**.

### MDLANG

Gives the default character language to be used for GPGS-F text. The value must be in the range 1 to 11 according to the description of routine *CLANG* (page 7-10).

**Examples:**      1 = ANSI ASCII      2 = Norwegian      3 = Swedish

**Default value: 2 (Norwegian)**

### MPOLSZ

Maximum number of vertices (corners) in polygons. Current value may be requested by *DATPAR* (page 23-7).

**Default value: 100 (lower limit)**

### MPCROS

Maximum number of crossings between an arbitrary straight line and a polygon, thus limiting the complexity of polygons. Used for software hatching and pattern filling. Current value may be requested by *DATPAR* (page 23-7).

**Default value: 20 (lower limit)**

### MRPSIZ

Maximum size of user definable pattern. By size is meant the total number of cells in the pattern. Current value may be requested by *DATPAR* (page 23-7).

**Default value: 256 (lower limit)**

## MFUNIT

Default Fortran unit number used when opening **MFILE** (see **page A-3**). The actual value to use may be set by the application program using the *SETFNU* routine (**page 7-9**). Current value may be requested by *DATFNU* (**page 23-7**).

**Default value: 10**

## MEUNIT

Default Fortran unit number used when opening **ERRFIL** and **RTFILE** (see **page A-3**). The same number is used as the two files are never open at the same time. The actual value to use may be set by the application program using the *SETFNU* (**page 24-4**) routine. Current value may be requested by *DATFNU* (**page 23-7**).

**Default value: 12**

## MXUPOL

HLHS (Hidden Lines / Hidden Surfaces) parameter. Gives the maximum number of polygons that may be stored in the module. Current value may be requested by *DATHID* (**page 23-7**).

**Default value: 1000 (lower limit)**

## MXUPTS

HLHS parameter. Gives the maximum number of vertices of a polygon stored in the module. Must not be greater than **MPOLSZ** (see **page A-1**). Current value may be requested by *DATHID* (**page 23-7**).

**Default value: 100 (lower limit)**

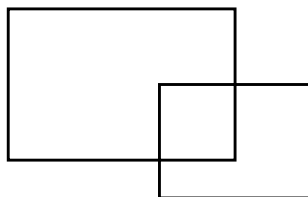
## MAUPTS

HLHS parameter. Gives the average number of vertices of polygons stored in the module. Used to set a maximum number of vertices that may be stored (**MAUPTS\*MXUPOL**). Current value may be requested by *DATHID* (**page 23-7**).

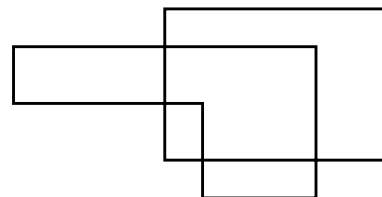
**Default value: 10 (lower limit)**

## MXCROS

HLHS parameter. Gives the allowed maximum number of crossings between two polygons, i.e. the number of edge versus edge crossings, as shown below.



Number of crossings: 2



Number of crossings: 4

**Default value: 40 (lower limit)**

## MXULIN

HLHS parameter. Gives the maximum number of lines that may be stored in the module. Current value may be requested by *DATHID* (page 23-7).

**Default value: 1000 (lower limit)**

## Name of files used by GPGS-F

These parameters are found in the include file **FNAMES**.

*These parameters should be set once for all as the system is installed. Changes at some later time will require a re-linking of all application programs using GPGS-F*

**NOTE!** With the Unix-version, the 3 following parameters must not be changed in the include file. Instead, the names are set up in the supplied *makefile*.

## MFILE

Name of file containing description on GPGS-F software characters. The name will have two parts, the first one giving a user, directory or equivalent, and the second part being the actual filename. The second part should not be changed.

**Examples:**        **VAX/VMS :**        **GPGS:FONTSxxx.DAT**  
                     **ND-500(0) :**        **(GPGS)GPGS-FONTS-xxx:DATA**

where xxx will be changed by the GPGS-F supplier when changes has been done.

**Default value: Machine dependent**

## ERRFIL

Name of file containing GPGS-F error messages used for extended error message format. Same rules apply as for **MFILE**.

## RTFILE

Name of file containing GPGS-F routine numbers used for extended error message format. Same rules apply as for **MFILE**.

## Changing parameters at a later time

If any of the parameters has to be changed after the GPGS-F system is installed, (parts of) the system has to be recompiled.

If a parameter marked as HLHS parameter is changed, only the HLHS module needs recompilation. If any of the others are changed, the complete system must be recompiled, including the HLHS module. Drivers need however not be recompiled.

**With the Unix version, the supplied *makefile* will ensure that the necessary modules are updated after changing any of the parameters.**





# Appendix B

## Software Character Fonts

---

The following pages show all defined characters of all GPGS-F software fonts. If a text string contains a character that is not defined, a space will be output.

The characters are drawn with default height/width ratio.

### Font 0: Default GPGS-F Font

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
p	q	r	s	t	u	v	w	x	y	z	{		}	~	□
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
	ı	ø	£	¤	¥	ı	§	°	©	®	«	¬		®	—
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
◊	±	²	³	´	µ		◊	ˆ	¹	º	»	¼	½	¾	¿
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
đ	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

### Font 1: Simplex Roman

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	!	"	#	\$	%	&	'	(	)	*	+	,	—	.	/
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
p	q	r	s	t	u	v	w	x	y	z	{		}	~	
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
	ı	¢	£	¤	¥	¦	§	¨							—
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
°	±			´	μ		·	¸							¿
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý		ß
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý		ÿ

### Font 2: Complex Roman

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	!	"	#	\$	%	&	'	(	)	*	+	,	—	.	/
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
p	q	r	s	t	u	v	w	x	y	z	{		}	~	
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
	ı	¢	£	¤	¥	¦	§	¨							—
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
°	±			´	μ		·	,							¿
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý		ß
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý		ÿ

### Font 3: Complex Italic

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
p	q	r	s	t	u	v	w	x	y	z	{		}	~	
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
	i	ø	£	¤	¥	!	§	¨							-
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
°	±			´	μ		·	,							¸
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý		ß
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý		ÿ

### Font 4: Duplex Roman

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
p	q	r	s	t	u	v	w	x	y	z	{		}	~	
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
	ı	¢	£	¤	¥	¦	§	¨							—
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
°	±			´	μ		·	,							¿
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý		ß
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý		ÿ

### Font 5: Simplex Greek

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	8	9						
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
	A	B	Γ	Δ	E	Φ	X	H	I	Ψ	K	Λ	M	N	O
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
Π	Θ	P	Σ	T	Υ	N	Ω	Ξ	Υ	Z					
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
	α	β	γ	δ	ε	φ	χ	η	ι	ψ	κ	λ	μ	ν	ο
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
π	ϑ	ρ	σ	τ	υ	ν	ω	ξ	υ	ζ					

### Font 6: Complex Greek and Cyrillic

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	!	"	#	\$	%	&	'	(	)	*	+	,	—	.	/
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
@	A	B	Γ	Δ	E	Φ	X	H	I	Ψ	K	Λ	M	N	O
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
Π	Θ	P	Σ	T	Υ	N	Ω	Ξ	Υ	Z	[	\	]	^	_
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
`	α	β	γ	δ	ε	φ	χ	η	ι	ψ	κ	λ	μ	ν	ο
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
π	ϑ	ρ	σ	τ	υ	ν	ω	ξ	υ	ζ	{		}	~	
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
Ë	ë														



**Font 7: Mathematical**

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
					⌘	⌘		<	>	×	±		≠	·	÷
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
φ	†	‡						∞				≤	→	≥	
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
@	∞		§	∂	€				∫	ℳ		}	[	]	°
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
Π	θ	}	Σ	~		√		×			[		]		
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
	∞		§	∂	€				∫	ℳ		}	[	]	°
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
Π	θ	}	Σ	~		√		×			}		}	~	



# Appendix C

## Additional GPGS-F Products

### C.1 MICRO-GPGS-F

MICRO-GPGS-F is a 2-dimensional subset of GPGS-F, originally developed for use on computers with limited addressing space. It is a fact that many GPGS-F application programs use just a very small part of GPGS-F, and these do not need all the advanced possibilities of the system. Even if there is no problem with limited addressing space, use of MICRO-GPGS-F instead of GPGS-F may speed up execution time because of its simplified internal structure.

All routines of MICRO-GPGS-F are identical to the corresponding GPGS-F routines, and this User's Guide may hence be used as a User's Guide for MICRO-GPGS-F as well, by just skipping routines that are not part of MICRO-GPGS-F.

An application using only the routines that are part of MICRO-GPGS-F, will give the same result whether GPGS-F or MICRO-GPGS-F is used, with one exception. If *SOFCHA* is not called to select character quality, MICRO-GPGS-F will by default use *hardware* characters, while GPGS-F will use *software*.

The interface between MICRO-GPGS-F and the drivers is identical to the interface between GPGS-F and the drivers. This means that all GPGS-F drivers may also be used with MICRO-GPGS-F.

**Table C.1**    *MICRO-GPGS-F subroutines.*

BGNPIC	BGNTRN	CESCAP	CFONT	CHARA	CHARC
CHARE	CHARF	CHARI	CHARS	CIRAPR	CIRC
CIRCR	CIRD	CIRDR	CJUST	CLANG	CLICTL
CLRDEV	COLOUR	COMP	COTHL	COTHSV	COTIND
COTRGB	CROTA	CROTAD	CSHEA	CSIZEL	CSIZES
DATATR	DATCHR	DATCIR	DATCLI	DATCXC	DATDEV
DATDNO	DATMAR	DATPNO	DATPOS	DATUSR	DATVP
DATWIN	ECHCTL	ECHTOL	ECHTXC	ECHVP	ENDPIC
ENDTRN	GPGS	IDEN	INTENS	INWAIT	LINE
LINER	MARKER	MODTRN	MSIZE	NDCWIN	NITDEV
NITOPT	PITYP	POLY	POLYR	REATOL	REQBUT
REQLOC	REQTXC	REQVAL	RLSDEV	ROTA	ROTAD
SAVTRN	SCAL	SELDEV	SHEA	SOFCHA	SOFCIR
TRAN	UPDAT	USRWIN	VPORT	WINDW	WINNDC
WINUSR	WRITOL	XLAT			

## C.1.1 Main Limitations Compared to GPGS-F

By comparing **Table C.1** to the list of GPGS-F routines, the limitations of MICRO-GPGS-F will be found. The following list gives a summary of these limitations.

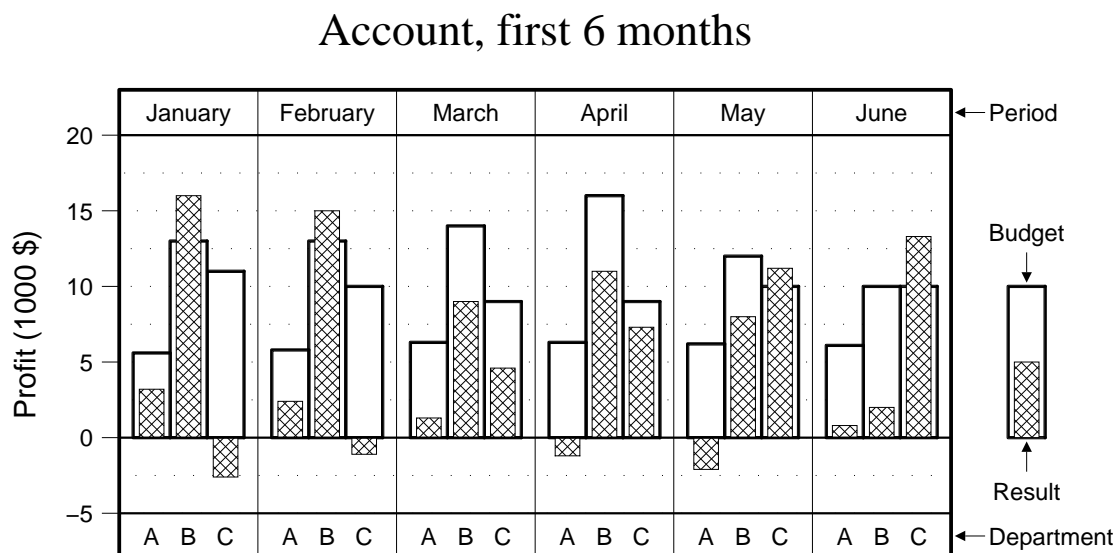
- No 3 dimensional routines.
- No picture segment handling, i.e. no retained or pseudo picture segments may be used. Following from this, use of background device is not available.
- No hidden lines/hidden surfaces removal.
- No software simulation of patterns or hatch styles. Only polygon types 1,2 and >127 may be used. Pattern or hatch is selected by *PITYP*.
- Pixel arrays are not available.
- Only font 0 (standard GPGS-F) available for software characters. The *CFONT* routine is however available for selecting hardware fonts.
- No user definable linetypes or line representations.
- No polyline routines (*TAB.*) or curve routines (*CURV.*).
- Reduced interaction facilities.
- The routines for handling multi window devices are not included.

## C.2 GRAPHISTO

GRAPHISTO is a subroutine package for presentation of 2 dimensional data (business graphics type) as curves, bar charts or pie charts.

The package contains high level self-scaling routines, making it possible to create complete plots by using very few subroutine calls. In addition, a large number of routines are available for defining the picture in more detail.

**Figure C.1** *GRAPHISTO example plot.*

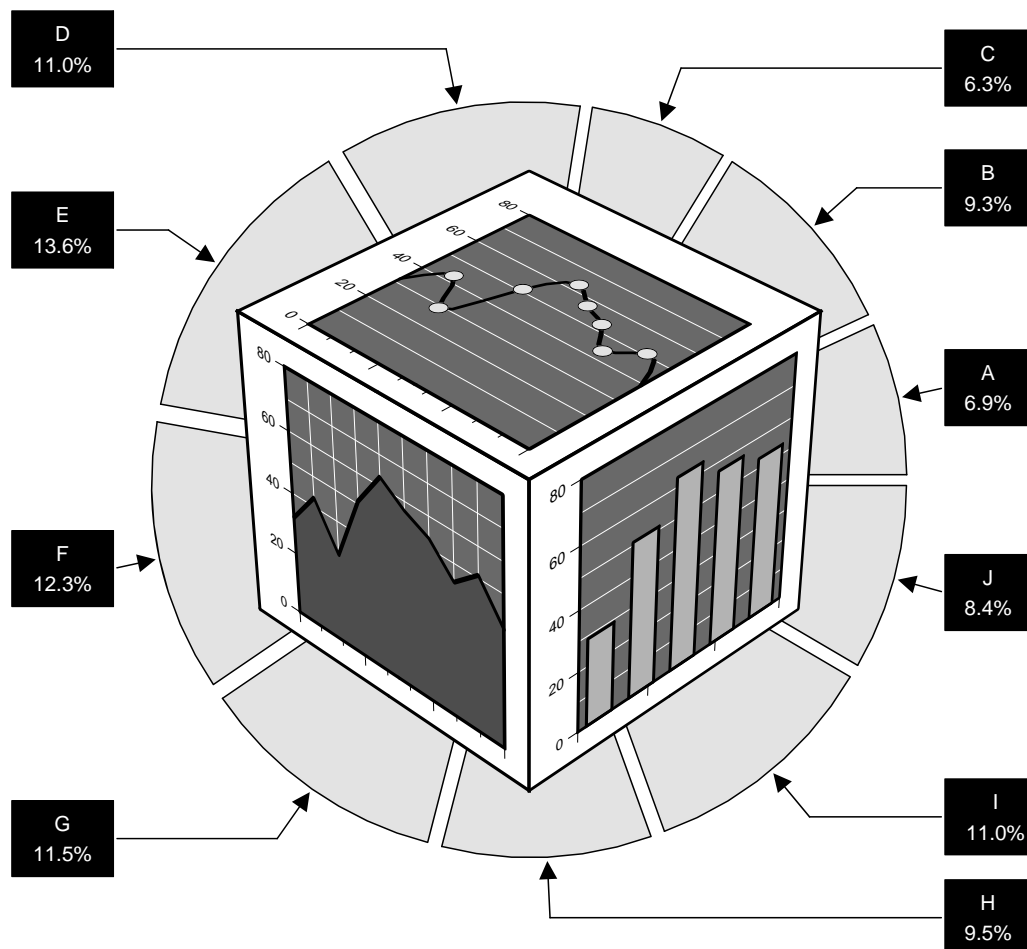


GRAPHISTO is interfaced to GPGS-F, i.e. GPGS-F routines are used for all basic operations, such as creating the graphics and defining temporary transformations to model the different parts of a complete plot.

The interface between GRAPHISTO and GPGS-F is defined so that all GRAPHISTO routines, after using GPGS-F, will reset any internal GPGS-F state. Because of this, an application may combine GRAPHISTO routine calls with any GPGS-F routine, e.g. the modelling transformation routines, allowing plots like the one below to be created.

GRAPHISTO does not contain routines for performing operations that are already available in GPGS-F. This means that some GPGS-F *must* be included in GRAPHISTO applications, such as all device handling routines.

**Figure C.2** *GRAPHISTO plots transformed by GPGS-F.*



The GRAPHISTO subroutine package is described in the **GRAPHISTO User's Manual**, which may be obtained from the GPGS-F version responsible.

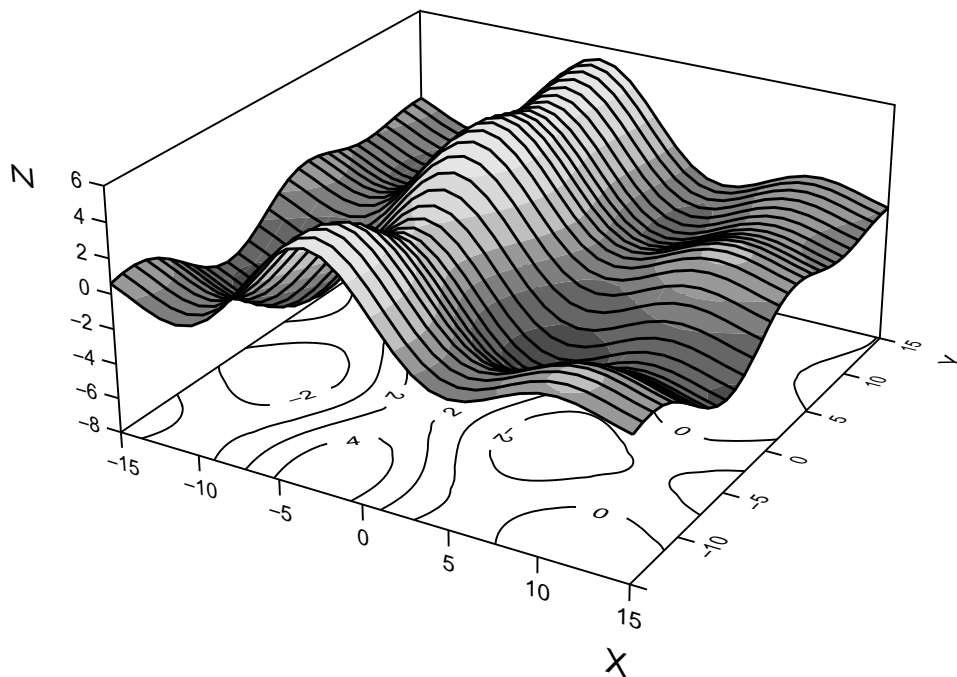
## C.3 SURRENDER

SURRENDER (short for **SUR**face **REN**DERing) is a subroutine package for presentation of 3 dimensional data ( $z=f(x,y)$ ), either as a 3D perspective plot or as a 2D contour plot.

Basic plots are made by using a single subroutine call, but additional routines are available for adding various plot attributes or setting options for the basic plotting routines. SURRENDER contains its own viewing module for easy specification of view direction for perspective plots.

SURRENDER is interfaced to both GRAPHISTO and GPGS-F. GRAPHISTO routines are used for adding axes and axes annotations, while GPGS-F is used for all basic drawing operations and transformation purposes.

**Figure C.3** *SURRENDER example plot.*



The SURRENDER subroutine package is described in the **SURRENDER User's Manual**, which may be obtained from the GPGS-F version responsible.

# Appendix D

## Machine Dependencies

---

All GPGS-F subroutines are machine independent. There are however some aspects concerning the use of the system that is dependent on the actual machine being used.

The most obvious machine dependency is how to link/load an application program with GPGS-F. As this in addition might be installation dependent, no description is given in this manual. Full details on this is obtained from the site responsible of GPGS-F.

A second machine dependency is how to specify the disc file or communication channel to use for GPGS-F input/output. This is described in the section below.

### D.1 File / Communication Channel Numbers

As explained in **Chapter 1**, GPGS-F will by default write its output to the standard output device, i.e. normally the terminal the program is run from. By using the *DEVOPT* routine (see **page 1-4**), the output may be redirected to another communication line, or to a disc file. This *must* be used if plotters are connected to separate channels, and *may* be used by drivers operating in **Write-Only Mode** to store plotting commands on disc files.

The file/channel number is used as follows on different computers:

#### ND 100 / 500(0) computers

The file number given may be a SINTRAN Logical Device Number or a Fortran file number. GPGS-F will first check if the number given is used as a Fortran file number. If so, that file is used for output. If not, the number is expected to be a SINTRAN LDN. If files to be used for GPGS-F input/output are opened by Fortran, the file access must be specified as 'RW', or 'W' for drivers operating in **Write-Only Mode**.

#### VAX/ VMS computers

GPGS-F will use logical file name **FOR0nn** for input/output, where **nn** is the number given to *DEVOPT*. Thus, the actual file or terminal line must be opened by the application program, or assigned to the logical file name before running the application. If no Fortran open or assignment is used, the physical file **FOR0nn.DAT** will be used.

### **Unix computers**

With the Unix version of GPGS-F, the input/output actions are performed by C language functions. The file number is thus a C language unit number, and the file must be opened by a C function.

A GPGS-F utility routine is available for this purpose. It is invoked by

**lunit = NITTTY (Idev, Filename)**

where **Idev** is the GPGS-F device number and **Filename** is the name of the file or communication line to be used (examples: '/dev/tty2', 'plotfile.dat'). The routine will return the file number through **lunit**. If this is negative, the file could not be opened.

A file/channel opened by *NITTTY* must be closed by a second GPGS-F utility routine

**CALL CLSTTY (lunit)**

where **lunit** is the file number returned by *NITTTY*. Do not use Fortran CLOSE statements to close files opened by *NITTTY*.

### **Other computers**

Information on this aspect for other computers may be obtained from the GPGS-F version responsible for the given version.



# Appendix E

## Device Driver Descriptions

---

This appendix contains a short description of the features provided by each GPGS-F device driver. Additional information is supplied with some drivers as they are delivered. This is mentioned in the description of the drivers it applies to.

There may be some new drivers that are not included in this appendix, if developed after the last revision of the manual. The site responsible for GPGS-F will have an updated list of drivers, and complete driver description of all drivers available on the computer he/she is responsible for.

### The following information is given for each driver:

#### 1 Heading:

Contains the GPGS-F driver number to be used with *NITDEV*, and a four letter name used when loading/linking the driver. The actual filename is dependent on what computer is used. The site responsible will know the details on this.

Note that in some cases the same name is given with more than one driver number. This means that the same driver is to be used by these driver numbers, and the driver will function slightly different depending on the number actually given by *NITDEV*.

#### 2 Description:

A more detailed description of the graphic device, and other device numbers using the same driver, if any.

If the driver is interfaced to other low-level software, the name of this is given. This software is normally obtained from the supplier of the graphic device.

#### 3 Options recognized by the driver:

Explains the meaning of device dependent options supplied through *DEVOPT*. If not otherwise specified, the options are integer options. Default value is 0 for all integer options.

#### 4 Device data:

- Maximum viewport size in X and Y direction (NDC). As returned by 2 first real elements from *DATDEV*.
- Default viewport size in meters. As returned by 4th real element from *DATDEV*.
- Spot size in fraction of default viewport. As returned by 5th and 7th real element from *DATDEV*, i.e. inverse of the resolution. If equal resolution in X and Y just one value is returned.

The device data given for 'non-physical devices' (like **Pseudo** and **File**) are in most cases the default values returned from *DATDEV*, and have no physical meaning.

## **5 Interactive facilities:**

Lists the interactive tools available, and explains any special echo type and extra data available. If not especially mentioned, sample and event mode input are not available.

For devices supporting pick input, the length of the namestack is given. Note that this includes the segment identifier, i.e. the allowable number of names is one less than the number given.

ESCAPE tools (tool numbers 900-999) available are also listed.

## **6 Retained segment facilities:**

- Retained segments capability. States if retained segments are available, and how these are stored (in the device, in the driver or in the GPGS-F buffer/pick simulation module).
- Selective erase. If not available means that the complete image is redrawn when the picture is changed.
- Image transformations available.
- Number of segment priorities available.

## **7 Raster facilities:**

- Length (number of foreground colours) and *type* of colour table.  
*type* may be one of:
  - Fixed:* Not possible to change colour table.
  - Static:* Changing the colour table will not affect previously drawn primitives.
  - Dynamic:* Changing the colour table will change the colour of previously drawn primitives.
- Polygon fill with solid colour capability.
- Default texture type (used if *PITYP(0)* is called). If not given, hatch is default.
- Number of hardware/driver predefined hatch styles.
- Number of hardware/driver predefined patterns.
- Number of user definable hatch styles that may be stored in the hardware/driver.
- Number of user definable patterns that may be stored in the hardware/driver.
- Pixel array drawing and readback capability.

Note that software hatching is available with all devices. Software patterned fill is available if pixel array drawing is possible.

## **8 Special features:**

Lists various capabilities of the device hardware or driver, such as number of linetypes, text sizes, text directions and whether shearing is available, circle arcs capability, and if markers are generated by hardware or driver software.

## **9 Miscellaneous:**

Gives additional device specific information of interest.

The drivers are described in device number sequence, starting with lowest number. If you know the driver name and want to find the driver number, consult the table on the next page.

**Table E.1** *Alphabetic List of Device Drivers.*

Driver	
Name	Number(s)
ALTK	41
APOL	33
C600	40
CALC	13
CC81	19, 86
CCOL	88
CNA2	82
CPCI	81
DUMY	1
FILE	8
GTEC	92
HP20	5
HP21	11
HP48	51
HPGL	4, 9, 18, 80, 83
ICGM	35
KING	14
LASR	87
META	3
NDLA	91
PSCR	90
PSEU	0
RASP	6
REGI	57, 64, 74
RUBY	71

Driver	
Name	Number(s)
SIMU	none (page E-4)
SNAP	2
TA10	84
TD22	55
TDGO	63
TECH	73
TPAZ	68, 70
TX05	62
TX14	20, 21
TX25	54
TX27	53
TX29	67
TX42	58
TX43	59
TX44	22
TX45	89
TX62	12
TX63	10
VCOL	85
VERS	15
WW32	65, 66
WWGM	69
XWDW	72

## No. — 'SIMU' Buffer/pick simulation module

### Description:

The buffer/pick simulation module, or **SIMU** driver, is no real driver, but a set of utility routines for simulating picture segment handling and pick input on terminals with no local segment storage.

The module is normally stored on a separate file, and has to be linked/loaded with programs using retained picture segments, if the description of the driver states: *'Retained segments stored in GPGS-F buffer/pick simulation module'*.

Linking/loading the module when it is actually not needed will not give any error, but the processing time will in some cases increase.

## No. 0 'PSEU' Pseudo Segment Generator

### Description:

Driver for generating pseudo picture segments in GPGS-F buffers or in picture libraries.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.0 \times 1.0$

### Default viewport size:

1.0 meter

### Spot size (fraction of default viewport):

Not relevant.

### Interactive tools:

None

### Retained segments:

None

### Raster facilities:

All commands are stored in the pseudo segments.

Pixel arrays and software patterned polygons should not be used with this driver, as these routines must know the physical resolution of the device in use.

### Special features:

All primitives and primitive attributes are stored in the pseudo segments without checking.

### Miscellaneous:

## No. 1 'DUMMY' Dummy driver

### Description:

No-operation driver for debugging purposes.

Specifying this driver makes it possible to execute a GPGS-F program without generating any output.

This driver is always available, i.e. there is no object file that has to be linked/loaded.

## No. 2 'SNAP' DIPC dump

### Description:

Driver for writing GPGS-F Device Independent Picture Code (the code passed between GPGS-F and the device drivers) to standard output or to disc file.

This driver is mainly to be used for debugging purposes by GPGS-F version responsables.

### Device dependent options:

Option 3 : Retained segments storage switch, for checking DIPC for different kinds of devices (see description of *DATDEV* on page 23-4).

0 = no buffer usage (can not store segments)

2 = retained segments stored in driver.

4 = retained segments stored in GPGS-F simulation module.

### Maximum viewport size (NDC):

1.0 × 1.0

### Default viewport size:

1.0 meter

### Spot size (fraction of default viewport):

1/512

### Interactive tools:

None

### Retained segments:

As specified by device dependent option 3.

### Raster facilities:

All commands will be written to the output file.

### Special features:

All primitives and primitive attributes are written to output file without checking.

### Miscellaneous:

The DIPC commands are written using Fortran formatted WRITE.

## **No. 3 'META' GPGS-F ASCII metafile generator**

### **Description:**

Driver for writing device independent picture code on formatted sequential file. The file will contain ASCII characters in the range 32-126 only, and may thus easily be moved between different computers (compare to driver 8).

### **Device dependent options:**

Option 4 : Resolution in X direction (number of pixels in 1.0 NDC). Default: 512.

Option 5 : Resolution in Y direction. Default: same as in X direction.

**NOTE!** options 4 and 5 are relevant only when pixel related GPGS-F routines are used. When later displaying the file contents, correct output will be generated only on devices with the resolution given by these options.

Text option 1 : Output file name.

### **Maximum viewport size (NDC):**

1.0 × 1.0

### **Default viewport size:**

1.0 meter

### **Spot size (fraction of default viewport):**

As specified by options 4 and 5.

### **Interactive tools:**

None

### **Retained segments:**

None

### **Raster facilities:**

All commands will be written to the output file. Pixel arrays and software patterned polygons will be generated based on the resolution specified by options 4 and 5.

### **Special features:**

All primitives and primitive attributes are written to output file without checking.

### **Miscellaneous:**

If not given by text option 1, the driver will ask for the output file name.

## **METASHOW: Program for displaying contents of file**

The **METASHOW** program will read a file generated by the **META** driver and display the contents on a device specified by the user.

The program will ask for the name of the file to read, GPGS-F device number to use, options to pass to the driver (through *DEVOPT*), and size of the resulting drawing.

With the Unix version, specifying option 1 to *DEVOPT* > 0 will make the program ask for the name of a file to use for driver output (this file is opened by the **METASHOW** program by using the GPGS-F routine *NITTTY*).



## No. 4 'HPGL' Hewlett-Packard 7475 plotter

### Description:

Driver for Hewlett-Packard 7475 plotter.

### Device dependent options:

- Option 3 : = 1 → The driver will operate in '*Write-Only Mode*'.  
Other values give '*Read/Write Mode*'.
- Option 4 : Paper size when '*Write-Only Mode*'.  
3 = A3 paper, 4 = A4 paper.
- Option 6 : = 1 → Default linepattern length will be 2% of distance between P1 and P2 (backwards compatibility).
- Option 7 : = 1 → Plot is rotated 90 degrees.

### Maximum viewport size (NDC):

Write-Only Mode / A3 paper :  $1.463 \times 1.0$   
Write-Only Mode / A4 paper :  $1.429 \times 1.0$   
Read/Write Mode : Dependent on current Hard Clip Limits.

### Default viewport size:

Write-Only Mode / A3 paper : 0.274 meter  
Write-Only Mode / A4 paper : 0.192 meter  
Read/Write Mode : Dependent on current Hard Clip Limits.

### Spot size (fraction of default viewport):

Write-Only Mode / A3 paper : 1/11040  
Write-Only Mode / A4 paper : 1/7721  
Read/Write Mode : Dependent on current Hard Clip Limits.

### Interactive tools:

Available in '*Read/Write Mode*' only:  
201 - Pen position.  
Optional pen status, 1=Up, 0=Down (*REATOL*).

Escape tools:  
912 - Pen speed.  
920 - Direct access to plotter commands.

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length : 6

### Special features:

- 8 hardware generated linetypes. Pattern length is by default 5 mm. Specifying a 2-digit linetype *mn* gives linetype *n* with pattern length *m*% of distance between P1 and P2 (see HP manual). Linepattern scalable.
- Hardware text in all sizes and directions. Shearing available.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:

The plotter will normally use Xon/Xoff handshake.  
Additional information available.

## No. 5 'HP20' Hewlett-Packard 7220 plotter

### Description:

Driver for Hewlett-Packard 7220 plotter.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.403 \times 1.0$

### Default viewport size:

0.285 meter

### Spot size (fraction of default viewport):

1/11400

### Interactive tools:

201 - Pen position.

Optional pen status, 1=Up, 0=Down (*REATOL*).

Escape tools:

912 - Pen speed.

920 - Direct access to plotter commands.

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length : 8

### Special features:

- 8 hardware generated linetypes. Pattern length is by default 2% of the distance P1-P2 (see HP manual for description on P1 and P2). Specifying a 2-digit linetype *mn* gives linetype *n* with pattern length *m*%.
- Hardware text in all sizes and directions. Shearing available.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:

The plotter will normally use Xon/Xoff handshake.

Automatic paper advance specified by **CALL CLRDEV (idev, icod)**, where

**icod**=100 gives half page advance,

**icod**=200 gives full page advance.

Additional information available.

## No. 6 'RASP' Raster printers

### Description:

Driver for raster (matrix) printers. The driver builds a raster image that is dumped on the printer. The following printers are currently supported:

Philips GP300                      Wenger 4/1                      HP LaserJet II

The list will be extended on demand.

### Device dependent options:

Option 3 : Printer code.

- 3 - Philips GP300
- 8 - HP LaserJet II, resolution 150 dots/inch
- 10 - Wenger 4/1, resolution 72 dots/inch, 1-colour.
- 11 - Wenger 4/1, resolution 144 dots/inch, 1-colour.
- 12 - Wenger 4/1, resolution 72 dots/inch, 4-colour.
- 13 - Wenger 4/1, resolution 144 dots/inch, 4-colour.

Option 4 : Default linewidth in pixels. If not given, 1 is used.

### Maximum viewport size (NDC):

Printer code 3	: $1.381 \times 1.0$
Printer code 8	: $1.413 \times 1.0$
Printer code 9-12	: $1.388 \times 1.0$

### Default viewport size:

Printer code 3, 9-12	: 0.204 meter
Printer code 8	: 0.197 meter

### Spot size (fraction of default viewport):

Printer code 3, 10, 12	: 1/1152
Printer code 8	: 1/1168
Printer code 9, 11	: 1/576

### Interactive tools:

None

### Retained segments:

None

### Raster facilities:

- Fixed colour table. Length : printer dependent.

Printer code 3, 8, 9, 11	: Length 1
Printer code 10, 12	: Length 7
- Polygon fill with solid colour.
- 77 predefined patterns.
- 3 user definable patterns. Fixed size 16×16.
- Pixel array drawing.

### Special features:

- Linetypes are generated by driver software. Linewidth: 1 to 9 pixels.
- No hardware text.
- Markers are generated by driver software.

### Miscellaneous:

## No. 8 'FILE' GPGS-F binary metafile generator

### Description:

Driver for writing device independent picture code on unformatted binary file. The file contents may be displayed only on computers of the same type as the one on which the file was generated (compare to driver 3).

### Device dependent options:

Text option 1 : Output file name. Not used with the VAX/VMS version.

### Maximum viewport size (NDC):

1.0 × 1.0

### Default viewport size:

1.0 meter

### Spot size (fraction of default viewport):

Not relevant.

### Interactive tools:

None

### Retained segments:

None

### Raster facilities:

Pixel array not available, all other raster commands written to output file.

### Special features:

All primitives and primitive attributes are written to output file without checking.

### Miscellaneous:

With the VAX/VMS version, logical file **FOR008** will be used as output file. With other versions, the driver will ask for the output file name if this is not given by text option 1.

## FILESHOW: Program for displaying contents of file

The **FILESHOW** program will read a file generated by the **FILE** driver and display the contents on a device specified by the user.

The VAX/VMS version will read from logical file **FOR008**, other versions will ask for the name of the file to read.

All versions will ask for the output device number, the size of the resulting drawing, and the number of options to supply through *DEVOPT*.

With the Unix version, specifying option 1 to *DEVOPT*>0 will make the program ask for the name of a file to use for driver output. This file is opened within the **FILESHOW** program by using the GPGS-F routine *NITTTY*.

## No. 9 'HPGL' Hewlett-Packard 7470 plotter

### Description:

Driver for Hewlett-Packard 7470 plotter.

### Device dependent options:

- Option 3 : = 1 → The driver will operate in '*Write-Only Mode*'.  
Other values give '*Read/Write Mode*'.  
Option 6 : = 1 → Default linepattern length will be 2% of distance between P1  
and P2 (backwards compatibility).

### Maximum viewport size (NDC):

$1.424 \times 1.0$

### Default viewport size:

0.191 meter

### Spot size (fraction of default viewport):

1/7650

### Interactive tools:

- Available in '*Read/Write Mode*' only:  
201 - Pen position.  
Optional pen status, 1=Up, 0=Down (*REATOL*).  
Escape tools:  
912 - Pen speed.  
920 - Direct access to plotter commands.

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length : 2

### Special features:

- 8 hardware generated linetypes. Pattern length is by default 5 mm. Specifying a 2-digit linetype *mn* gives linetype *n* with pattern length *m*% of distance between P1 and P2 (see HP manual). Linepattern scalable.
- Hardware text in all sizes and directions. Shearing available.
- Markers are generated by driver software.

### Miscellaneous:

The plotter will normally use Xon/Xoff handshake. In '*Read/Write Mode*', the plotter will stop and wait for manual mounting of pen if pen different than 1 or 2 is selected.

Additional information available.

## No. 10 'TX63' Tektronix 4663 plotter

### Description:

Driver for Tektronix 4663 A2-size plotter.

### Device dependent options:

None

### Maximum viewport size (NDC):

Page format:	C hor.	A2 hor.	A3 hor. or vert.
Max. viewport:	$1.354 \times 1.0$	$1.440 \times 1.0$	$1.440 \times 1.0$

### Default viewport size:

Page format:	C hor.	A2 hor.	A3 hor. or vert.
Size (meter):	0.394	0.4	0.277

### Spot size (fraction of default viewport):

1/3000

### Interactive tools:

4 - Bell (*ECHTOL*).  
201 - Pen position.  
Optional **MOVE(0)/DRAW(1)/LAST(2)** button pressed (*REATOL*).

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length : 2  
The plotter will stop and wait for manual mounting of pen if other pens are addressed.

### Special features:

- 6 hardware generated linetypes.
- Hardware text in all sizes and directions. Shearing available.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:

## No. 11 'HP21' Hewlett-Packard 7221 plotter

### Description:

Driver for Hewlett-Packard 7221 plotter.

### Device dependent options:

- Option 2 : Handling of out-of-range pen numbers.  
0 → installation dependent default action.  
1 → use pen number 1.  
2 → stop and wait until pen is manually changed, and **ENTER** pushed.
- Option 3 : Paper change action.  
0 → installation dependent default action.  
1 → change paper manually, and push **ENTER**.  
2 → automatic paper advance.

### Maximum viewport size (NDC):

1.5 × 1.0

### Default viewport size:

0.250 meter

### Spot size (fraction of default viewport):

1/2000

### Interactive tools:

- 201 - Pen position.  
Optional pen status, **1=Up, 0=Down (REATOL)**.

Escape tools:  
912 - Pen speed.

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length : 4 or 8

### Special features:

- 10 hardware generated linetypes. Linetype 1-10 give *fixed dash lines*, linetypes 11-20 give same patterns as *variable dash lines* (see HP manual).
- Hardware text in all sizes and directions. Shearing available.
- Hardware circle arcs.
- 16 hardware generated markers.

### Miscellaneous:

Action performed on **CALL CLRDEV (ldev, lcod)**:  
**lcod** = 0, default action as given by device option 3.  
**lcod** = 100, advance paper a half page.  
**lcod** = 200, advance paper a full page.

Additional information available.

## No. 12 'TX62' Textronix 4662 plotter

### Description:

Driver for Tektronix 4662 plotter. Versions available for 1-pen and 8-pen plotter.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.5 \times 1.0$

### Default viewport size:

0.250 meter

### Spot size (fraction of default viewport):

$1/2732$

### Interactive tools:

- 4 - Bell (*ECHTOL*).
- 201 - Pen position.  
Optional pen status, **1=Up**, **0=Down** (*REATOL*).

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length : 1 or 8

### Special features:

- 5 different linetypes generated by driver software.
- Hardware text in all sizes and directions.
- Markers are generated by driver software.

### Miscellaneous:



## No. 13 'CALC' CalComp pen plotters

### Description:

Driver for CalComp pen plotters, and compatibles. The driver interfaces to HCBS (Host Computer Basic Software). See also driver no. 81.

### Device dependent options:

Option 1 : Passed as third argument to HCBS routine *PLOTS*.

### Maximum viewport size (NDC):

40.0 × 1.0

### Default viewport size:

Plotter dependent.

### Spot size (fraction of default viewport):

Plotter dependent

### Interactive tools:

Escape tools:  
913 - Farr(1) = Line pattern scaling factor.

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length : Plotter dependent

### Special features:

- 5 different linetypes generated by driver software.
- Hardware text simulated by driver software.
- Markers are generated by driver software.

### Miscellaneous:

The driver assumes that HCBS uses cm's as device units.

The following HCBS routines are called by the driver:

*PLOTS*   *PLOT*   *NEWPEN*

## No. 14 'KING' Kongsberg Kingmatic

### Description:

Driver for Kongsberg Kingmatic drawing table.

### Device dependent options:

None

### Maximum viewport size (NDC):

$2.5 \times 2.0$

### Default viewport size:

0.60 meter

### Spot size (fraction of default viewport):

1/6000

### Interactive tools:

None

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length : 4

### Special features:

- 10 hardware generated linetypes.
- Hardware text in all sizes and directions.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:

## No. 15 'VERS' Versatec monochrome plotters

### Description:

Driver for Versatec monochrome raster plotters.

The driver is interfaced to Versaplot software.

See also driver no. 85

### Device dependent options:

Option 3 : Passed as first argument to *PLOTS*.

Option 4 : Paper width in inches of actual plotter.

Default: Installation dependent.

Option 5 : Max. paper length in percent of paper width.

Default: 130.

Option 6 : Plotter density, dots per inch.

Default: Installation dependent.

### Maximum viewport size (NDC):

Plotter dependent.

### Default viewport size:

Plotter dependent.

### Spot size (fraction of default viewport):

Plotter dependent.

### Interactive tools:

None

### Retained segments:

None

### Raster facilities:

- Fixed colour table. Length: 1
- 10 user definable patterns. Fixed size 16×16

### Special features:

- 7 hardware generated linetypes.  
Linewidth: 1 to 9 dots. Default: Installation dependent.
- Hardware text in all sizes and directions.  
Character height/width ratio is constant.
- Hardware circle arcs.
- 14 hardware generated markers.

### Miscellaneous:

The Versaplot software has to be changed to use this driver.

Additional information available.

## No. 18 'HPGL' Hewlett-Packard 758x plotters

### Description:

Driver for Hewlett-Packard 7580, 7585 and 7586 plotters.

### Device dependent options:

Option 3 : = 1 → The driver will operate in 'Write-Only Mode'.

Option 4 : Paper size if 'Write-Only Mode',  $n=0-4$  means  $A_n$  size.

Other paper sizes may be specified as  $lllww$  where  $lll$  is the paper length and  $ww$  is the paper width, both in cm's.

Option 5 : Plotter model if 'Write-Only Mode'. One of: 7580, 7585, 7586, -7586 (7586 with sheet media), 9001 (as 7586, but using *AH* instead of *PG*).

Option 6 : = 1 → Default linepattern length will be 2% of distance between P1 and P2 (backwards compatibility).

Option 7 : = 1 → Plot is rotated 90 degrees.

### Maximum viewport size (NDC):

Write-Only Mode:	A0	A1	A2	A3	A4
7580 :	-	1.383×1.0	1.384×1.0	1.359×1.0	1.0×1.645
7585/6 :	1.469×1.0	1.383×1.0	1.0×1.523	1.359×1.0	1.0×1.645

Read/Write Mode : Dependent on current Hard Clip Limits.

### Default viewport size:

Write-Only Mode:	A0	A1	A2	A3	A4
7580 :	-	0.568 m	0.393 m	0.271 m	0.165 m
7585/6 :	0.817 m	0.568 m	0.373 m	0.271 m	0.165 m

Read/Write Mode : Dependent on current Hard Clip Limits.

### Spot size (fraction of default viewport):

Write-Only Mode:	A0	A1	A2	A3	A4
7580 :	-	1/22720	1/15720	1/10860	1/6600
7585/6 :	1/32680	1/22720	1/14950	1/10860	1/6600

Read/Write Mode : Dependent on current Hard Clip Limits.

### Interactive tools:

Available in 'Read/Write Mode' only:

201 - Pen position. Optional pen status, 1=Up, 0=Down (*REATOL*)

Escape tools:

912 - Pen speed and acceleration.

920 - Direct access to plotter commands.

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length : 8

### Special features:

- 8 hardware generated linetypes. Pattern length is by default 5 mm. Specifying a 2-digit linetype  $mn$  gives linetype  $n$  with pattern length  $m\%$  of the distance between P1 and P2 (see HP manual). Adding 100 to the linetype will draw lines using complete pattern segments (negative linetypes in HPGL). Linepattern scalable.
- Hardware text in all sizes and directions. Shearing available.
- Hardware circle arcs.
- 15 hardware generated markers.

### Miscellaneous:

The plotter will normally use Xon/Xoff handshake.

Additional information available.

## No. 19 'CC81' CalComp-81 plotter

### Description:

Driver for CalComp-81 A3-size pen plotter. The same plotter is available from other vendors (Philips, Servogor and others?).

See also driver no. 86

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.207 \times 1.0$

### Default viewport size:

0.280 meter

### Spot size (fraction of default viewport):

1/2800

### Interactive tools:

201 - Pen position.

Escape tools:

920 - Direct access to plotter commands.

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length: 8

### Special features:

- 6 hardware generated linetypes. Pattern length is scaled by using a 2-digit linetype 'nm' where 'm' is the wanted linetype. Increasing 'n' increase the pattern length.
- Hardware text in all sizes and directions. Shearing available.
- Hardware circle arcs.
- 9 hardware generated markers.

### Miscellaneous:

The plotter will use Xon/Xoff handshake.

Action performed on **CALL CLRDEV (ldev, lcod)**:

**lcod** = 0-99, paper must be changed manually.

**lcod** = 100, paper is advanced approx. 4 cm's more than actually used.

**lcod** = 101-163, paper is advanced (**lcod**-100) cm's.

## No. 20 'TX14' Tektronix 4010-4013

### Description:

Driver for Tektronix 4010-4013 storage tubes, and a lot of compatibles.

See also driver no. 21.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.3 \times 1.0$

### Default viewport size:

0.1422 meter

### Spot size (fraction of default viewport):

1/780

### Interactive tools:

- 2 - Keyboard.
- 3 - Pick, 10 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor.  
Optional character typed, A1 format (*REATOL*).
- 202 - Optional tablet.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- No selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

Fixed colour table. Length : 1

### Special features:

- 5 software generated linetypes.
- Hardware text in 1 size, 1 direction.
- Markers are generated by driver software.

### Miscellaneous:

## No. 21 'TX14' Tektronix 4014-4015

### Description:

Driver for Tektronix 4014-4015 storage tubes, and a lot of compatibles.

See also driver no. 20.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.3 \times 1.0$

### Default viewport size:

0.2804 meter

### Spot size (fraction of default viewport):

1/3120

### Interactive tools:

- 2 - Keyboard.
- 3 - Pick, 10 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor.  
Optional character typed, A1 format (*REATOL*).
- 202 - Optional tablet.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- No selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

Fixed colour table. Length : 1

### Special features:

- 5 hardware generated linetypes.
- Hardware text in 4 size, 1 direction.
- Markers are generated by driver software.

### Miscellaneous:

## No. 22 'TX44' Tektronix 4114

### Description:

Driver for Tektronix 4114 storage tube.

### Device dependent options:

Option 3 : Coordinate mode, 10=10 bits, 12=12 bits (default).

Option 4 : If set to 1, the driver will not ask the terminal for segment status when opening new segments and changing segment attributes.

### Maximum viewport size (NDC):

$1.3 \times 1.0$

### Default viewport size:

0.267 meter

### Spot size (fraction of default viewport):

1/3120

### Interactive tools:

- 2 - Keyboard.
  - 3 - Pick, 1 level namestack.
  - 4 - Bell (*ECHTOL*).
  - 201 - Graphic cursor.
    - Optional character typed, A1 format (*REATOL*).
    - Echo types: cursor, rubberband line, segment drag.
  - 202 - Tablet. Optional button number as A1 format character (*REATOL*).
  - 203 - Tablet stroke.
- Escape tools :
- 920 - Direct access to terminal commands.

### Retained segments:

- Retained segments stored in device.
- No selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

Fixed colour table. Length : 1

### Special features:

- 9 hardware generated linetypes.
- Hardware text in all sizes and directions.
- 9 hardware generated markers, 1 size only.

### Miscellaneous:

Additional information available.



## No. 33 'APOL' Apollo display

### Description:

Driver for monochrome or colour display.  
The driver interfaces Apollo GPR functions.

### Device dependent options:

- Option 1 :     Operating mode.  
              = 0 (or *DEVOPT* not used) → a separate graphic window is created.  
              = 1 → Borrow mode, should only be used if special needs.
- Option 3 - 6 : Position of window borders in sequence *left-right-bottom-top* given  
              as percentage of the largest available square on the display.  
              (0,0) is lower left corner of display.  
              If not specified, installation dependent values are used.

### Maximum viewport size (NDC):

Application dependent.

### Default viewport size:

Application dependent.

### Spot size (fraction of default viewport):

Application dependent.

### Interactive tools:

- 3 - Pick, 10 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor.
  - Optional character typed, A1 format (*REATOL*).
  - Mouse buttons return L/M/R for Left/Middle/Right.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Dynamic colour table. Length : display dependent.
- Polygon fill with solid colour.
- 3 user definable patterns. Fixed size 32×32.
- Pixel array drawing.

### Special features:

- 8 hardware generated linetypes.
- Hardware text in 3 sizes, 1 direction.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:

## No. 35 'ICGM' ISO CGM generator

### Description:

Driver for generating a CGM metafile according to ISO 8632-1 1987 (E).

### Device dependent options:

- Option 1 : Output unit number for the metafile.
- Option 2 : CGM Scaling Mode. 0 → Abstract, 1 → Metric.
- Option 3 : Type of CGM encoding.
  - 2 → Character (default)
  - 3 → Binary
  - 4 → Clear Textaccording to ISO 8632-*n*-1987 (E), where *n* is the specified encoding.
- Option 4 : Number of pixels in default viewport, X-direction. Default: 512.
- Option 5 : Number of pixels in default viewport, Y-direction. Default: 512.  
**NOTE!** Options 4 and 5 are relevant only when pixel related GPGS-F routines are used (pixel array and patterned polygon fill).
- Option 6 : 'Page' size in X-direction. If scaling mode (see option 2) is abstract, this is specified as % of VDC (i.e. 50 means VDC 0.5).  
If scaling mode is metric, the value is specified as cm's.  
Default : 142% if scaling mode is abstract, 27 if scaling mode is metric.
- Option 7 : 'Page' size in Y-direction. Specified as option 6.  
Default : 100% if scaling mode is abstract, 19 if scaling mode is metric.  
**NOTE!** Options 6 and 7 define the mapping from GPGS-F to CGM coordinates. In addition, the 'page' size given will be written to the CGM file as arguments to the 'VDC Extent' command. If one of the options is specified negative, the 'page' size will be set to the absolute value of the option, *and the 'VDC Extent' command will not be written to the file.*
- Option 8 : Default initialization of background/foreground colours.
  - 0 → Default background = black, default foreground = white,
  - 1 → Default background = white, default foreground = black.

### Maximum viewport size (NDC):

As specified by options 6 and 7. Default:  $1.42 \times 1.0$

### Default viewport size:

As specified by options 6 and 7.

Default: 1.0 meter if scaling mode is abstract, 0.19 meter if scaling mode is metric.

### Spot size (fraction of default viewport):

As specified by Options 4 and 5.

### Interactive tools:

None

### Retained segments:

None

### Raster facilities:

- Static/dynamic colour table. Length: 255.
- Polygon fill with all fill types.
- 672 predefined patterns (24 geometric patterns repeated 28 times with different colour combinations).
- Predefined hatch styles are left to the interpreter.
- 8 user definable patterns, no size limitation.
- No user definable hatch styles.
- Pixel array drawing.

### Special features:

All primitives and primitive attributes are mapped into CGM code and written sequentially to the output file without checking.

If the value of a GPGS-F attribute does not fit into a reserved CGM index, those GPGS-F codes are rather conveyed as the negative of its original value (CGM: negative = implementation dependent).

Concatenated text is taken special care of.

### Miscellaneous:

The application program must open the output file and give the unit number as option 1.

Clear Text Encoding assumes that the file is opened as a standard Fortran FORMATTED, SEQUENTIAL file (Unix users: use Fortran OPEN).

Character and Binary Encoding are written to unformatted sequential files (Unix users: use *N/TTY*). While Character Encoding produces an ASCII file, Binary Encoding will use the computer's internal character codes.

## No. 40 'C600' CalComp 600

### Description:

Driver for CalComp 600 series digitizers.

This is an input-only driver.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.333 \times 1.0$

### Default viewport size:

0.762 meter

### Spot size (fraction of default viewport):

Installation dependent.

(1/7620, 1/3000, 1/3048)

### Interactive tools:

201 - Locator position.

Optional ASCII value of button pressed, A1 format (*REATOL*).

### Retained segments:

Not relevant.

### Raster facilities:

Not relevant.

### Special features:

Not relevant.

### Miscellaneous:

## No. 41 'ALTK' Altek AC40

### Description:

Driver for Altek AC40 digitizer.

This is an input-only driver.

### Device dependent options:

None

### Maximum viewport size (NDC):

Installation dependent.

### Default viewport size:

Installation dependent.

### Spot size (fraction of default viewport):

Installation dependent.

(1/1000 inch, 1/100 cm, 1/50 cm)

### Interactive tools:

201 - Locator position.

Optional ASCII value of button pressed, A1 format (*REATOL*).

### Retained segments:

Not relevant.

### Raster facilities:

Not relevant.

### Special features:

Not relevant.

### Miscellaneous:

## No. 51 'HP48' Hewlett-Packard 2648

### Description:

Driver for Hewlett-Packard 2648 raster terminal, and compatibles.

### Device dependent options:

None

### Maximum viewport size (NDC):

$2.0 \times 1.0$

### Default viewport size:

0.127 meter

### Spot size (fraction of default viewport):

1/360

### Interactive tools:

- 2 - Keyboard.
- 3 - Pick, 10 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).  
Echo types : cursor, rubberband line.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

Fixed colour table. Length : 1

### Special features:

- 9 hardware generated linetypes.
- Hardware text in 8 sizes, 4 directions.
- Markers are generated by driver software.

### Miscellaneous:

## No. 53 'TX27' Tektronix 4027

### Description:

Driver for Tektronix 4027 colour raster terminal.

### Device dependent options:

None

### Maximum viewport size (NDC):

1.43 × 1.65 (possible to draw in off-screen bitmap).

### Default viewport size:

0.191 meter

### Spot size (fraction of default viewport):

1/448

### Interactive tools:

- 2 - Keyboard.
- 3 - Pick, 10 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Dynamic colour table. Length : 7
- Polygon fill with solid colour.

### Special features:

- 9 hardware generated linetypes.
- Hardware text in 1 size, 1 direction.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:

## No. 54 'TX25' Tektronix 4025

### Description:

Driver for Tektronix 4025 monochrome raster terminal.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.43 \times 1.65$  (possible to draw in off-screen bitmap).

### Default viewport size:

0.170 meter

### Spot size (fraction of default viewport):

1/448

### Interactive tools:

- 2 - Keyboard.
- 3 - Pick, 10 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

Dynamic colour table. Length : 1

### Special features:

- 9 hardware generated linetypes.
- Hardware text in 1 size, 1 direction.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:



## No. 55 'TD22' Tandberg Data TDV 2215

### Description:

Driver for Tandberg Data TDV 2215, using semigraphic character set.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.06 \times 1.0$

### Default viewport size:

0.190 meter

### Spot size (fraction of default viewport):

$1/150 \times 1/75$

### Interactive tools:

- 2 - Keyboard.
- 3 - Pick, 10 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).  
Interrupt given by alpha key + <CR>

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Fixed colour table. Length : 1
- Pixel array drawing.

### Special features:

- Linetypes generated by driver software.
- Hardware text in 1 size, 4 directions.
- Markers are generated by driver software.

### Miscellaneous:

## No. 57 'REGI' DEC VT125

### Description:

Driver for DEC VT125 and other monochrome ReGIS terminals.

See also drivers 64 and 74.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.6 \times 1.0$

### Default viewport size:

0.150 meter

### Spot size (fraction of default viewport):

$1/768 \times 1/240$

### Interactive tools:

2 - Keyboard, alpha cursor positioning possible.

3 - Pick, 10 level namestack.

4 - Bell (*ECHTOL*).

201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).

Escape tools:

920 - Direct access to terminal commands.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations: *VXLAT*
- Segment priorities: 4095

### Raster facilities:

Dynamic colour table. Length: 3

### Special features:

- 10 hardware generated linetypes.
- Hardware text in limited number of sizes, 8 directions.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:

## No. 58 'TX42' Tektronix 4112

### Description:

Driver for Tektronix 4112 monochrome raster terminal.

### Device dependent options:

Option 3 : Coordinate mode, 10=10 bits, 12=12 bits (default).

Option 4 : If set to 1, the driver will not ask the terminal of segment status when opening new segments and changing segment attributes.

### Maximum viewport size (NDC):

$1.333 \times 1.0$

### Default viewport size:

0.188 meter (- dialogue area)

### Spot size (fraction of default viewport):

1/480 (- dialogue area)

### Interactive tools:

- 2 - Keyboard.
- 3 - Pick, 1 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).  
Echo types : cursor, rubberband line, segment drag.
- 202 - Tablet. Optional button number as A1 format character (*REATOL*).
- 203 - Tablet stroke.
- Escape tools :
  - 920 - Direct access to terminal commands.

### Retained segments:

- Retained segments stored in device.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Dynamic colour table. Length : 1 or 7
- Polygon fill with solid colour.
- 16 predefined patterns.
- 125 user definable patterns. No size limitation.
- Pixel array drawing.

### Special features:

- 9 hardware generated linetypes.
- Hardware text in all sizes and directions.
- 9 hardware generated markers, 1 size only.

### Miscellaneous:

The driver will reserve the bottom 2 lines for dialogue area.  
Additional information available.

## No. 59 'TX43' Tektronix 41xx/42xx

### Description:

Driver for Tektronix colour raster terminals 4106-07-09-11-13-15-25, 4207-08-09.

### Device dependent options:

Option 3 : Coordinate mode, 10=10 bits, 12=12 bits (default).

Option 4 : If set to 1, the driver will not ask the terminal of segment status when opening new segments and changing segment attributes.

### Maximum viewport size (NDC):

4115-25 :  $1.250 \times 1.0$

Others :  $1.333 \times 1.0$

### Default viewport size:

4106-07, 4207-08-09 : 0.188 meter

Others : 0.267 meter

### Spot size (fraction of default viewport):

4115-25 : 1/1024

4111 : 1/768

Others : 1/480

### Interactive tools:

2 - Keyboard.

3 - Pick, 1 level namestack.

4 - Bell (*ECHTOL*).

201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).  
Echo types : cursor, rubberband line, segment drag.

202 - Tablet. Optional button number as A1 format character (*REATOL*).

203 - Tablet stroke.

Escape tools :

920 - Direct access to terminal commands.

### Retained segments:

- Retained segments stored in device.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Dynamic colour table. Length : 7, 15, 63 or 255.
- Polygon fill with solid colour.
- 141 predefined patterns.
- 33 user definable patterns (4111-15-25 only). No size limitation.
- Pixel array drawing.

### Special features:

- 9 hardware generated linetypes.
- Hardware text in all sizes and directions.  
Shearing available with 4111-15-25.
- 9 hardware generated markers, 1 size only.

### Miscellaneous:

Additional information available.

## No. 62 'TX05' Tektronix 4105

### Description:

Driver for Tektronix 4105 colour raster terminal.

### Device dependent options:

Option 3 : Coordinate mode, 10=10 bits (default), 12=12 bits.

### Maximum viewport size (NDC):

$1.333 \times 1.0$

### Default viewport size:

0.188 meter

### Spot size (fraction of default viewport):

1/360

### Interactive tools:

- 2 - Keyboard.
- 3 - Pick, 10 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).
- Escape tools :
  - 920 - Direct access to terminal commands.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Dynamic colour table. Length : 7
- Polygon fill with solid colour.
- 141 predefined patterns.
- Pixel array drawing with option 34 (optional pixel roms).

### Special features:

- 9 hardware generated linetypes.
- Hardware text in limited number of sizes, 4 directions.
- Markers are generated by driver software.

### Miscellaneous:

Additional information available.

## No. 63 'TDGO' Tandberg Data Graphic Option

### Description:

Driver for Tandberg Data TDV 2200 with graphic option.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.379 \times 1.0$

### Default viewport size:

0.190 meter

### Spot size (fraction of default viewport):

$1/522 \times 1/336$

### Interactive tools:

- 2 - Keyboard.
- 3 - Pick, 10 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).
- Escape tools :
  - 920 - Direct access to terminal commands.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Fixed colour table. Length : 1
- Polygon fill with solid colour.
- 10 predefined patterns.
- 9 user definable patterns. Fixed size 8×8.
- Pixel array drawing and readback.

### Special features:

- 5 hardware generated linetypes.
- Hardware text in 9 sizes, 4 directions.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:

## No. 64 'REGI' DEC VT240

### Description:

Driver for DEC VT240 colour terminal.

See also drivers 57 and 74.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.666 \times 1.0$

### Default viewport size:

0.150 meter

### Spot size (fraction of default viewport):

$1/800 \times 1/240$

### Interactive tools:

2 - Keyboard, alpha cursor positioning possible.

3 - Pick, 10 level namestack.

4 - Bell (*ECHTOL*).

201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).

Escape tools :

920 - Direct access to terminal commands.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

Dynamic colour table. Length : 3

### Special features:

- 10 hardware generated linetypes.
- Hardware text in limited number of sizes, 8 directions.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:

## No. 65 'WW32' Westward 3219

### Description:

Driver for Westward 3219 monochrome terminal.

See also driver 66.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.306 \times 1.0$

### Default viewport size:

0.280 meter

### Spot size (fraction of default viewport):

1/1568

### Interactive tools:

- 2 - Keyboard, alpha cursor positioning possible.
- 3 - Pick, 10 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).  
Echo types : cursor, rubberband line.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Fixed colour table. Length : 1
- Polygon fill with solid colour.
- 4 user definable hatch styles.
- 5 user definable patterns. Fixed size 8×8.
- Pixel array drawing.

### Special features:

- 6 hardware generated linetypes. Linewidth 1 (default) to 127 pixels.
- Hardware text in 12 sizes, 4 directions.
- Hardware circle arcs.
- 13 hardware generated markers. 1 size only.

### Miscellaneous:



## No. 66 'WW32' Westward 3220

### Description:

Driver for Westward 3220 colour terminal.

See also driver 65.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.306 \times 1.0$

### Default viewport size:

0.280 meter

### Spot size (fraction of default viewport):

1/784

### Interactive tools:

- 2 - Keyboard, alpha cursor positioning possible.
- 3 - Pick, 10 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).  
Echo types : cursor, rubberband line.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Fixed colour table. Length : 15 or 255
- Polygon fill with solid colour.
- 4 user definable hatch styles.
- 5 user definable patterns. Fixed size 8×8.
- Pixel array drawing.

### Special features:

- 6 hardware generated linetypes. Linewidth 1 (default) to 127 pixels.
- Hardware text in 12 sizes, 4 directions.
- Hardware circle arcs.
- 13 hardware generated markers. 1 size only.

### Miscellaneous:

## No. 67 'TX29' Tektronix 3D terminals

### Description:

Driver for Tektronix 3D terminals, 4129 and 4237.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.25 \times 1.0$

### Default viewport size:

0.280 meter

### Spot size (fraction of default viewport):

1/1024

### Interactive tools:

- 2 - Keyboard.
- 3 - Pick, 1 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).  
Echo types : cursor, rubberband line, segment drag.
- 202 - Tablet. Optional button number as A1 format character (*REATOL*).
- 203 - Tablet stroke.
- Escape tools :
  - 920 - Direct access to terminal commands.

### Retained segments:

- Retained segments stored in device (as 3D segments).
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Dynamic colour table. Length : 15, 255 or 4095
- Polygon fill with solid colour.
- 141 predefined patterns (16 on 4129).
- 125 user definable patterns. No size limitation.
- Pixel array drawing.

### Special features:

- 9 hardware generated linetypes.
- Hardware text in all sizes and directions. Shearing available.
- 9 hardware generated markers, 1 size only.

### Miscellaneous:

Utility routines for defining facets, light sources etc. are supplied with driver.  
Additional information available.

## No. 68 'TPAZ' Tandberg Topaz 2400

### Description:

Driver for Tandberg Topaz 2400 monochrome terminal.

See also driver 70.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.333 \times 1.0$

### Default viewport size:

0.200 meter

### Spot size (fraction of default viewport):

1/600

### Interactive tools:

2 - Keyboard.

3 - Pick, 10 level namestack.

4 - Bell (*ECHTOL*).

201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).

Escape tools :

920 - Direct access to terminal commands.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Fixed colour table. Length : 3
- Polygon fill with solid colour.
- 6 predefined hatch styles.
- 48 predefined patterns.
- Pixel array drawing.

### Special features:

- 8 hardware generated linetypes.
- Hardware text in limited number of sizes, 8 directions.
- Hardware circle arcs.
- 5 hardware generated markers.

### Miscellaneous:

## No. 69 'WWGM' Westward Graphics Manager

### Description:

Driver for Westward Graphics Manager, 3D mode.

### Device dependent options:

Option 5 : If set to 1, the driver will operate in 2D mode.

### Maximum viewport size (NDC):

$1.25 \times 1.0$

### Default viewport size:

0.280 meter

### Spot size (fraction of default viewport):

Dependent on display attached.

### Interactive tools:

- 2 - Keyboard.
- 3 - Pick, 1 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).  
Echo types : cursor, rubberband line.

### Retained segments:

- Retained segments stored in device (as 3D segments).
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Dynamic colour table. Length : 1, 15 or 255.
- Polygon fill with solid colour.

### Special features:

- 6 hardware generated linetypes.
- Hardware text in all sizes and directions.
- Hardware circle arcs.
- 13 hardware generated markers, 1 size only.

### Miscellaneous:

## No. 70 'TPAZ' Tandberg Topaz 2500

### Description:

Driver for Tandberg Topaz 2500 colour terminal.

See also driver 68.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.333 \times 1.0$

### Default viewport size:

0.200 meter

### Spot size (fraction of default viewport):

1/600

### Interactive tools:

2 - Keyboard.

3 - Pick, 10 level namestack.

4 - Bell (*ECHTOL*).

201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).

Escape tools :

920 - Direct access to terminal commands.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Dynamic colour table. Length : 15
- Polygon fill with solid colour.
- 6 predefined hatch styles.
- 240 predefined patterns.
- Pixel array drawing.

### Special features:

- 8 hardware generated linetypes.
- Hardware text in limited number of sizes, 8 directions.
- Hardware circle arcs.
- 5 hardware generated markers.

### Miscellaneous:

## No. 71 'RUBY' Tandberg 1200

### Description:

Driver for Tandberg 1200 (RUBY) monochrome raster terminal.

### Device dependent options:

Option 3 : = 1 → The driver assumes the terminal is in 2115 mode, and will reset this on exit.

### Maximum viewport size (NDC):

1.35 × 1.0

### Default viewport size:

0.2 meter

### Spot size (fraction of default viewport):

1/530

### Interactive tools:

- 2 - Keyboard.
- 3 - Pick, 10 level namestack.
- 4 - Bell (*ECHTOL*).
- 201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).  
Echo types : cursor, crosshair, rubberband line, rubberband rectangle.
- Escape tools :
  - 920 - Direct access to terminal commands.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Fixed colour table. Length : 3
- Polygon fill with solid colour.
- 6 predefined hatch styles.
- 48 predefined patterns.
- Pixel array drawing.

### Special features:

- 8 hardware generated linetypes.
- Hardware text in limited number of sizes, 1 direction.
- Hardware circle arcs.
- 5 hardware generated markers.

### Miscellaneous:

## No. 72 'XWDW' X.11 interface

### Description:

Driver interface to the X.11 window system.

### Device dependent options:

Option 1: Start condition. If set to 1, the initial window will not be mapped to the screen until explicitly done by *VISDWI* (page 21-6).

Option 2: Colour index to use for window background.

Options 3 - 6: Position of window borders in sequence *left-right-bottom-top*, relative to parent window. (0,0) is the lower left corner of the parent window.

Option 7: Units of options 3-6.

= 0 or 10 → percent of largest possible square on parent window.

= 1 or 11 → pixels.

If 0 or 1, the X11 flag **PPosition** is set, if 1 or 11 the flag **USPosition** is set (refer to X11 documentation).

Option 8: Controls setting of the X11 flags **backing\_store** and **save\_under** (default: **backing\_store=Always**, **save\_under=False**).

Option 9: Initial background/foreground colours (default: white/black on monochrome devices, black/white on colour devices).

Option 10: Number of read/write colour cells to allocate (default: the driver will allocate as many as possible).

Should be set to the actual number needed to leave some colours for other applications.

Option 11: Window border in pixels.

Option 12: Window border colour index.

Option 13: Close mode. If set to 1, the X11 routine **XCLOSEDisplay** will not be called by *RLSDEV*.

Text option 1: Window name (shown in the window frame).

Text option 2: Icon name.

The options are described in full detail in separate documentation delivered with the driver.

### Maximum viewport size (NDC):

Application dependent.

### Default viewport size:

Application dependent.

### Spot size (fraction of default viewport):

Application dependent.

### Interactive tools:

3 - Pick, 10 level namestack. Request and sample mode.

4 - Bell (*ECHTOL*).

201 - Graphic cursor. Request and sample mode. Echo modes 'Rubberband Line' and 'Rubberband Rectangle' are available in request mode.

401 to 405 - Mouse button status. *Sample mode only*.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Fixed or dynamic colour table. Length : display dependent.
- Polygon fill with solid colour.
- 24 predefined two-colour patterns. As the two colours used may be varied, the total number of patterns is display dependent, computed as  $[maxcol*(maxcol+1)* 12]$ , where *maxcol* is the max. number of foreground colours available.
- Pixel array drawing.

### Special features:

- Up to 30 simultaneously active windows.
- 10 hardware generated linetypes. Linewidth 1 (default) to 6 pixels.
- Hardware text in limited number of sizes, 1 direction, 16 fonts, 8 bits character set.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:

Additional information available. This gives detailed description of the device options, graphic primitives and input facilities.



## No. 73 'TECH' ND Technostation

### Description:

Driver for Norsk Data Technostation colour display.  
The driver is interfaced to Leonardo software.

### Device dependent options:

Option 3 : If not zero, deferral mode is set to **ASAP** (As Soon As Possible).  
Default mode is **ASTI** (At Some Time). See Leonardo documentation.

### Maximum viewport size (NDC):

1.25 × 1.0

### Default viewport size:

0.275 meter

### Spot size (fraction of default viewport):

1/1024

### Interactive tools:

3 - Pick, 10 level namestack.  
201 - Graphic cursor.  
Echo types : cursor, crosshair, rubberband line, rubberband rectangle.  
Tablet is used for both pick and locator device.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Dynamic colour table. Length : 255
- Polygon fill with solid colour.
- Pixel array drawing.

### Special features:

- 6 hardware generated linetypes. Linewidth 1 (default) to 8 pixels.
- Hardware text in limited number of sizes, 4 direction.
- Hardware circle arcs.
- 7 hardware generated markers.

### Miscellaneous:

## No. 74 'REGI' DEC VT340

### Description:

Driver for DEC VT340 colour terminal.

See also drivers 57 and 64.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.666 \times 1.0$

### Default viewport size:

0.150 meter

### Spot size (fraction of default viewport):

$1/800 \times 1/480$

### Interactive tools:

2 - Keyboard, alpha cursor positioning possible.

3 - Pick, 10 level namestack.

4 - Bell (*ECHTOL*).

201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).

Escape tools :

920 - Direct access to terminal commands.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Dynamic colour table. Length : 15
- Polygon fill with solid colour.

### Special features:

- 10 hardware generated linetypes.
- Hardware text in limited number of sizes, 8 directions.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:

## No. 75 'NROWS' Norsk Data OWS

### Description:

Driver for Norsk Data OWS-55/85.

### Device dependent options:

None.

### Maximum viewport size (NDC):

$1.333 \times 1.0$

### Default viewport size:

0.172 meter

### Spot size (fraction of default viewport):

1/360

### Interactive tools:

- 2 - Keyboard
  - 3 - Pick, 10 level namestack.
  - 4 - Bell (*ECHTOL*).
  - 201 - Graphic cursor. Optional character typed, A1 format (*REATOL*).
- Escape tools:
- 920 - Direct access to terminal commands.

### Retained segments:

- Retained segments stored in GPGS-F buffer/pick simulation module.
- Selective erase.
- Image transformations : *VXLAT*
- Segment priorities : 4095

### Raster facilities:

- Fixed colour table. Length : 7
- Rectangle fill with solid colour.

### Special features:

- 6 hardware generated linetypes.
- Hardware text in 1 size, 1 direction
- 9 hardware generated markers. 1 size only.

### Miscellaneous:

## No. 80 'HPGL' Hewlett-Packard 7550 plotter

### Description:

Driver for Hewlett-Packard 7550 plotter.

### Device dependent options:

- Option 3 : = 1 → The driver will operate in '*Write-Only Mode*'.  
Other values give '*Read/Write Mode*'.
- Option 4 : Paper size when '*Write-Only Mode*'.  
3 = A3 paper, 4 = A4 paper.
- Option 6 : = 1 → Default linepattern length will be 2% of distance between P1 and P2 (backwards compatibility).
- Option 7 : = 1 → Plot is rotated 90 degrees.

### Maximum viewport size (NDC):

Write-Only Mode / A3 paper :  $1.469 \times 1.0$   
Write-Only Mode / A4 paper :  $1.430 \times 1.0$   
Read/Write Mode : Dependent on current Hard Clip Limits.

### Default viewport size:

Write-Only Mode / A3 paper : 0.271 meter  
Write-Only Mode / A4 paper : 0.190 meter  
Read/Write Mode : Dependent on current Hard Clip Limits.

### Spot size (fraction of default viewport):

Write-Only Mode / A3 paper : 1/10870  
Write-Only Mode / A4 paper : 1/7600  
Read/Write Mode : Dependent on current Hard Clip Limits.

### Interactive tools:

Available in '*Read/Write Mode*' only:  
201 - Pen position.  
Optional pen status, 1=Up, 0=Down (*REATOL*).

Escape tools:  
912 - Pen speed and acceleration.  
920 - Direct access to plotter commands.

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length : 8

### Special features:

- 8 hardware generated linetypes. Pattern length is by default 5 mm. Specifying a 2-digit linetype *mn* gives linetype *n* with pattern length *m*% of the distance between P1 and P2 (see HP manual). Adding 100 to the linetype will draw lines using complete pattern segments (negative linetypes in HPGL). Linepattern scalable.
- Hardware text in all sizes and directions. Shearing available.
- Hardware circle arcs.
- 15 hardware generated markers.

### Miscellaneous:

The plotter will normally use Xon/Xoff handshake.  
Additional information available.

## No. 81 'CPCI' CalComp PCI controller

### Description:

Driver for CalComp pen plotters with PCI controller. The driver is interfaced to HCBS (Host Computer Basic Software). See also driver no. 13.

### Device dependent options:

Option 1 : Passed as third argument to HCBS routine *PLOTS*.

Option 3 : Number of pens available.  
Default: Installation dependent.

Option 4 : Paper width in cm's.  
Default: Installation dependent.

### Maximum viewport size (NDC):

45.0 × 1.0

### Default viewport size:

Installation / plotter dependent.

### Spot size (fraction of default viewport):

Installation / plotter dependent.

### Interactive tools:

None

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length: Installation / plotter dependent.

### Special features:

- 9 hardware generated linetypes.
- Hardware text in (nearly) all sizes and directions.
- Hardware circle arcs.
- 15 hardware generated markers.

### Miscellaneous:

The driver assumes that HCBS uses cm's as device units.

The following HCBS routines are called by the driver:

*CIRCLE DASHS FONT NEWPEN*  
*PLOT PLOTS SETCHR SYMBOL*

Note that the HCBS routine *TLRNCE* is not used.

## No. 82 'CNA2' Canon LBP-8

### Description:

Driver for Canon LBP-8 A2, II and III laser plotters.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.406 \times 1.0$

### Default viewport size:

0.197 meter

### Spot size (fraction of default viewport):

1/5606

### Interactive tools:

None

### Retained segments:

None

### Raster facilities:

- Fixed colour table. Length: 4 !!  
1 = 'Fine line', 2 = 'Semi-fine line', 3 = 'Semi-thick line', 4 = 'Thick line'  
Possible to draw with colour index 0 (erase mode).
- Polygon fill with solid colour.
- 8 predefined patterns.
- 2 user definable patterns. Fixed size 32×32.
- Pixel array drawing.

### Special features:

- 8 hardware generated linetypes.
- Hardware text in 1 size, 1 direction.
- Hardware circle arcs.
- 9 hardware generated markers.

### Miscellaneous:

On model LBP-8 A2, '**Full Paint Mode**' must be set by device switches (switches 7 and 8 on SW4).

## No. 83 'HPGL' Hewlett-Packard 7440 plotter

### Description:

Driver for Hewlett-Packard 7440 (ColorPro) plotter.

### Device dependent options:

Option 3 : = 1 → The driver will operate in '*Write-Only Mode*'. Other values give '*Read/Write Mode*'.

Option 4 : '*Write-Only Mode*' only:

= 1 → The plotter is equipped with the '*Graphics Enhancement Cartridge*'.

Option 7 : = 1 → Plot is rotated 90 degrees.

### Maximum viewport size (NDC):

1.424 × 1.0

### Default viewport size:

1.191 meter

### Spot size (fraction of default viewport):

1/7650

### Interactive tools:

Available in '*Read/Write Mode*' only:

201 - Pen position.

Optional pen status, 1=Up, 0=Down (*REATOL*).

Escape tools:

912 - Pen speed.

920 - Direct access to plotter commands.

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length : 8

### Special features:

- 8 hardware generated linetypes. Pattern length is by default 5 mm. Specifying a 2-digit linetype *mn* gives linetype *n* with pattern length *m%* of distance between P1 and P2 (see HP manual).
- Hardware text in all sizes and directions. Shearing available.
- Hardware circle arcs if '*Graphics Enhancement Cartridge*'.
- Markers are generated by driver software.

### Miscellaneous:

The plotter will normally use Xon/Xoff handshake.

Additional information available.

## No. 84 'TA10' Wild Aviotab TA10

### Description:

Driver for Wild Aviotab TA10.

### Device dependent options:

Option 3 : = 1 → The driver will operate in '*Write-Only Mode*'. Other values give '*Read/Write Mode*'.

Option 4 : Number of pens (2 or 4) when '*Write-Only Mode*'.  
If '*Read/Write Mode*', the number of pens is read from the plotter.

### Maximum viewport size (NDC):

Write-Only Mode : 1.0 × 1.0

Read/Write Mode : Requested from plotter.

### Default viewport size:

Write-Only Mode : 1.0 meter

Read/Write Mode : Requested from plotter.

### Spot size (fraction of default viewport):

Write-Only Mode : 1/50000

Read/Write Mode : Requested from plotter.

### Interactive tools:

Available in '*Read/Write Mode*' only:

201 - Pen position.

Escape tools:

912 - Pen speed and acceleration.

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length: 2 or 4

### Special features:

- 6 hardware generated linetypes.
- Hardware text in all sizes and directions.
- Hardware circle arcs.
- 15 hardware generated markers.

### Miscellaneous:



## No. 85 'VCOL' Versatec colour raster plotters

### Description:

Driver for Versatec colour raster plotters.  
The driver is interfaced to Versaplot software.  
See also driver no. 15

### Device dependent options:

- Option 3 : Passed as first argument to *PLOTS*.
- Option 4 : Paper width in inches of actual plotter.  
Default: Installation dependent.
- Option 5 : Max. paper length in percent of paper width.  
Default: 130.
- Option 6 : Plotter density, dots per inch.  
Default: Installation dependent.

### Maximum viewport size (NDC):

Plotter dependent.

### Default viewport size:

Plotter dependent.

### Spot size (fraction of default viewport):

Plotter dependent.

### Interactive tools:

None

### Retained segments:

None

### Raster facilities:

- Fixed colour table. Length: 8
- Polygon fill with solid colour.
- 256 predefined patterns.
- 10 user definable patterns. Fixed size 16×16

### Special features:

- 7 hardware generated linetypes.  
Linewidth: 1 to 9 dots. Default: Installation dependent.
- Hardware text in all sizes and directions.  
Character height/width ratio is constant.
- Hardware circle arcs.
- 14 hardware generated markers.

### Miscellaneous:

The Versaplot software has to be changed to use this driver.  
Additional information available.

## No. 86 'CC81' CalComp-84 plotter

### Description:

Driver for CalComp-84 A4-size pen plotter. The same plotter is available from other vendors (Philips, Servogor and others?).

See also driver no. 19

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.435 \times 1.0$

### Default viewport size:

0.200 meter

### Spot size (fraction of default viewport):

1/2000

### Interactive tools:

201 - Pen position.

Escape tools:

920 - Direct access to plotter commands.

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length: 8

### Special features:

- 6 hardware generated linetypes. Pattern length is scaled by using a 2-digit linetype 'nm' where 'm' is the wanted linetype. Increasing 'n' increase the pattern length.
- Hardware text in all sizes and directions. Shearing available.
- Hardware circle arcs.
- 9 hardware generated markers.

### Miscellaneous:

The plotter will use Xon/Xoff handshake.

Action performed on **CALL CLRDEV (ldev, lcod)**:

**lcod** = 0-99, paper must be changed manually.

**lcod** = 100, paper is advanced approx. 4 cm's more than actually used.

**lcod** = 101-163, paper is advanced (**lcod**-100) cm's.

## No. 87 'LASR' Laser printers, Tektronix mode

### Description:

Driver for laser printers understanding Tektronix 4010/4014 vector commands.  
May currently be used with **DEC LN03 Plus** and **QMS Lasergrafix**.

### Device dependent options:

- Option 3 : Printer type. 1 = LN03, 2 = QMS  
Default: Installation dependent.
- Option 4 : Code to generate, 4010 or 4014  
Default: Installation dependent.

### Maximum viewport size (NDC):

LN03 :  $1.333 \times 1.0$   
QMS :  $1.312 \times 1.0$

### Default viewport size:

0.195 meter

### Spot size (fraction of default viewport):

LN03 : 1/3072  
QMS : 1/3120

### Interactive tools:

None

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length: 1

### Special features:

- Linetypes: 5 different hardware generated in 4014 mode,  
generated by driver software in 4010 mode.
- Hardware text: 4 sizes, 1 direction in 4014 mode,  
1 size, 1 direction in 4010 mode.
- Markers are generated by driver software.

### Miscellaneous:

## No. 88 'CCOL' CalComp colour raster plotters

### Description:

Driver for CalComp colour raster plotters.

The driver is interfaced to CalComp Basic Software.

Installation dependent parameter must be set corresponding to the unit of measure used in CalComp Software (cm's or inches).

### Device dependent options:

Option 1 : Passed as third argument to *PLOTS*.

Option 3 : Passed as first argument to *PLOTS*.

Option 4 : Paper width in cm's or inches (see also option 5).  
Default: Installation dependent.

Option 5 : Factor to divide option 4 by to get cm's or inches.  
E.g.: option 4 = 725, option 5 = 10 → size=72.5

Option 6 : Plotter density, dots per inch.  
Default: Installation dependent.

### Maximum viewport size (NDC):

10.0 × 1.0

### Default viewport size:

Plotter dependent.

### Spot size (fraction of default viewport):

Plotter dependent.

### Interactive tools:

None

### Retained segments:

None

### Raster facilities:

- Fixed colour table. Length: 7
- Polygon fill with solid colour.
- 105 predefined patterns.
- 10 user definable patterns. Fixed size 32×32
- Pixel array drawing.

### Special features:

- 9 hardware generated linetypes.  
Linewidth: 1 to 16 pixels. Default: Installation dependent.
- Hardware text in all sizes and directions, shearing available.
- Hardware circle arcs.
- 15 hardware generated markers.

### Miscellaneous:

Additional information available.

## **No. 89 'TX45' Tektronix 4510 rasterizer**

### **Description:**

Driver for Tektronix 4510 rasterizer. Any hardcopy model may be used (consult Tektronix documentation for details).

### **Device dependent options:**

Option 3 : If set to 1, the driver will operate in 'Write-Only Mode'. Other values give 'Read/Write Mode'.

Option 4 : Paper size when 'Write-Only Mode'.

1=A1, 2=A2, 3=A3, 4=A4  
5=D, 6=C, 7=B, 8=A (English)

Option 5 : Default linewidth in pixels (1-4). If not given, 1 is used.

### **Maximum viewport size (NDC):**

$1.307 \times 1.0$

### **Default viewport size:**

Write-Only Mode:

A1 : 0.594 m, A2 : 0.420 m, A3 : 0.297 m, A4 : 0.210 m  
D : 0.558 m, C : 0.431 m, B : 0.279 m, A : 0.215 m

Read/Write Mode: Dependent on device.

### **Spot size (fraction of default viewport):**

Dependent on device.

### **Interactive tools:**

Escape tools:

920 - Direct access to rasterizer commands.

### **Retained segments:**

None

### **Raster facilities:**

- Dynamic colour table. Length : 255
- Polygon fill with solid colour.
- 141 predefined patterns.

### **Special features:**

- 9 hardware generated linetypes. Linewidth: 1 to 4 pixels.
- Hardware text in all sizes and directions. Shearing available.
- 9 hardware generated markers.

### **Miscellaneous:**

Additional information available.

## No. 90 'PSCR' PostScript generator

### Description:

Driver generating PostScript files.

### Device dependent options:

- Option 2 : = 1 → The PostScript command '*showpage*' will not be output.  
Option 3 : = 1 → Generate colour commands instead of the default gray shades.  
Option 4 : Specifies paper size and orientation.  
0 = A4 Landscape, 1 = A4 Portrait,  
2 = A3 Landscape, 3 = A3 Portrait  
Option 5 : Raster resolution to be used by GP GS-F pixel related routines. Specified in number of pixels per inch. Default 150.  
Option 6 and 7: Paper size width and height in millimetres (overrides option 4).  
Option 8 and 9: Offset from PostScript origin to GP GS-F origin in millimetres.

### Maximum viewport size (NDC):

A4 Landscape : $1.425 \times 1.0$	A4 Portrait : $1.0 \times 1.425$
A3 Landscape : $1.423 \times 1.0$	A3 Portrait : $1.0 \times 1.423$

### Default viewport size:

A4 : 0.195 m  
A3 : 0.282 m

### Spot size (fraction of default viewport):

Relevant for pixel related routines only.  
A4 : 1/1156      A3 : 1/1668  
(with default value of option 5).

### Interactive tools:

Escape tools (see PostScript documentation for details):  
931 - Iarr(1) = PostScript '*line cap*' parameter (default 0)  
932 - Iarr(1) = PostScript '*line join*' parameter (default 0)  
Farr(1) = PostScript '*miter limit*' parameter (default 2.613)

### Retained segments:

None.

### Raster facilities:

- Static colour table. Length : 255
- Polygon fill with solid colour (or gray shades).
- 24 predefined patterns for black/white mode, 672 for colour mode.
- 5 user definable patterns. Max. size (width  $\times$  height) : 1024
- Pixel array drawing.

### Special features:

- 9 hardware generated linetypes. Linewidth and linepattern scalable.
- Hardware text in all sizes and directions, shearing, 20 fonts, 8 bits character set.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:

Additional information available (see this for more detailed description of the device dependent options).

## No. 91 'LASR' ND 720/730 laser printers

### Description:

Driver for Norsk Data 720 and 730 laser printers.

### Device dependent options:

None

### Maximum viewport size (NDC):

$1.429 \times 1.0$

### Default viewport size:

0.193 meter

### Spot size (fraction of default viewport):

$1/7722$

### Interactive tools:

None

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length: 1

### Special features:

- 8 hardware generated linetypes. Pattern length is by default 2% of the distance P1-P2 (see ND documentation). Specifying a 2-digit linetype *mn* gives linetype *n* with pattern length *m*%.
- Hardware text in all sizes and directions. Shearing available.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:

## No. 92 'GTEC' Graphtec pen plotters

### Description:

Driver for Graphtec MP3000/4000 series, FP6202/6302, GP1002/1102.

### Device dependent options:

Option 3 : = 1 → The driver will operate in '*Write-Only Mode*'.  
Other values give '*Read/Write Mode*'.

Option 4 : Paper size if '*Write-Only Mode*',  $n=0-4$  means  $A_n$  size.

Option 5 : Plotter model. One of: 3100/3200/3300/3400 / 4100/4200/4300/4400 /  
(-)6202/(-)6302 / (-)1002/(-)1102. If positive numbers are given for the  
last four models, the driver assumes roll paper is loaded, if negative  
numbers are given, the driver assumes sheet paper is loaded.

### Maximum viewport size (NDC):

Plotter dependent.

### Default viewport size:

Plotter dependent.

### Spot size (fraction of default viewport):

Plotter dependent.

### Interactive tools:

Available in '*Read/Write Mode*' only:

201 - Pen position.

Optional pen status  $mn$ , where  $m$  is pen number,  $n$  is up(1)/down(0).

Escape tools:

912 - Pen speed (Fda(1)=speed in cm/s).

920 - Direct access to plotter commands.

### Retained segments:

None

### Raster facilities:

Fixed colour table. Length : 8

### Special features:

- 8 hardware generated linetypes. Pattern length is by default 5 mm. Specifying a 2-digit linetype  $mn$  gives linetype  $n$  with pattern length  $m*2.5$  mm.
- Hardware text in all sizes and directions. Shearing available.
- Hardware circle arcs.
- 15 hardware generated markers.

### Miscellaneous:

The driver assumes that the plotter uses Xon/Xoff handshake.

*The programmable unit must be set to 0.1 mm.*



## No. 93 'HPG2' HP-GL/2 generator

### Description:

Driver generating HP-GL/2 commands. By default, PCL Dual-Context instructions are included, see option 3 below.

### Device dependent options:

- Option 2 : = 1 → Paper must be loaded manually.  
Recognized by PCL devices only (see option 3).
- Option 3 : = 1 → Do not include Dual-Context PCL instructions.
- Option 4 : Fixed paper sizes (see also options 6 and 7). Default is A4  
1 - 4 = A1 - A4, 10 = A0
- Option 5 : Type of device  
0 = Black/white printer, 1 = Colour raster  
2 = Colour vector (pen plotter) 3 = Monochrome raster
- Option 6 : Paper width in mm's (overrides option 4).
- Option 7 : Paper height in mm's (overrides option 4).
- Option 8 : = 1 → The plot is rotated 90 degrees.

### Maximum viewport size (NDC):

Depending on options 4, 6 and 7.

### Default viewport size:

Depending on options 4, 6 and 7.

### Spot size (fraction of default viewport):

Depending on options 4, 6 and 7.

### Interactive tools:

None

### Retained segments:

None

### Raster facilities:

- Fixed colour table with black/white printer (length 1) and colour vector (length 7), static colour table with colour/monochrome raster (length 255).
- Polygon fill with solid colour.
- 7 (black/white printer) or 49 (monochrome/colour raster) predefined patterns.
- 8 user definable patterns. Max. size (width × height) : 4096

### Special features:

- 8 hardware generated linetypes. Pattern length is by default 5 mm. Specifying a 2-digit linetype *mn* gives linetype *n* with pattern length *m%* of distance between P1 and P2 (see HP manual). Adding 100 to the linetype will draw lines containing one or more complete pattern segments (negative linetypes in HP-GL/2). Linewidth and linepattern scalable.
- Hardware text in all sizes and directions. Shearing available.
- Hardware circle arcs.
- Markers are generated by driver software.

### Miscellaneous:

When device type 0 (black/white printer) is selected, colour indices 2 to 255 may still be used with solid polygon fill to get different shades of gray.



# Appendix F

## Routine Name Index

---

### Some commonly used argument names:

<b>Angle</b>	: Angle in radians.
<b>Dangle</b>	: Angle in degrees.
<b>Ident</b>	: Picture segment identifier.
<b>Idev</b>	: Identifier of a graphic device driver.
<b>lbuff</b>	: Primary buffer for segment storage.
<b>lunit</b>	: Unit number for picture library, range 1 to 99
<b>lsw(tch)</b>	: On/off switch. 1=On, 0=Off.
<b>lvis</b>	: Linetype, range 0 to 255 0=invisible, 1=solid, 2=endpoint, 3=dotted, 4=dashed, 5=dash-dot.
<b>Tmat</b>	: Transformation matrix 4×4 real array.

### Common argument name start/end letters:

<b>lx...</b>	: Absolute Integer coordinates	(e.g. <b>lx</b> , <b>lyarr</b> )
<b>X...</b>	: Absolute Real coordinates	(e.g. <b>X</b> , <b>Yarr</b> , <b>Zpoint</b> )
<b>ldx...</b>	: Relative Integer coordinates	(e.g. <b>ldx</b> , <b>ldzarr</b> )
<b>Dx...</b>	: Relative Real coordinates	(e.g. <b>Dx</b> , <b>Dzarr</b> )
<b>l..arr</b>	: Integer array	(e.g. <b>lxarr</b> , <b>ldxarr</b> )
<b>...arr</b>	: Real array	(e.g. <b>Dxarr</b> , <b>Farr</b> )
<b>...win</b>	: Window coordinate	(e.g. <b>Xwin</b> )
<b>...ndc</b>	: NDC coordinate	(e.g. <b>Xndc</b> )
<b>...usr</b>	: User coordinate	(e.g. <b>Xusr</b> )

With some routines, more than one page number is given. The first one shows where the general description of the routine is found, the second refers to a page where the routine is used for a special purpose.

‘Old’ routines that are still part of the system for compatibility reasons, are shown with a special font, as

COLOUR (lcol)

All arguments that return values to the application program are underlined, as

**AWAIT (Time, ltool)**

Subroutine name (argument list) Description of routine.	Page Number
<b>AUTOX (Mx, My)</b> Specify automatic value or index increment, 2D.	10-2
<b>AUTOX3 (Mx, My, Mz)</b> Specify automatic value or index increment, 3D.	10-2
<b>AWAIT (Time, <u>ltool</u>)</b> Wait for event input.	8-6
<b>AXON (Xeye, Yeye, Zeye)</b> Set eye position for axonometric projection.	6-12
<b>BACDEV (Idev)</b> Specify background device.	19-1
<b>BACDRW</b> Draw copy of retained segments on background device.	19-1
<b>BACVPT (Bvarr(1) )</b> Specify viewport for background device.	19-2
<b>BFACEV (lsw)</b> Specify storage mode for back-facing polygons.	22-7
<b>BGNBTC</b> Begin batch of retained segment operations.	16-3
<b>BGNNAM (lname)</b> Begin named graphic element group.	20-2
<b>BGNPIC (lident)</b> Create (open) a new picture segment.	3-1
<b>BGNTRN</b> Push transformation matrix on internal stack.	6-9
<b>BLICTL (lswtch)</b> Control picture element blinking.	13-2
<b>BLIPIC (lident, lsw)</b> Control picture segment blinking.	17-3
<b>CESCAP (lchar)</b> Specify escape character for format control sequences.	7-2
<b>CFONT (lfont)</b> Select character font.	7-8
<b>CFPROP (lsw)</b> Control proportional character spacing.	7-12

Subroutine name (argument list) Description of routine.	Page Number
CHARA (larr(1), llth) Draw character string, from A1 format data.	7-1
<b>CHARC (Chstri)</b> Draw character string.	7-1
<b>CHARE (Flpno, Length, Lfrac)</b> Draw a floating point number as a string, using E format.	7-3
<b>CHARF (Flpno, Length, Lfrac)</b> Draw a floating point number as a string, using F format.	7-3
<b>CHARI (Intno, Length)</b> Draw an integer number as a string.	7-3
CHARS (larr(1)) Draw character string, from Hollerith data.	7-1
<b>CIRAPR (Dist)</b> Specify circle approximation tolerance.	4-10
<b>CIRC (Xc, Yc, Angle, Ivis)</b> Draw absolute 2D circle arc, with arc given as radians.	4-7
<b>CIRC3 (Xc, Yc, Zc, Xp, Yp, Zp, Angle, Ivis)</b> Draw absolute 3D circle arc, with arc given as radians.	4-7
<b>CIRCR (Dxc, Dyc, Angle, Ivis)</b> Draw relative 2D circle arc, with arc given as radians.	4-7
<b>CIRCR3 (Dxc,Dyc,Dzc, Dxp,Dyp,Dzp, Angle, Ivis)</b> Draw relative 3D circle arc, with arc given as radians.	4-7
<b>CIRD (Xc, Yc, Dangle, Ivis)</b> Draw absolute 2D circle arc, with arc given as degrees.	4-7
<b>CIRD3 (Xc, Yc, Zc, Xp, Yp, Zp, Dangle, Ivis)</b> Draw absolute 3D circle arc, with arc given as degrees.	4-7
<b>CIRDR (Dxc, Dyc, Dangle, Ivis)</b> Draw relative 2D circle arc, with arc given as degrees.	4-7
<b>CIRDR3 (Dxc,Dyc,Dzc, Dxp,Dyp,Dzp, Dangle, Ivis)</b> Draw relative 3D circle arc, with arc given as degrees.	4-7
<b>CJUST (Horiz, Vert)</b> Specify text string alignment.	7-7
<b>CLANG (llang)</b> Select language for character encoding.	7-10

Subroutine name (argument list) Description of routine.	Page Number
<b>CLICTL (lswtch)</b> Set clipping on/off.	2-4
<b>CLRDEV (ldev, lopt)</b> Clear graphic device.	1-2
<b>CLRDWI (ldwi)</b> Clear device window.	21-3
<b>CLRLIB (lunit)</b> Clear and initialize picture library.	14-5
<b>CLSTTY (lunit)</b> Close GPGS-F output file/channel.	D-2
<b>COLOUR (lcol)</b> Set direct colour of picture elements.	11-2
<b>COMP (Tmat(1,1))</b> Multiply user and system transformation matrix.	6-10
<b>COTHLS (Ind1, Hue(1), Rlight(1), Sat(1), Lth)</b> Change colour table, using the HLS colour model.	11-4
<b>COTHSV (Ind1, Hue(1), Sat(1), Val(1), Lth)</b> Change colour table, using the HLS colour model.	11-5
<b>COTIND (Ind)</b> Select colour table index for subsequent picture elements.	11-2
<b>COTRGB (Ind1, Red(1), Green(1), Blue(1), Lth)</b> Change colour table, using the RGB colour model.	11-3
<b>CROTA (Angle)</b> Set character rotation angle in radians.	7-6
<b>CROTAD (Dangle)</b> Set character rotation angle in degrees.	7-6
<b>CSHEA (Shear)</b> Set character shearing factor.	7-5
<b>CSIZEL (Xlett, Ylett)</b> Set letter size in character space.	7-4
<b>CSIZES (Xspace, Yspace)</b> Set character space size.	7-4
<b>CURV (Fx, Fy, Plowl, Uppl, Step, Ivis)</b> Create 2D curve based on absolute functions.	10-4

Subroutine name (argument list) Description of routine.	Page Number
<b>CURV3 (Fx, Fy, Fz, Plowl, Uppl, Step, Ivis)</b> Create 3D curve based on absolute functions.	10-4
<b>CURVR (Dfx, Dfy, Plowl, Uppl, Step, Ivis)</b> Create 2D curve based on relative functions.	10-4
<b>CURVR3 (Dfx, Dfy, Dfz, Plowl, Uppl, Step, Ivis)</b> Create 3D curve based on relative functions.	10-4
<b>DATATR (<u>larr(1)</u>, Lthi, <u>Farr(1)</u>, Lthf)</b> Inquire picture element attribute values.	23-2
<b>DATBUF (<u>larr(1)</u>, Lthi)</b> Inquire primary buffer status.	23-8
<b>DATCBX (Tx, Ty, Strlen, Nlines, <u>Cx</u>, <u>Cy</u>, <u>Bxarr(1)</u>, <u>Byarr(1)</u>)</b> Inquire box enclosing text string.	7-13
<b>DATCHR (<u>larr(1)</u>, Lthi, <u>Farr(1)</u>, Lthf)</b> Inquire character attribute values.	23-3
<b>DATCIR (<u>lmod</u>, <u>Dist</u>)</b> Inquire circle attribute values.	23-3
<b>DATCLI (<u>lswtch</u>)</b> Inquire clipping switch value.	23-3
DATCXA ( <u>larr(1)</u> , llth, <u>Strlen</u> , <u>Nlines</u> ) Inquire text extent, text given in A1 format.	7-13
<b>DATCXC (Chstri, <u>Strlen</u>, <u>Nlines</u>)</b> Inquire text extent.	7-13
DATCXS ( <u>larr(1)</u> , <u>Strlen</u> , <u>Nlines</u> ) Inquire text extent, text given in Hollerith format.	7-13
<b>DATDEV (<u>larr(1)</u>, Lthi, <u>Farr(1)</u>, Lthf)</b> Inquire driver and device data.	23-4, 5-1
<b>DATDNO (<u>ldev</u>)</b> Inquire current active device number.	23-1
<b>DATDWI (<u>ldwi</u>, <u>larr(1)</u>, Lthi, <u>Farr(1)</u>, Lthf)</b> Inquire device window size.	21-12
<b>DATFNU (<u>lfontu</u>, <u>lerru</u>)</b> Inquire Fortran unit numbers used internally.	23-7
<b>DATHID (<u>larr(1)</u>, Lthi)</b> Inquire HLHS module status.	23-7

Subroutine name (argument list) Description of routine.	Page Number
<b>DATLIB (<u>larr</u>(1), <u>Lthi</u>)</b> Inquire picture library status.	23-8
<b>DATMAR (<u>Size</u>)</b> Inquire marker size.	23-4
<b>DATPAR (<u>lpolsz</u>, <u>lcros</u>, <u>lpatsz</u>)</b> Inquire value of installation dependent parameters.	23-7
<b>DATPIX (<u>Xwlow</u>, <u>Ywlow</u>, <u>ldimx</u>, <u>Indarr</u>(1,1), <u>Nx</u>, <u>Ny</u>, <u>Istat</u>)</b> Readback pixel array from device.	12-15
<b>DATPNO (<u>Ident</u>)</b> Inquire identifier of current open picture segment.	23-1
<b>DATPOS (<u>Xpos</u>, <u>Ypos</u>, <u>Zpos</u>)</b> Inquire current position, transformed user coordinates.	23-2
<b>DATUSR (<u>Xusr</u>, <u>Yusr</u>, <u>Zusr</u>)</b> Inquire current position, user coordinates.	23-2
<b>DATVP (<u>Varr3</u>(1))</b> Inquire current viewport limits.	23-2
<b>DATWIN (<u>Warr3</u>(1))</b> Inquire current window limits.	23-2
<b>DEFER (<u>ldefer</u>, <u>laldev</u>)</b> Set deferral mode.	16-2
<b>DELPIC (<u>Ident</u>)</b> Delete picture segment.	14-7, 16-4
<b>DEPCTL (<u>lswtch</u>)</b> Control depth modulation.	13-2
<b>DEVOPT (<u>larr</u>(1), <u>llthi</u>, <u>Rarr</u>(1), <u>llthr</u>, <u>Carr</u>(1), <u>llthc</u>)</b> Set device or device window options.	1-4
<b>DSABLE (<u>ltool</u>)</b> Disable tool for event input.	8-5
<b>ECHCTL (<u>ltool</u>, <u>Istat</u>)</b> Set echo on/off.	8-8
<b>ECHTOL (<u>ltool</u>, <u>larr</u>(1), <u>Lthi</u>, <u>Farr</u>(1), <u>Lthf</u>)</b> Select echo and/or feedback type.	8-8
<b>ECHTXC (<u>ltool</u>, <u>Chstri</u>)</b> Echo a text string.	8-11



Subroutine name (argument list) Description of routine.	Page Number
<b>ECHVP (Itool, Varr(1) )</b> Set echo viewport.	8-8
<b>ELIP (Xcn, Ycn, Xrad, Yrad, Ang0, Angle, Rotang, Ivis)</b> Create a 2D elliptic arc.	4-11
<b>ENABLE (Itool)</b> Enable tool for event input.	8-5
<b>ENDBTC</b> End batch of retained picture segment operations.	16-3
<b>ENDNAM</b> End named graphic element group.	20-2
<b>ENDPIC</b> Close picture segment.	3-1
<b>ENDTRN</b> Pop transformation matrix from internal stack.	6-9
<b>ERFILE (Ifile)</b> Specify output file number for error messages.	24-3
<b>ERROUT (Iarr)</b> System or application supplied error handling routine.	24-4
<b>FLUSHE (Itool)</b> Flush event queue for tool.	8-5
<b>GETBUT (Istat)</b> Get button event report.	8-7
<b>GETHIT (Maxnam, Namarr(1), Lennam)</b> Get pick event report.	8-6
<b>GETLOC (Xndc, Yndc)</b> Get locator event report.	8-7
<b>GETTXC (Chstri, Length)</b> Get text event report.	8-6
<b>GETVAL (Value)</b> Get valuator event report.	8-7
<b>GPGS</b> Initialize the GPGS-F system.	1-1
<b>GPGSOF</b> Close down GPGS-F.	24-5

Subroutine name (argument list) Description of routine.	Page Number
<b>GUFSCS (<u>Iunit</u>, <u>Istat</u>)</b> Close picture library file.	14-6
<b>GUF SOP (<u>Iunit</u>, <u>Fname</u>, <u>Nrec</u>, <u>Fstat</u>, <u>Istat</u>)</b> Open picture library file.	14-5
<b>HIDCTL (<u>Isw</u>)</b> Control storage of polygons/lines in HLHS module.	22-2
<b>HITPOS (<u>Xndc</u>, <u>Yndc</u>)</b> Get pick event position.	8-6
<b>HTCDEF (<u>Index</u>, <u>Angle</u>)</b> Define hatch table entry.	12-8
<b>IDEN</b> Set transformation matrix to identity.	6-1
<b>INPDEV (<u>Idev</u>)</b> Select device for interaction.	8-11
<b>INPDWI (<u>Idwi</u>, <u>Isw</u>)</b> Specify device window(s) to receive input.	21-13
<b>INQDRV (<u>Idev</u>, <u>Istat</u>)</b> Inquire availability of device driver.	1-5
<b>INQDWI (<u>Idwi</u>, <u>Iwsid</u>, <u>Ichg</u>)</b> Inquire device window identifier and change mode.	21-9, 21-10
<b>INQGPV (<u>Ibasv</u>, <u>Isubv</u>)</b> Inquire GPGS-F version numbers.	1-6
<b>INSCOL (<u>Imod</u>)</b> Select colour index of inserted pseudo segment.	15-2
<b>INSERT (<u>Ident</u>)</b> Insert pseudo segment form primary buffer.	15-1
<b>INSHID (<u>lopts(1)</u>, <u>Length</u>)</b> Insert result of HLHS computation.	22-3
<b>INSLIB (<u>Iunit</u>, <u>Ident</u>)</b> Insert pseudo segment from picture library.	15-2
<u>Index</u> = INWAIT (Time, <u>led(1)</u> , <u>Iarr(1)</u> , <u>Lthi</u> , <u>Farr(1)</u> , <u>Lthf</u> ) Wait for input from tool(s).	8-13
<b>LINE (<u>X</u>, <u>Y</u>, <u>Ivis</u>)</b> Create 2D line from absolute floating point coordinates.	4-2

Subroutine name (argument list) Description of routine.	Page Number
<b>LINE3 (X, Y, Z, Ivis)</b> Create 3D line from absolute floating point coordinates.	4-2
<b>LINER (Dx, Dy, Ivis)</b> Create 2D line from relative floating point coordinates.	4-2
<b>LINER3 (Dx, Dy, Dz, Ivis)</b> Create 3D line from relative floating point coordinates.	4-2
<b>LINI (Ix, Iy, Ivis)</b> Create 2D line from absolute integer coordinates.	4-2
<b>LINI3 (Ix, Iy, Iz, Ivis)</b> Create 3D line from absolute integer coordinates.	4-2
<b>LINIR (Idx, Idy, Ivis)</b> Create 2D line from relative integer coordinates.	4-2
<b>LINIR3 (Idx, Idy, Idz, Ivis)</b> Create 3D line from relative integer coordinates.	4-2
<b>LINWID (Wscal)</b> Set linewidth scaling factor.	13-1
<b>LOGERR (Iarr(1))</b> Print error message.	24-5
<b>LPGPGS (Ivis)</b> Set line representation to single stroke line.	9-3
<b>LPHOTD (Ivis, Width, Delta)</b> Set line representation to hot-dog line.	9-4
<b>LPOFFS (Ivis, Offset)</b> Set line representation to offset line.	9-4
<b>LPPARA (Ivis, Width, Delta)</b> Set line representation to parallel lines.	9-3
<b>LPPATT (Ivis, Itypar(1), Rlenar(1), Length)</b> Define line pattern.	9-2
<b>LPSCAL (Scale)</b> Set scaling factor of driver/hardware line pattern.	4-2
<b>LPSCTL (Isw)</b> Control detectability of picture elements.	20-3
<b>LPSET (Cid, Value)</b> Set options for application defined line representation.	9-5

Subroutine name (argument list) Description of routine.	Page Number
<b>LPSPIC (Ident, lsw)</b> Set detectability of picture segment.	20-4
<b>MARKER (Imar)</b> Create centred marker.	4-12
<b>MODTRN (Imod)</b> Set transformation modification mode.	6-16
<b>MOVDWI (ldwi, lxpos, lypos, lmod, lcorn)</b> Set position of device window.	21-7
<b>MSGLEV (llevel)</b> Select error message format.	24-2
<b>MSIZE (Size)</b> Specify marker size.	4-12
<b>NAME (lname)</b> Set name of individual picture element.	20-1
<b>NDCWIN (Xndc, Yndc, Zndc, <u>Xwin</u>, <u>Ywin</u>, <u>Zwin</u>)</b> Convert from NDC to window coordinates.	8-14
<b>NITBUF (larr(1), Length)</b> Define array as primary picture segment buffer.	14-3
<b>NITDEV (ldev)</b> Initialize graphic device.	1-2
<b>NITDWI (ldwi, ltyp, lref)</b> Create new device window, or link to existing window.	21-2
<b>NITHID</b> Initialize HLHS module.	22-2
<b>NITLIB (lunit)</b> Define file as picture library.	14-5
<b>NITOPT (larr(1), llth)</b> Set device options.	1-5
<b><u>lunit</u> = NITTTY (ldev, Filename)</b> Open file/channel for GPGS-F output.	D-2
<b>PATDEF (Index, lcarr(1,1), Nx, Ny)</b> Define pattern table entry.	12-7
<b>PERS (Xeye, Yeye, Zeye)</b> Set eye position for perspective projection.	6-12

Subroutine name (argument list) Description of routine.	Page Number
<b>PIREF (Xref, Yref)</b> Set reference point for patterns and hatch lines.	12-9
<b>PISIZ (Dx, Dy, Hdist)</b> Set pattern size and/or hatch line distance.	12-8
<b>PITYP (lptyp)</b> Select polygon texture type.	12-5
<b>PIXARR (Width, Height, Idimx, Indarr(1,1), Nx, Ny)</b> Create pixel array.	12-12
<b>POLY (Xarr(1), Yarr(1), Lth, ltype)</b> Create 2D polygon from absolute coordinates.	12-3
<b>POLY3 (Xarr(1), Yarr(1), Zarr(1), Lth, ltype)</b> Create 3D polygon from absolute coordinates.	12-3
<b>POLYR (Dxarr(1), Dyarr(1), Lth, ltype)</b> Create 2D polygon from relative coordinates.	12-3
<b>POLYR3 (Dxarr(1), Dyarr(1), Dzarr(1), Lth, ltype)</b> Create 3D polygon from relative coordinates.	12-3
<b>POPDWI (ldwi, lsw)</b> Push/pop device window.	21-6
<b>PRCIND (lnd)</b> Select colour index for polygon perimeter.	12-4
<b>PRIPIC (lident, lpri)</b> Set picture segment priority.	17-3
<b>RDRDWI (ldwi)</b> Redraw all visible segments in device window.	21-10
<b>READWI (ldwi)</b> Inquire device window receiving last input.	21-13
<b>REATOL (ltool, larr(1), Lthi, Farr(1), Lthf)</b> Read additional input data from tool.	8-12
<b>REDRAW</b> Redraw all visible segments.	16-4
<b>REFER (lident)</b> Insert symbolic reference to pseudo segment.	15-5
<b>REQBUT (ltool, lbutno, lstat)</b> Request button tool input.	8-3

Subroutine name (argument list) Description of routine.	Page Number
<b>REQHIT (Itool, Maxnam, <u>Namarr(1)</u>, <u>Lennam</u>)</b> Request pick tool input.	8-3
<b>REQLOC (Itool, <u>Xndc</u>, <u>Yndc</u>)</b> Request locator tool input.	8-2
<b>REQTXC (Itool, <u>Chstri</u>, <u>Length</u>)</b> Request text tool input.	8-3
<b>REQVAL (Itool, <u>Value</u>)</b> Request valuator tool input.	8-3
<b>RESPIC (Idold, Idnew)</b> Copy picture segment from picture library to primary buffer.	14-6
<b>RETAIN (lswtch)</b> Select whether picture segments are to be retained.	16-1
<b>RLSBUF (larr(1))</b> Release primary picture segment buffer.	14-3
<b>RLSDEV (ldev)</b> Release device.	1-2
<b>RLSDWI (ldwi)</b> Release device window.	21-3
<b>RLSLIB (lunit)</b> Release picture library file.	14-6
<b>ROTA (Angle, laxis)</b> Specify rotation, angle in radians.	6-5
<b>ROTAD (Dangle, laxis)</b> Specify rotation, angle in degrees.	6-5
<b>RPADWI (ldwi, lref, lreftp)</b> Reparent device window.	21-8
<b>RSZDWI (ldwi, lwid, lhei, lmod)</b> Resize device window.	21-8
<b>SAVMOD (<u>lswtch</u>)</b> Inquire transformation modification mode.	23-3
<b>SAVPIC (Idold, Idnew)</b> Copy picture segment from primary buffer to picture library.	14-6
<b>SAVTRN (<u>Tmat(1,1)</u>)</b> Save system transformation matrix in application array.	6-10

Subroutine name (argument list) Description of routine.	Page Number
<b>SCAL (Scale, laxis)</b> Specify scaling transformation.	6-4
<b>SELBUF (larr(1))</b> Select primary picture segment buffer.	14-3
<b>SELDEV (ldev)</b> Select output device.	1-2
<b>SELDWI (ldwi)</b> Select output device window.	21-5
<b>SELLIB (lunit)</b> Select picture library.	14-5
<b>SETFNU (lfontu, lerru)</b> Set Fortran unit numbers to be used by GPGS-F.	7-9, 24-4
<b>SHEA (Shear, laxis1, laxis2)</b> Specify shearing transformation.	6-6
<b>SMPBUT (ltool, lbutno, lstat)</b> Sample button tool.	8-5
<b>SMPHIT (ltool, Maxnam, <u>Namarr(1)</u>, <u>Lennam</u>)</b> Sample pick tool.	8-4
<b>SMPLOC (ltool, <u>Xndc</u>, <u>Yndc</u>)</b> Sample locator tool.	8-5
<b>SMPTXC (ltool, <u>Chstri</u>, <u>Length</u>)</b> Sample text tool.	8-4
<b>SMPVAL (ltool, <u>Value</u>)</b> Sample valuator tool.	8-4
<b>SOFCHA (lsw)</b> Select software/hardware text generation.	7-6
<b>SOFCIR (lsw)</b> Select software/hardware circle generation.	4-10
<b>SOFPIX (lsw)</b> Select software/hardware pixel array generation.	12-12
<b>SOFPOL (lqual)</b> Select textured polygon drawing quality.	12-6
<b>TABI (lxarr(1), lyarr(1), Lthi, lvis)</b> Create 2D polyline from absolute integer coordinates.	10-1

Subroutine name (argument list) Description of routine.	Page Number
<b>TABI3 (Ixarr(1), Iyarr(1), Izarr(1), Lthi, Ivis)</b> Create 3D polyline from absolute integer coordinates.	10-1
<b>TABIR (Idxarr(1), Idyarr(1), Lthi, Ivis)</b> Create 2D polyline from relative integer coordinates.	10-1
<b>TABIR3 (Idxarr(1), Idyarr(1), Idzarr(1), Lthi, Ivis)</b> Create 3D polyline from relative integer coordinates.	10-1
<b>TABL (Xarr(1), Yarr(1), Lthi, Ivis)</b> Create 2D polyline from absolute floating point coordinates.	10-1
<b>TABL3 (Xarr(1), Yarr(1), Zarr(1), Lthi, Ivis)</b> Create 3D polyline from absolute floating point coordinates.	10-1
<b>TABLR (Dxarr(1), Dyarr(1), Lthi, Ivis)</b> Create 2D polyline from relative floating point coordinates.	10-1
<b>TABLR3 (Dxarr(1), Dyarr(1), Dzarr(1), Lthi, Ivis)</b> Create 3D polyline from relative floating point coordinates.	10-1
<b>TRAN (Tmat(1,1))</b> Replace system transformation matrix.	6-10
<b>UPDAT (Iregen)</b> Flush retained segment operations.	16-3, 1-5
<b>USRWIN (Xusr, Yusr, Zusr, <u>Xwin</u>, <u>Ywin</u>, <u>Zwin</u>)</b> Convert from user to window coordinates.	8-14
<b>VANS (Xvan, Yvan, Zvan)</b> Set vanishing point for perspective projection.	6-7
<b>VIDEN (Ident)</b> Reset image transformation.	18-1
<b>VISDWI (Idwi, Isw)</b> Set device window visibility.	21-6
<b>VISPIC (Ident, Isw)</b> Set picture segment visibility.	17-1
<b>VPORT (V2arr(1))</b> Set 2D viewport boundaries.	2-3
<b>VPORT3 (V3arr(1))</b> Set 3D viewport boundaries.	2-3
<b>VXLAT (Ident, Xdisp, Ydisp)</b> Set picture segment position, 2D.	18-1



Subroutine name (argument list) Description of routine.	Page Number
<b>VXLAT3 (Ident, Xdisp, Ydisp, Zdisp)</b> Set picture segment position, 3D.	<b>18-1</b>
<b>WINDW (W2arr(1))</b> Set 2D window boundaries.	<b>2-2</b>
<b>WINDW3 (W3arr(1))</b> Set 3D window boundaries.	<b>2-2</b>
<b>WINNDC (Xwin, Ywin, Zwin, <u>Zndc</u>, <u>Yndc</u>, <u>Zndc</u>)</b> Convert from window to NDC coordinates.	<b>8-14</b>
<b>WINUSR (Invert, Xwin, Ywin, Zwin, <u>Xusr</u>, <u>Yusr</u>, <u>Zusr</u>)</b> Convert from window to user coordinates.	<b>8-14</b>
<b>WRITOL (Iarr(1), Lthi, Farr(1), Lthf)</b> Select echo and/or feedback type.	<b>8-13</b>
<b>XLAT (Xdisp, Ydisp)</b> Specify 2D translation.	<b>6-3</b>
<b>XLAT3 (Xdisp, Ydisp, Zdisp)</b> Specify 3D translation.	<b>6-3</b>



# Appendix G

## Routine Number Index

This appendix lists the GPGS-F routine numbers used in short format error messages (described in **Chapter 24**). The routines of the GRAPHISTO and SURRENDER packages are also listed, as well as in the GRAPHISTO and SURRENDER user manuals.

### G.1 GPGS-F Routines

1 NITDEV	26 IDEN	62 PRIPIC	108 CIRC3
2 RLSDEV	27 XLAT	63 BLIPIC	109 CIRC3R
3 SELDEV	28 XLAT3	64 DELPIC	110 CIRC
4 CLRDEV	29 ROTA	65 VISPIC	111 CIRC3R
5 INQDRV	30 SCAL	66 LPSPIC	112 CIRD3
	31 SHEA		113 CIRD3R
6 NITBUF	32 VANS	67 SAVPIC	114 CIRD
7 RLSBUF	33 ROTAD	68 RESPIC	115 CIRD3R
8 SELBUF			
	34 SAVTRN	80 INWAIT	120 TABI3
10 NITLIB	35 BGNTRN	81 REATOL	121 TABIR3
11 RLSLIB	36 ENDTRN	82 WRITOL	122 TABI
12 SELLIB			123 TABIR
13 CLRLIB	37 PERS	90 SOFCHA	124 TABL3
	38 AXON	91 CESCAP	125 TABLR3
14 NITOPT			126 TABL
	39 SAVMOD	93 AUTOX	127 TABLR
15 UPDAT		94 AUTOX3	
16 DEFER	40 VPORT		128 CURV3
17 GPGS	41 VPORT3	96 SOFCIR	129 CURVR3
18 REDRAW		97 CIRAPR	130 CURV
	42 VXLAT		131 CURVR
19 DEVOPT	43 VXLAT3	100 LINI3	
	47 VIDEN	101 LINIR3	132 POLY3
20 WINDW		102 LINI	133 POLYR3
21 WINDW3	50 MSGLEV	103 LINIR	134 POLY
	51 ERFILE	104 LINE3	135 POLYR
22 CLICTL	52 SETFNU	105 LINER3	
	53 INQGPV	106 LINE	
23 MODTRN		107 LINER	
24 TRAN	60 BGNPIC		
25 COMP	61 ENDPIC		

140 CHARS	190 CSIZES	230 GETTXC	300 DATDEV
141 CHARA	191 CSIZEL	231 GETHIT	301 DATDNO
142 CHARI		232 GETVAL	302 DATPNO
143 CHARE	192 CSHEA	233 GETLOC	303 DATBUF
144 CHARF	193 CROTA	234 GETBUT	304 DATLIB
145 CHARC	194 CROTAD	235 HITPOS	305 DATVP
	195 CJUST		306 DATWIN
146 MARKER		236 ENABLE	310 DATATR
	196 CFONT	237 DSABLE	311 DATCIR
160 REFER	197 CLANG	238 AWAIT	312 DATCLI
161 INSERT		239 FLUSHE	313 DATCHR
162 INSLIB	198 MSIZE	240 INPDEV	314 DATCXA
163 INSCOL			315 DATCXS
	199 CFPROP	241 ECHCTL	316 DATCXC
166 RETAIN		242 ECHTOL	317 DATCBX
	200 COTIND	243 ECHTXC	318 DATMAR
167 BGNBTC	201 COTRGB	244 ECHVP	319 DATPOS
168 ENDBTC	202 COTHLS		320 DATUSR
	203 COTHSV	245 READWI	321 DATPIX
170 BACDEV			322 DATHID
171 BACVPT	208 SOFPOL	247 NDCWIN	323 DATPAR
172 BACDRW	209 PATDEF	248 WINNDC	324 DATFNU
	210 HTCDEF	249 WINUSR	
180 LPSCTL		250 USRWIN	400 NITDWI
181 BLICTL	211 PITYP		401 RLSDWI
182 DEPCTL	212 PISIZ	260 NITHID	402 CLRDWI
184 COLOUR	213 PIREF	261 HIDCTL	403 SELDWI
		262 INSHID	
185 NAME	214 SOFPIX	263 BFACEV	404 VISDWI
186 BGNNAM	215 PIXARR		405 RDRDWI
187 ENDNAM		270 LPPATT	406 POPDWI
	216 PRCIND	271 LPGPGS	407 MOVDWI
188 LINWID		272 LPPARA	408 RSZDWI
189 LPSCAL	220 REQTXC	273 LPOFFS	409 RPADWI
	221 REQHIT	274 LPHOTD	
	222 REQVAL	275 LPSET	410 INQDWI
	223 REQLOC		411 DATDWI
	224 REQBUT	280 ELIP	412 INPDWI
	225 SMPTXC		
	226 SMPHIT		
	227 SMPVAL		
	228 SMPLOC		
	229 SMPBUT		

## G.2 GRAPHISTO Routines

600 DEFPAG	628 AUTIND	653 BLNKNG	678 BARGRP
601 DEFAX	629 AUTINC	654 BLGID	
602 DEFAXI		655 BLONOF	679 HACDEN
	630 HEADNG	656 BLAREC	
603 SELAX	631 TABLEG	657 BLAPOL	680 FLBAR
	632 BARLEG	658 BLACIR	681 FLHIS
605 AXANOT	633 FRAME	659 BLAARC	682 FILBAR
606 AXTPOS			
608 AXREXP	634 LEGBOX	660 DEFPIE	683 BAR3D
	635 FILLEG	661 PIESEG	
609 PAGSIZ			684 BARSET
	636 PFRAME	662 FILREC	685 HIGSET
610 AXIS			
	637 PLOCSP	663 BLAFRM	686 CLPPLO
611 AXLAB	638 LSQFIT	664 BLDEL	
612 AXILAB	639 PLOFX		687 HIGDAT
613 AXGLAB		665 PIEDEF	
	640 MIMATB	666 PIESET	688 PLOBAK
614 AXTIT	641 MIMACV		
		667 HACSEG	689 HACREP
615 AXTIC	642 NICLAB	668 FILSEG	
616 AXGTIC	643 NIGLAB		690 TABCHA
	644 NICFMT	669 DATSEG	691 SMOCHA
617 AXGRD	645 NIGFMT		692 HISCHA
618 AXGGRD		670 DEFAXA	693 PIECHA
	646 UNDEF	671 DEFAXX	
619 AXDLAB			694 PLARRW
620 AXMLAB	647 FILTAB	672 AXALAB	695 PLTEXT
	648 HACTAB	673 AXXLAB	
621 PLOTAB			696 DATBLA
622 PLOSMO	649 FATLIN	674 PLBAR	
623 PLOCUR		675 PLHIS	699 MMLENG
624 PLOMAR	650 PLOPAG		
	651 PAGPLO	676 PLOREC	
625 PLOLIN	652 PLOTRN		
		677 PLOSTP	
626 PLOBAR			
627 HACBAR			

## G.3 SURRENDER Routines

700 PLOMA3	710 AXIS3		752 CONSET
	711 AXES3	730 PTSMAT	
701 M3BOX		731 IRRMAT	759 PLOSUP
702 M3CON	712 AXGRD3		
703 M3PTS		740 PLOCON	760 PLOPOC
	713 M3GRID	741 SMOCON	761 PLOPO4
704 PLOMAC	714 M3WALL		
705 PLOMA4	715 M3MRK	743 PLODCN	770 PLOMAD
		744 SMODCN	
706 M3TRN	720 VUFOC		771 M3BOXD
	721 VUAXO	745 LEGTXT	772 M3COND
707 M3DRW	722 VUPER	746 LEGCOL	773 M3DRWD
708 M3LIN	723 VUANG		774 M3GRDD
	724 VUAND	747 POLCON	775 M3LIND
709 M3CON4	725 VUSCA	748 POLDCN	776 M3SPLD
			777 M3WALD
	726 P3DUSR	750 SETOPT	778 M3MRKD
	727 USRP3D		
		751 SRFSET	790 P3TEXT
	728 PTSVIS		

# Appendix H

## C Language Interface

A C language interface to GPGS-F is available as a set of C routines build on top of the Fortran routines.

The names of the C routines are identical to the Fortran routines with a leading **c\_** (e.g. the C routine **c\_nitdev** is the interface to the Fortran routine *NITDEV*).

The argument types used in the C routines varies from the Fortran arguments, as C arguments are transferred by value, while Fortran arguments are transferred by reference. If a C argument is to return a value, a pointer to the variable to receive the value must be given as argument. For arrays there is however no difference, as C defines the value of an array to be the address of its first element (**arr = &arr[0]**). Note that in C, the first array index is 0 (zero), while in Fortran it is 1 (by default), and that in C, two dimensional arrays are stored by rows while in Fortran they are stored by columns.

**Table H.1** *Summary of Fortran and C argument declarations.*

Fortran	C (input only)	C (output)
INTEGER I	int i	int *i
INTEGER I(N)	int i[n]	
INTEGER I(M,N)	int i[n][m]	
LOGICAL L	int l	int *l
REAL X	float x	float *x
REAL X(N)	float x[n]	
REAL X(N,M)	float x[n][m]	
CHARACTER*N C	char c[n] <b>or</b> char *c	
CHARACTER*N C(M)	char *c[m]	

**Note!** The table above is correct only for computers where type **int** in C is the same number of bits as the default **INTEGER** type in Fortran.

If not, **short int** or **long int** is used within the C routines.

As shown by the table above, when a Fortran routine expects an argument of type **logical**, a variable of type **int** must be used in C. As the internal representation of a Fortran logical is not the same on all computers, the C interface of GPGS-F uses the following definition:

- If the value supplied by C is zero, GPGS-F will receive a **FALSE** value.
- If C supplies any other value than zero, GPGS-F will receive a **TRUE** value.

### Example H.1    *A Fortran and C application program.*

Fortran	C
REAL W(4)	static float w[4]
DATA W/0.0, 0.5, 0.0, 0.5)	= {0.0, 0.5, 0.0, 0.5};
	float xw, yw, zw, xd, yd;
CALL GPGS	c_gpgs();
CALL NITDEV(72)	c_nitdev(72);
CALL WINDW(W)	c_windw(w);
CALL BGNPIC(1)	c_bgnpic(1);
CALL COTIND(2)	c_cotind(2);
CALL LINE(0.1, 0.1, 0)	c_line(0.1, 0.1, 0);
CALL CHARC('Testing')	c_charc("Testing");
CALL REQLOC(201, XD, YD)	c_reqloc(201, &xd, &yd);
CALL NDCWIN(XD,YD,0.0, XW,YW,ZW)	c_ndcwin(xd,yd,0.0, &xw,&yw,&zw);
CALL LINE(XW, YW, 0)	c_line(xw, yw, 0);
CALL MARKER(8)	c_marker(8);
CALL ENDPIC	c_endpic();
CALL RLSDEV(72)	c_rlsdev(72);

The rest of this appendix contains a list of all C interface routines to GPGS-F, with argument declarations. (The C interface routines to GRAPHISTO and SURRENDER are described in the GRAPHISTO User's Manual and SURRENDER User's Manual.)

---

```
void c_autox(mx, my)
int          mx, my;


---


void c_autox3(mx, my, mz)
int          mx, my, mz;


---


void c_await(time, itool)
float        time;
int          *itool;


---


void c_axon(xeye, yeye, zeye)
float        xeye, yeye, zeye;


---


void c_bacdev(idev)
int          idev;


---


void c_bacdrw()


---


void c_bacvpt(bvarr)
float        *bvarr;


---


void c_bfacev(isw)
int          isw;


---


void c_bgnnam(iname)
int          iname;


---


void c_bgnpic(ident)
int          ident;


---


void c_bgntrn()


---


void c_blictl(iswtch)
int          iswtch;


---


void c_blipic(ident, isw)
int          ident, isw;
```

---



<b>void c_cescap(ichar)</b>
int                ichar;
<b>void c_cfont(ifont)</b>
int                ifont;
<b>void c_cfprop(isw)</b>
int                isw;
<b>void c_charc(chstri)</b>
char                *chstri;
<b>void c_chare(flpno, length, lfrac)</b>
float               flpno;
int                        length, lfrac;
<b>void c_charf(flpno, length, lfrac)</b>
float               flpno;
int                        length, lfrac;
<b>void c_chari(intno, length)</b>
int                   intno, length;
<b>void c_cirapr(dist)</b>
float                dist;
<b>void c_circ(xc, yc, angle, ivis)</b>
float                xc, yc, angle;
int                        ivis;
<b>void c_circ3(xc, yc, zc, xp, yp, zp, angle, ivis)</b>
float                xc, yc, zc, xp, yp, zp, angle;
int                                        ivis;
<b>void c_circr(dxc, dyc, angle, ivis)</b>
float                dxc, dyc, angle;
int                        ivis;
<b>void c_circr3(dxc, dyc, dzc, dxp, dyp, dzp, angle, ivis)</b>
float                dxc, dyc, dzc, dxp, dyp, dzp, angle;
int                                        ivis;
<b>void c_cird(xc, yc, dangle, ivis)</b>
float                xc, yc, dangle;
int                        ivis;
<b>void c_cird3(xc, yc, zc, xp, yp, zp, dangle, ivis)</b>
float                xc, yc, zc, xp, yp, zp, dangle;
int                                        ivis;
<b>void c_cirdr(dxc, dyc, dangle, ivis)</b>
float                dxc, dyc, dangle;
int                        ivis;
<b>void c_cirdr3(dxc, dyc, dzc, dxp, dyp, dzp, dangle, ivis)</b>
float                dxc, dyc, dzc, dxp, dyp, dzp, dangle;
int                                        ivis;
<b>void c_cjust(horiz, vert)</b>
float                horiz, vert;
<b>void c_clang(ilang)</b>
int                        ilang;
<b>void c_clictl(iswtch)</b>
int                        iswtch;

---

```

void c_clrdev(idev, iopt)
int          idev, iopt;


---


void c_clrdwi(idwi)
int          idwi;


---


void c_clrlib(iunit)
int          iunit;


---


void c_clstty(iunit)
int          iunit;


---


void c_comp(tmat)
float        *tmat;


---


void c_cothls(indl, hue,  rlight,  sat, lth)
int          indl,                lth;
float        *hue, *rlight, *sat;


---


void c_cothsv(indl, hue,  sat,  val, lth)
int          indl,                lth;
float        *hue, *sat, *val;


---


void c_cotind(ind)
int          ind;


---


void c_cotrgb(indl, red,  green,  blue, lth)
int          indl,                lth;
float        *red, *green, *blue;


---


void c_crota(angle)
float        angle;


---


void c_crotad(dangle)
float        dangle;


---


void c_cshea(shear)
float        shear;


---


void c_csize1(xlett, ylett)
float        xlett, ylett;


---


void c_csize2(xspace, yspace)
float        xspace, yspace;


---


void c_curv(fx, fy, plowl, uppl, step, ivis)
float        (*fx) ();
float        (*fy) ();
float        plowl, uppl, step;
int          ivis;


---


void c_curv3(fx, fy, fz, plowl, uppl, step, ivis)
float        (*fx) ();
float        (*fy) ();
float        (*fz) ();
float        plowl, uppl, step;
int          ivis;


---


void c_curvr(dfy, dfy, plowl, uppl, step, ivis)
float        (*dfx) ();
float        (*dfy) ();
float        plowl, uppl, step;
int          ivis;

```

---

---

```

void c_curvr3(dfx, dfy, dfz, plowl, uppl, step, ivis)
float      (*dfx) ();
float      (*dfy) ();
float      (*dfz) ();
float      plowl, uppl, step;
int      ivis;

```

---

```

void c_datatr(iarr, lthi, farr, lthf)
int      *iarr, lthi, lthf;
float      *farr;

```

---

```

void c_datbuf(iarr, lthi)
int      *iarr, lthi;

```

---

```

void c_datcbx(tx, ty, strlen, nlines, cx, cy, bxarr, byarr)
float      tx, ty, strlen, *cx, *cy, *bxarr, *byarr;
int      nlines;

```

---

```

void c_datchr(iarr, lthi, farr, lthf)
int      *iarr, lthi, lthf;
float      *farr;

```

---

```

void c_datcir(imod, dist)
int      *imod;
float      *dist;

```

---

```

void c_datcli(iswtch)
int      *iswtch;

```

---

```

void c_datcxc(chstri, strlen, nlines)
char      *chstri;
float      *strlen;
int      *nlines;

```

---

```

void c_datdev(iarr, lthi, farr, lthf)
int      *iarr, lthi, lthf;
float      *farr;

```

---

```

void c_datdno(idev)
int      *idev;

```

---

```

void c_datdwi(idwi, iarr, lthi, farr, lthf)
int      idwi, *iarr, lthi, lthf;
float      *farr;

```

---

```

void c_datfnu(ifontu, ierru)
int      *ifontu, *ierru;

```

---

```

void c_dathid(iarr, lthi)
int      *iarr, lthi;

```

---

```

void c_datlib(iarr, lthi)
int      *iarr, lthi;

```

---

```

void c_datmar(size)
float      *size;

```

---

```

void c_datpar(ipolsz, icros, ipatsz)
int      *ipolsz, *icros, *ipatsz;

```

---

```

void c_datpix(xwlol, ywlol, idimx, indarr, nx, ny, istat)
float      xwlol, ywlol;
int      idimx, *indarr, nx, ny, *istat;

```

---

```

void c_datpno(ident)
int      *ident;

```

---

---

```

void c_datpos(xpos, ypos, zpos)
float      *xpos, *ypos, *zpos;


---


void c_datusr(xusr, yusr, zusr)
float      *xusr, *yusr, *zusr;


---


void c_datvp(varr3)
float      *varr3;


---


void c_datwin(warr3)
float      *warr3;


---


void c_defer(idefer, ialdev)
int        idefer, ialdev;


---


void c_delpic(ident)
int        ident;


---


void c_depctl(iswtch)
int        iswtch;


---


void c_devopt(iarr, ilthi, rarr, ilthr, carr, ilthc)
int        *iarr, ilthi,      ilthr,      ilthc;
float      *rarr;
char      **carr;


---


void c_dsable(itool)
int        itool;


---


void c_echctl(itool, istat)
int        itool, istat;


---


void c_echtol(itool, iarr, lthi, farr, lthf)
int        itool, *iarr, lthi,      lthf;
float      *farr;


---


void c_echtxc(itool, chstri)
int        itool;
char      *chstri;


---


void c_echvp(itool, varr)
int        itool;
float      *varr;


---


void c_elip(xcn, ycn, xrad, yrad, ang0, angle, rotang, ivis)
float      xcn, ycn, xrad, yrad, ang0, angle, rotang;
int        ivis;


---


void c_enable(itool)
int        itool;


---


void c_endnam()


---


void c_endpic()


---


void c_endtrn()


---


void c_erfile(ifile)
int        ifile;


---


void c_flush(itool)
int        itool;


---


void c_getbut(istat)
int        *istat;

```

---

---

```
void c_gethit(maxnam, namarr, lennam)
int          maxnam, *namarr, *lennam;

```

---

```
void c_getloc(xndc, yndc)
float        *xndc, *yndc;

```

---

```
void c_gettxc(chstri, length)
char         *chstri;
int          *length;

```

---

```
void c_getval(value)
float        *value;

```

---

```
void c_gpgs()

```

---

```
void c_gpgsof()

```

---

```
void c_gufsc1(iunit, istat)
int          iunit, *istat;

```

---

```
void c_gufsop(iunit, fname, nrec, fstat, istat)
int          iunit,      nrec,      *istat;
char         *fname,     *fstat;

```

---

```
void c_hidctl(isw)
int          isw;

```

---

```
void c_hitpos(xndc, yndc)
float        *xndc, *yndc;

```

---

```
void c_htcdef(index, angle)
int          index;
float        angle;

```

---

```
void c_iden()

```

---

```
void c_inpdev(idev)
int          idev;

```

---

```
void c_inpdwi(idwi, isw)
int          idwi, isw;

```

---

```
void c_inqdrv(idev, istat)
int          idev, *istat;

```

---

```
void c_inqdwi(idwi, iwsid, ichg)
int          idwi, *iwsid, *ichg;

```

---

```
void c_inqgpv(ibasv, isubv)
int          *ibasv, *isubv;

```

---

```
void c_inscol(imod)
int          imod;

```

---

```
void c_insert(ident)
int          ident;

```

---

```
void c_inshid(iopts, length)
int          *iopts, length;

```

---

```
void c_inslib(iunit, ident)
int          iunit, ident;

```

---

```
void c_line(x, y, ivis)
float        x, y;
int          ivis;

```

---

---

```
void c_line3(x, y, z, ivis)
float      x, y, z;
int        ivis;

```

---

```
void c_liner(dx, dy, ivis)
float      dx, dy;
int        ivis;

```

---

```
void c_liner3(dx, dy, dz, ivis)
float      dx, dy, dz;
int        ivis;

```

---

```
void c_lini(ix, iy, ivis)
int        ix, iy, ivis;

```

---

```
void c_lini3(ix, iy, iz, ivis)
int        ix, iy, iz, ivis;

```

---

```
void c_linir(idx, idy, ivis)
int        idx, idy, ivis;

```

---

```
void c_linir3(idx, idy, idz, ivis)
int        idx, idy, idz, ivis;

```

---

```
void c_linwid(wscal)
float      wscal;

```

---

```
void c_lpgpgs(ivis)
int        ivis;

```

---

```
void c_lphotd(ivis, width, delta)
int        ivis;
float      width, delta;

```

---

```
void c_lpooffs(ivis, offset)
int        ivis;
float      offset;

```

---

```
void c_lppara(ivis, width, delta)
int        ivis;
float      width, delta;

```

---

```
void c_lppatt(ivis, itypar, rlenar, length)
int        ivis, *itypar,      length;
float      *rlenar;

```

---

```
void c_lpscal(scale)
float      scale;

```

---

```
void c_lpsctl(isw)
int        isw;

```

---

```
void c_lpset(cid, value)
char       *cid;
float      value;

```

---

```
void c_lpspic(ident, isw)
int        ident, isw;

```

---

```
void c_marker(imar)
int        imar;

```

---

```
void c_modtrn(imod)
int        imod;

```

---

---

```

void c_movdwi(idwi, ixpos, iypos, imod, icorn)
int          idwi, ixpos, iypos, imod, icorn;

```

---

```

void c_msglev(ilevel)
int          ilevel;

```

---

```

void c_msize(size)
float        size;

```

---

```

void c_name(iname)
int          iname;

```

---

```

void c_ndcwin(xndc, yndc, zndc, xwin, ywin, zwin)
float        xndc, yndc, zndc, *xwin, *ywin, *zwin;

```

---

```

void c_nitbuf(iarr, length)
int          *iarr, length;

```

---

```

void c_nitdev(idev)
int          idev;

```

---

```

void c_nitdwi(idwi, ityp, iref)
int          idwi, ityp, iref;

```

---

```

void c_nithid()

```

---

```

void c_nitlib(iunit)
int          iunit;

```

---

```

int c_nittty(idev, filename)
int          idev;
char         *filename;

```

---

```

void c_patdef(index, icarr, nx, ny)
int          index, *icarr, nx, ny;

```

---

```

void c_pers(xeye, yeye, zeye)
float        xeye, yeye, zeye;

```

---

```

void c_piref(xref, yref)
float        xref, yref;

```

---

```

void c_pisiz(dx, dy, hdist)
float        dx, dy, hdist;

```

---

```

void c_pityp(iptyp)
int          iptyp;

```

---

```

void c_pixarr(width, height, idimx, indarr, nx, ny)
float        width, height;
int          idimx, *indarr, nx, ny;

```

---

```

void c_poly(xarr, yarr, lth, itype)
float        *xarr, *yarr;
int          lth, itype;

```

---

```

void c_poly3(xarr, yarr, zarr, lth, itype)
float        *xarr, *yarr, *zarr;
int          lth, itype;

```

---

```

void c_polyr(dxarr, dyarr, lth, itype)
float        *dxarr, *dyarr;
int          lth, itype;

```

---

```

void c_polyr3(dxarr, dyarr, dzarr, lth, itype)
float        *dxarr, *dyarr, *dzarr;
int          lth, itype;

```

---

---

```

void c_popdwi(idwi, isw)
int          idwi, isw;

void c_prcind(ind)
int          ind;

void c_pripic(ident, ipri)
int          ident, ipri;

void c_rdrdwi(idwi)
int          idwi;

void c_readwi(idwi)
int          *idwi;

void c_reatol(itool, iarr, lthi, farr, lthf)
int          itool, *iarr, lthi,          lthf;
float                               *farr;

void c_redraw()

void c_refer(ident)
int          ident;

void c_reqbut(itool, ibutno, istat)
int          itool, *ibutno, *istat;

void c_reghit(itool, maxnam, namarr, lennam)
int          itool, maxnam, *namarr, *lennam;

void c_reqloc(itool, xndc, yndc)
int          itool;
float                               *xndc, *yndc;

void c_reqtxc(itool, chstri, length)
int          itool,          *length;
char          *chstri;

void c_reqval(itool, value)
int          itool;
float          *value;

void c_respic(idold, idnew)
int          idold, idnew;

void c_retain(iswtch)
int          iswtch;

void c_rlsbuf(iarr)
int          *iarr;

void c_rlsdev(idev)
int          idev;

void c_rlsdwi(idwi)
int          idwi;

void c_rlslib(iunit)
int          iunit;

void c_rota(angle, iaxis)
float          angle;
int          iaxis;

void c_rotad(dangle, iaxis)
float          dangle;
int          iaxis;

```

---



---

```
void c_rpadwi(idwi, iref, ireftp)
```

```
int          idwi, iref, ireftp;
```

---

```
void c_rszdwi(idwi, iwid, ihei, imod)
```

```
int          idwi, iwid, ihei, imod;
```

---

```
void c_savmod(iswtch)
```

```
int          *iswtch;
```

---

```
void c_savpic(idold, idnew)
```

```
int          idold, idnew;
```

---

```
void c_savtrn(tmat)
```

```
float        *tmat;
```

---

```
void c_scal(scale, iaxis)
```

```
float        scale;
```

```
int          iaxis;
```

---

```
void c_selbuf(iarr)
```

```
int          *iarr;
```

---

```
void c_seldev(idev)
```

```
int          idev;
```

---

```
void c_seldwi(idwi)
```

```
int          idwi;
```

---

```
void c_sellib(iunit)
```

```
int          iunit;
```

---

```
void c_setfnu(ifontu, ierru)
```

```
int          ifontu, ierru;
```

---

```
void c_shea(shear, iaxis1, iaxis2)
```

```
float        shear;
```

```
int          iaxis1, iaxis2;
```

---

```
void c_smpbut(itool, ibutno, istat)
```

```
int          itool, *ibutno, *istat;
```

---

```
void c_smphit(itool, maxnam, namarr, lennam)
```

```
int          itool, maxnam, *namarr, *lennam;
```

---

```
void c_smploc(itool, xndc, yndc)
```

```
int          itool;
```

```
float        *xndc, *yndc;
```

---

```
void c_smptxc(itool, chstri, length)
```

```
int          itool, *length;
```

```
char        *chstri;
```

---

```
void c_smpval(itool, value)
```

```
int          itool;
```

```
float        *value;
```

---

```
void c_sofcha(isw)
```

```
int          isw;
```

---

```
void c_sofcir(isw)
```

```
int          isw;
```

---

```
void c_sofpix(isw)
```

```
int          isw;
```

---

```
void c_sofpol(igual)
```

```
int          igual;
```

---

---

```

void c_tabi(ixarr, iyarr, lthi, ivis)
int      *ixarr, *iyarr, lthi, ivis;

```

---

```

void c_tabi3(ixarr, iyarr, izarr, lthi, ivis)
int      *ixarr, *iyarr, *izarr, lthi, ivis;

```

---

```

void c_tabir(idxarr, idyarr, lthi, ivis)
int      *idxarr, *idyarr, lthi, ivis;

```

---

```

void c_tabir3(idxarr, idyarr, idzarr, lthi, ivis)
int      *idxarr, *idyarr, *idzarr, lthi, ivis;

```

---

```

void c_tabl(xarr, yarr, lthi, ivis)
float      *xarr, *yarr;
int      lthi, ivis;

```

---

```

void c_tabl3(xarr, yarr, zarr, lthi, ivis)
float      *xarr, *yarr, *zarr;
int      lthi, ivis;

```

---

```

void c_tablr(dxarr, dyarr, lthi, ivis)
float      *dxarr, *dyarr;
int      lthi, ivis;

```

---

```

void c_tablr3(dxarr, dyarr, dzarr, lthi, ivis)
float      *dxarr, *dyarr, *dzarr;
int      lthi, ivis;

```

---

```

void c_tran(tmat)
float      *tmat;

```

---

```

void c_updat(iregen)
int      iregen;

```

---

```

void c_usrwin(xusr, yusr, zusr, xwin, ywin, zwin)
float      xusr, yusr, zusr, *xwin, *ywin, *zwin;

```

---

```

void c_vans(xvan, yvan, zvan)
float      xvan, yvan, zvan;

```

---

```

void c_viden(ident)
int      ident;

```

---

```

void c_visdwi(idwi, isw)
int      idwi, isw;

```

---

```

void c_vispic(ident, isw)
int      ident, isw;

```

---

```

void c_vport(v2arr)
float      *v2arr;

```

---

```

void c_vport3(v3arr)
float      *v3arr;

```

---

```

void c_vxlat(ident, xdisp, ydisp)
int      ident;
float      xdisp, ydisp;

```

---

```

void c_vxlat3(ident, xdisp, ydisp, zdisp)
int      ident;
float      xdisp, ydisp, zdisp;

```

---

```

void c_window(w2arr)
float      *w2arr;

```

---

---

```
void c_windw3(w3arr)
```

```
float          *w3arr;
```

---

```
void c_winndc(xwin, ywin, zwin, xndc, yndc, zndc)
```

```
float          xwin, ywin, zwin, *xndc, *yndc, *zndc;
```

---

```
void c_winusr(invert, xwin, ywin, zwin, xusr, yusr, zusr)
```

```
int            invert;
```

```
float          xwin, ywin, zwin, *xusr, *yusr, *zusr;
```

---

```
void c_xlat(xdisp, ydisp)
```

```
float          xdisp, ydisp;
```

---

```
void c_xlat3(xdisp, ydisp, zdisp)
```

```
float          xdisp, ydisp, zdisp;
```



# Appendix I

## Error Messages

<b>Basic GPGS-F error messages</b>	
<b>Error Number</b>	<b>Message Description</b>
<b>1</b>	<b>End of medium - buffer limit exceeded.</b> No more space in primary picture segment buffer.
<b>2</b>	<b>Buffer not initialized or buffer already initialized.</b> Attempt to initialize ( <i>NITBUF</i> ) the same picture buffer twice, or attempt to select ( <i>SELBUF</i> ) or release ( <i>RLSBUF</i> ) a buffer that is not initialized.
<b>3</b>	<b>No current buffer.</b> Attempt to create a pseudo or retained segment, but no picture buffer has been initialized.
<b>4</b>	<b>No device.</b> Graphical output generated, but there is no current device.
<b>5</b>	<b>Undefined device facility requested.</b> Driver cannot perform requested operation.
<b>6</b>	<b>Segment reference not allowed for this device.</b> Current device must be the pseudo device when <i>REFER</i> is used.
<b>7</b>	<b>Picture segment open.</b> A routine requiring that there is no open segment, (e.g. <i>VPORT</i> , <i>SELDEV</i> ) is called while a segment is open.
<b>8</b>	<b>No picture segment open.</b> Graphic output generated, but there is no picture segment open.
<b>9</b>	<b>Picture segment does not exist.</b> Reference to a non existing picture segment.
<b>10</b>	<b>Picture segment already exists.</b> Attempt to create ( <i>BGNPIC</i> ) or copy ( <i>RESPIC</i> / <i>SAVPIC</i> ) a segment, using an identifier that is already used.
<b>11</b>	<b>Illegal value of argument.</b> Argument is outside its allowable range.
<b>12</b>	<b>Illegal area.</b> Upper limit lower than lower limit, or similar.

<b>Basic GPGS-F error messages (cont.)</b>	
<b>Error Number</b>	<b>Message</b> Description
<b>13</b>	<b>Illegal device number.</b> Attempt to initialize a device ( <i>NITDEV</i> ) using an illegal device number, or clear / select / release a device that is not initialized.
<b>14</b>	<b>No background device has been selected.</b> <i>BACDRW</i> called, but there is no background device.
<b>15</b>	<b>Undefined arithmetic operation.</b> Error in floating data.
<b>16</b>	<b>Stack error.</b> Too many nested calls or unmatched calls.
<b>17</b>	<b>String error.</b> Illegal character or control command.
<b>18</b>	<b>Too large array.</b>
<b>19</b>	<b>System error.</b> Contact system people.
<b>21</b>	<b>System resources exceeded.</b> Some table has overflowed.
<b>22</b>	<b>Recursive picture segment reference in pseudo picture.</b>
<b>23</b>	<b>Driver error.</b> Contact system people.
<b>25</b>	<b>Illegal command.</b> Illegal code found in inserted pseudo segment.
<b>26</b>	<b>Inconsistent end of picture segment.</b> Error when fetching pseudo segment. May be caused by a picture buffer being over-written.
<b>27</b>	<b>Buffer inconsistency.</b> Buffer administration area destroyed.

<b><i>Error messages from GPGS-F file system (picture library routines)</i></b>	
<b>Error Number</b>	<b>Message Description.</b>
<b>30</b>	<b>Library not initialized.</b> Referring to a picture library (e.g. <i>INSLIB</i> , <i>SELLIB</i> ) that has not been initialized by <i>NITLIB</i> .
<b>31</b>	<b>Library already initialized.</b> Attempt to initialize the same library twice.
<b>32</b>	<b>Too many libraries open.</b> A maximum of 4 libraries may be open at the same time.
<b>33</b>	<b>No current library.</b> <i>DATLIB</i> , <i>RESPIC</i> or <i>SAVPIC</i> , but there is no current library.
<b>34</b>	<b>File access error.</b> GPGS-F could not read or write the library file.
<b>35</b>	<b>Unexpected I/O error.</b> Contact system people.
<b>36</b>	<b>Library not properly closed.</b> Inconsistency in library file, probably because it was not closed the previous time it was used.
<b>37</b>	<b>Segment table full, can not store more segments.</b>
<b>38</b>	<b>Attempt to write past end of file.</b>
<b>39</b>	<b>Library file not compatible with current GPGS-F version.</b> The program building the library must be linked with the latest GPGS-F version and run.

### ***Error messages from interaction routines***

<b>Error Number</b>	<b>Message</b> Description.
<b>40</b>	<b>No tool active - infinite wait.</b>
<b>41</b>	<b>Illegal class number.</b> Input tool does not exist
<b>42</b>	<b>Too many tools enabled for event input.</b>
<b>44</b>	<b>Tool enabled for event input.</b> Tool used in request or sample mode, or tool to be enabled is already set to event mode.
<b>45</b>	<b>Tool not enabled for event input.</b> Tool to be disabled is not enabled.
<b>46</b>	<b>Current event report is of illegal type.</b> The data in the current event report is not of the requested type.
<b>47</b>	<b>No current event.</b> Event report data requested, but there is no current event.

### ***Error messages from raster routines***

<b>Error Number</b>	<b>Message</b> Description.
<b>50</b>	<b>Singular transformation matrix.</b> Impossible to invert the matrix.
<b>51</b>	<b>Too many vertices in hatched or patterned polygon.</b>
<b>52</b>	<b>Degenerate polygon.</b> Impossible to compute the polygon plane normal.
<b>53</b>	<b>Too complex polygon.</b> Impossible to keep intersections between a line and polygon vertices, internal table overflow.
<b>54</b>	<b>Internal error when scanconverting the polygon.</b> Contact system people.



<b>Error messages from GRAPHISTO</b>	
<b>Error Number</b>	<b>Message</b> Description.
<b>61</b>	<b>Missing definition of axis.</b> Both axes must be defined before plotting the first axis, and before any data plotting (curves, bars, etc.).
<b>62</b>	<b>Illegal area dimension.</b> Illegal definition of page or plot area, e.g. lower value specified larger than upper value.
<b>63</b>	<b>No current axis.</b> Attempt to add axis annotation, but there is no current axis.
<b>64</b>	<b>No previous axis.</b> Attempt to draw an axis next to the previous, but there is no previous axis.
<b>65</b>	<b>Illegal angle for axis title.</b> Axis title must be parallel or perpendicular to the axis.
<b>66</b>	<b>Pie chart not defined.</b> Attempt to plot a pie segment before the pie is defined.
<b>67</b>	<b>Too many protected areas.</b> Storing capacity of the blanking module is exceeded. Either too many protected areas are defined, or the total number of points added to the module is too large.
<b>68</b>	<b>Illegal option name.</b> The text string given to select an option did not match any of the legal choices.
<b>69</b>	<b>One axis must be discrete.</b> Attempt to use one of the bar plotting routines requiring a discrete axis, but both axes have been defined to be linear.

<b><i>Error messages from SURRENDER</i></b>	
<b>Error Number</b>	<b>Message</b> Description.
<b>71</b>	<b>Too small working array.</b> Working array supplied to SURRENDER surface or contour plotting routine is too small.
<b>72</b>	<b>No viewing transformation present.</b> Attempt to reuse transformation, but no viewing transformation is defined.
<b>73</b>	<b>Illegal (sub)area specified.</b> Either the data set size specified with a surface or contour plotting routine is too small, or the subarea selected by an option setting routine is illegal (lower limit larger than upper limit).
<b>74</b>	<b>No 3D surface present.</b> Attempt to add annotation to a surface plot, but no surface has yet been defined.
<b>75</b>	<b>Singular matrix for inversion.</b> Transformation matrix is singular. May occur with surface plots when the Z axis is to be automatically defined, and the difference between the lower and upper Z value is very small compared to the axes ranges specified in X and Y direction.
<b>76</b>	<b>Undefined points mismatch.</b> With the <i>PLOMA4</i> and <i>PLOPO4</i> routines, there must be a strict correspondence between undefined points in the two data sets, i.e. a given node must either be defined in both data sets, or undefined in both.

<b><i>Error messages from font routines</i></b>	
<b>Error Number</b>	<b>Message</b> Description.
<b>80</b>	<b>Error when opening font file.</b>
<b>81</b>	<b>Error when reading from font file.</b>
<b>82</b>	<b>Error when closing font file.</b>

<b><i>Error messages from HLHS module</i></b>	
<b>Error Number</b>	<b>Message Description.</b>
<b>90</b>	<b>HLHS module has not been initialized.</b> Attempt to use the HLHS module before it has been initialized.
<b>91</b>	<b>Too many polygons added to the HLHS module.</b>
<b>92</b>	<b>Too many vertices added to the HLHS module.</b>
<b>93</b>	<b>Too many vertices in a polygon added to the HLHS module.</b>
<b>94</b>	<b>Too complex polygon in HLHS module.</b> Impossible to keep all intersections between two polygons, internal table overflow.
<b>95</b>	<b>Internal error when splitting a polygon.</b> Contact system people.
<b>96</b>	<b>Too many lines added to the HLHS module.</b>

<b><i>Error messages from routines handling multi window devices</i></b>	
<b>Error Number</b>	<b>Message Description.</b>
<b>100</b>	<b>Device window does not exist.</b>
<b>101</b>	<b>Device window already exists.</b>
<b>102</b>	<b>Illegal device window.</b> Attempt to close window 1.
<b>103</b>	<b>Parent device window does not exist.</b>
<b>104</b>	<b>Illegal reparent command.</b> Attempt to reparent a top level window.
<b>105</b>	<b>Illegal device window identifier.</b>
<b>106</b>	<b>Illegal operation for this type of device window.</b>
<b>108</b>	<b>Referenced device window does not exist.</b>



# Keyword Index

An index of GPGS-F subroutines is found in **Appendix F**.

Keywords are given in **Keyword-in-Context** format.

## A

character **Alignment** 7-7  
 inquire current character **Alignment** 23-4  
     circle **Arc** drawing 4-7  
     circle **Arc** smoothness 4-10  
 inquire current circle **Arc** smoothness 23-3  
     elliptic **Arc** drawing 4-11  
     pixel **Array**, *see* **Pixel array**  
 inquire character **Attributes** 23-3  
     inquire circle **Attributes** 23-3  
     picture element **Attributes** 13-1 to 13-2, 20-3  
 inquire picture element **Attributes** 23-2  
 retained segment **Attributes** 17-1 to 17-4, 20-4  
     **Automatic** index increment 10-2, 12-3  
     **Automatic** value increment 10-2, 10-5, 12-3  
     **Axonometric** projection 6-12

## B

**Background** colour 11-1, 11-2  
 copy retained segments to **Background** device 19-1, 21-13  
     select **Background** device 19-1  
     viewport of **Background** device 19-2  
 HLHS module and **Back-facing** polygons 22-7  
     picture element **Blinking** 13-2  
 inquire current picture element **Blinking** mode 23-2  
     retained segment **Blinking** 17-3  
 insert pseudo segment from **Buffer** 15-1  
     primary **Buffer** for segment storage 14-3  
     copy picture segment to **Buffer** from library 14-6  
     copy picture segment from **Buffer** to library 14-6  
     inquire segment **Buffer** status 23-8  
     **Buffer** / pick simulation module 14-2, 16-1, 17-1, 20-5, E-4  
     get **Button** event report 8-7  
     echo selection for **Button** input 8-10  
     request **Button** input 8-3  
     sample **Button** input 8-5

## C

- C language interface **H-1 to H-13**
- Character**, *see also* **Text**
- Character** alignment **7-7**
- inquire current **Character** alignment **23-4**
- inquire **Character** attributes **23-3**
- Character** drawing **7-1**
- Character** encoding **7-10**
- inquire current **Character** font **23-3**
- Character** font selection **7-8**
- software **Character** fonts **B-2 to B-9**
- inquire number of hardware **Character** fonts **23-5**
- Character** format control **7-2**
- Character** language **7-10**
- inquire current **Character** language **23-3**
- Character** rotation **7-6**
- inquire current **Character** rotation **23-4**
- software / hardware **Character** selection **7-6**
- inquire current software / hardware **Character** selection **23-3**
- Character** shearing **7-5**
- inquire current **Character** shearing factor **23-4**
- Character** size **7-4**
- inquire current **Character** size **23-3**
- Character** spacing **7-4**
- inquire **Circle** attributes **23-3**
- inquire hardware **Circle** capability **23-6**
- Circle** drawing **4-7**
- software / hardware **Circle** selection **4-10**
- inquire current software / hardware **Circle** selection **23-3**
- Circle** smoothness **4-10**
- inquire current **Circle** smoothness **23-3**
- Clear** device **1-2**
- Clear** device window **21-3**
- Clear** picture library **14-5**
- inquire hardware **Clipping** method **23-5**
- Clipping** mode **2-4**
- inquire current **Clipping** mode **23-3**
- Clipping** pseudo segment **15-4**
- Close** GPGS-F **24-5**
- Close** picture library file **14-6**
- Close** picture segment **3-1**
- background **Colour** **11-1, 11-2**
- polygon perimeter **Colour** **12-4**
- Colour** index **11-1**
- Colour** index selection **11-2**

inquire current	<b>Colour</b> index	<b>23-2</b>
inquire maximum	<b>Colour</b> index	<b>23-6</b>
	HLS <b>Colour</b> model	<b>11-4</b>
	HSV <b>Colour</b> model	<b>11-5</b>
	RGB <b>Colour</b> model	<b>11-3</b>
	<b>Colour</b> of inserted pseudo segment	<b>15-2</b>
	<b>Colour</b> table	<b>11-1</b>
default	<b>Colour</b> table	<b>11-6</b>
define	<b>Colour</b> table	<b>11-2</b> to <b>11-6</b>
inquire	<b>Colour</b> table type	<b>23-6</b>
	<b>Coordinate</b> conversion	<b>8-14</b>
	<b>Coordinate</b> processing	<b>2-4</b> to <b>2-5</b>
	<b>Coordinate</b> rotation	<b>6-4</b>
	<b>Coordinate</b> scaling	<b>6-4</b>
	<b>Coordinate</b> shearing	<b>6-6</b>
	<b>Coordinate</b> translation	<b>6-3</b>
normalized device	<b>Coordinates</b> (NDC)	<b>2-2</b>
user	<b>Coordinates</b>	<b>2-1</b>
window	<b>Coordinates</b>	<b>2-1</b>
	<b>Copy</b> picture segment	<b>14-6</b>
	<b>Copy</b> retained segments to background device	<b>19-1, 21-13</b>
	<b>Create</b> device window	<b>21-2</b>
	<b>Create</b> picture segment	<b>3-1</b>
inquire	<b>Current</b> active device	<b>23-1</b>
inquire	<b>Current</b> character alignment	<b>23-4</b>
inquire	<b>Current</b> character font	<b>23-3</b>
inquire	<b>Current</b> character language	<b>23-3</b>
inquire	<b>Current</b> character rotation	<b>23-4</b>
inquire	<b>Current</b> character shearing factor	<b>23-4</b>
inquire	<b>Current</b> character size	<b>23-3</b>
inquire	<b>Current</b> circle smoothness	<b>23-3</b>
inquire	<b>Current</b> clipping mode	<b>23-3</b>
inquire	<b>Current</b> colour index	<b>23-2</b>
inquire	<b>Current</b> depth modulation mode	<b>23-2</b>
	<b>Current</b> device	<b>1-2</b>
	<b>Current</b> device window	<b>21-5</b>
inquire	<b>Current</b> escape character	<b>23-3</b>
inquire	<b>Current</b> linewidth scaling factor	<b>23-2</b>
inquire	<b>Current</b> marker size	<b>23-4</b>
inquire	<b>Current</b> picture element blinking mode	<b>23-2</b>
inquire	<b>Current</b> picture element detectability	<b>23-2</b>
inquire	<b>Current</b> picture segment	<b>23-1</b>
	<b>Current</b> position	<b>4-1</b>
inquire	<b>Current</b> position	<b>23-2</b>
	<b>Current</b> position with circles	<b>4-8</b>

**Current** position with curves 10-4  
**Current** position with elliptic arcs 4-11  
**Current** position with lines 4-3  
**Current** position with markers 4-12  
**Current** position with pixel arrays 12-12  
**Current** position with polygons 12-3  
**Current** position with polylines 10-2  
**Current** position with text 7-1  
inquire **Current** software / hardware character selection 23-3  
inquire **Current** software / hardware circle selection 23-3  
inquire **Current** transformation mode 23-3  
inquire **Current** viewport limits 23-2  
inquire **Current** window limits 23-2  
**Curve** drawing 10-4

## D

**Deferral** mode 16-2  
inquire number of **Definable** hatch styles 23-6  
inquire number of **Definable** patterns 23-6  
**Define** colour table 11-2 to 11-6  
**Define** line representation 9-3 to 9-5  
**Define** linetype 9-2  
**Define** linewidth 9-3 to 9-4  
**Define** polygon hatch style 12-8  
**Define** polygon pattern 12-7  
**Delete** pseudo segment 14-7  
**Delete** retained segment 16-4  
polygon hatch **Density** 12-8  
**Depth** modulation 13-2  
inquire current **Depth** modulation mode 23-2  
picture element **Detectability** 20-3  
inquire current picture element **Detectability** 23-2  
retained segment **Detectability** 20-4  
alternate input **Device** 8-11  
copy retained segments to background **Device** 19-1, 21-13  
current **Device** 1-2  
inquire current active **Device** 23-1  
select background **Device** 19-1  
update **Device** 16-3  
viewport of background **Device** 19-2  
**Device** control 1-2  
normalized **Device** coordinates (NDC) 2-2  
inquire **Device** data 23-4 to 23-6  
**Device** driver 1-1  
list of **Device** drivers E-3



Device options 1-4  
inquire Device size 5-1, 23-4  
inquire Device type 23-6  
Device window, *see* device Window  
Draw circle arc 4-7  
Draw curve 10-4  
Draw elliptic arc 4-11  
Draw in true scale 5-1  
Draw line 4-2  
Draw marker 4-12  
Draw numbers 7-3  
Draw pixel array 12-12  
Draw polygon 12-3  
Draw polyline 10-1  
Draw text 7-1  
device Driver 1-1  
flush Driver output buffer 1-5, 16-3  
list of device Drivers E-3  
inquire Drivers linked 1-5

## E

Echo specification 8-8 to 8-10  
input Echo viewport (area) 8-8  
picture Element 4-1, 13-1  
picture Element attributes 13-1 to 13-2, 20-3  
inquire picture Element attributes 23-2  
picture Element blinking 13-2  
inquire current picture Element blinking mode 23-2  
picture Element depth modulation 13-2  
inquire current picture Element depth modulation mode 23-2  
picture Element detectability 20-3  
inquire current picture Element detectability 23-2  
picture Element linewidth scaling factor 13-1  
inquire current picture Element linewidth scaling factor 23-2  
picture Element namestack 20-1 to 20-3  
inquire maximum picture Element namestack length 23-4  
Elliptic arc drawing 4-11  
character Encoding 7-10  
selective Erase 12-1, 16-2, 17-1  
Error data file number 24-4  
default Error data file number A-2  
inquire Error data file number 23-7  
application supplied Error handling routine 24-4  
Error message file 24-3  
Error message format 24-2

**Error** messages   **I-1** to **I-7**  
    **Escape** character for format control   **7-2**  
    inquire current **Escape** character   **23-3**  
    **Escape** functions interaction tool   **8-10**  
    **Event** mode input   **8-4** to **8-7**  
    get **Event** report from queue   **8-6**  
    flush **Event** reports   **8-5**  
    inquire text **Extent**   **7-13**  
    **Eye** position for viewing   **6-12**

## **F**

    close picture library **File**   **14-6**  
    error message **File**   **24-3**  
    open picture library **File**   **14-5**  
    default error data **File** number   **A-2**  
    error data **File** number   **24-4**  
    inquire error data **File** number   **23-7**  
    GPGS-F input / output **File** number   **1-4, D-1**  
    inquire **File** numbers   **23-7**  
    default font **File** unit number   **A-2**  
    font **File** unit number   **7-9**  
    inquire font **File** unit number   **23-7**  
    **Flush** driver output buffer   **1-5, 16-3**  
    **Flush** event reports   **8-5**  
    **Focal** point   **6-14**  
    character **Font** selection   **7-8**  
    inquire current character **Font**   **23-3**  
    **Fontfile** unit number   **7-9**  
    default **Fontfile** unit number   **A-2**  
    inquire **Fontfile** unit number   **23-7**  
    inquire number of hardware **Fonts**   **23-5**  
    software **Fonts**   **B-2** to **B-9**  
    error message **Format**   **24-2**  
    character **Format** control   **7-2**

## **G**

**Graphic** element (primitive), *see* picture **Element**  
    **GRAPHISTO**   **C-2**  
    specify **Grey** level   **11-6**

## **H**

**Hardware** character selection   **7-6**  
    inquire **Hardware** circle capability   **23-6**  
    **Hardware** circle selection   **4-10**

- inquire **Hardware** clipping method    23-5
- inquire number of **Hardware** fonts    23-5
  - Hardware** pixel array selection    12-12
  - Hardware** polygon selection    12-6
- inquire **Hardware** text capability    23-5
- Hatch** density    12-8
- select **Hatch** polygon texture    12-5
- define polygon **Hatch** style    12-8
  - polygon **Hatch** style definition, summary    12-9
  - polygon **Hatch** style reference point    12-9
  - polygon **Hatch** style table    12-4, 12-7
- inquire number of definable **Hatch** styles    23-6
- predefined polygon **Hatch** styles    12-8
  - Hidden** lines and surfaces, *see* **HLHS** module
  - HLHS** module    22-1 to 22-8
- storing back-facing polygons in **HLHS** module    22-7
  - using dummy device with **HLHS** module    22-6
  - HLHS** module limitations    22-8
- inquire **HLHS** module limits    23-7
- inquire **HLHS** module utilization    23-7
- HLS** colour model    11-4
- HSV** colour model    11-5

## I

- device window **Identifier**    21-9
- picture segment **Identifier**    3-1, 14-2
  - Image** transformations    18-1
  - inquire **Image** transformations capability    23-5
- automatic index **Increment**    10-2, 12-3
- automatic value **Increment**    10-2, 10-5, 12-3
  - colour **Index**    11-1
- inquire current colour **Index**    23-2
- inquire maximum colour **Index**    23-6
  - polygon hatch style **Index**    12-4, 12-7
  - polygon pattern **Index**    12-4, 12-7
- automatic **Index** increment    10-2, 12-3
  - colour **Index** selection    11-2
- Initialize** device    1-2
- Initialize** device window    21-2
- Initialize** GPGS-F    1-1
- Initialize** picture library    14-5
- Initialize** segment buffer    14-3
- event mode **Input**    8-4 to 8-7
- request mode **Input**    8-2 to 8-4
- sample mode **Input**    8-4 to 8-5

GPGS-F **Input** / output file number 1-4, D-1  
additional **Input** data 8-12  
alternate **Input** device 8-11  
    **Input** echo specification 8-8 to 8-10  
    **Input** from device window 21-13  
    **Input** tool numbers 8-1  
    **Insert** pseudo segment from buffer 15-1  
    **Insert** pseudo segment from library 15-2  
colour of **Inserted** pseudo segment 15-2  
    **Installation** dependent parameters A-1 to A-3  
inquire **Installation** dependent parameters 23-7  
    **Interaction**, *see* **Input**  
polygon **Interior** style 12-4

## L

character **Language** 7-10  
inquire current character **Language** 23-3  
insert pseudo segment from **Library** 15-2  
    close picture **Library** file 14-6  
    open picture **Library** file 14-5  
    copy picture segment to **Library** from buffer 14-6  
    copy picture segment from **Library** to buffer 14-6  
    picture **Library** segment storage 14-4  
inquire picture **Library** status 23-8  
    **Line** drawing 4-2  
    **Line** representation 9-3 to 9-5  
    **Linepattern** scaling 4-2  
hidden **Lines** and surfaces, *see* **HLHS** module  
    define **Linetype** 9-2  
predefined **Linetypes** 4-1  
    **Linewidth** 9-3  
    **Linewidth** scaling factor 13-1  
inquire current **Linewidth** scaling factor 23-2  
    get **Locator** event report 8-7  
echo selection for **Locator** input 8-10  
    request **Locator** input 8-2  
    sample **Locator** input 8-5

## M

**Marker** drawing 4-12  
    **Marker** size 4-12  
inquire current **Marker** size 23-4  
transformation **Matrix**, *see* **Transformation** matrix  
    error **Message** file 24-3

error **Message** format 24-2  
error **Messages** I-1 to I-7  
    **MICRO-GPGS-F** C-1  
    **Modelling** transformations 6-1 to 6-17  
depth **Modulation** 13-2  
inquire current depth **Modulation** mode 23-2  
    **Monochrome** devices 11-6

## N

**NDC** (normalized device coordinates) 2-2  
picture element **Namestack** 20-1 to 20-3  
inquire maximum **Namestack** length 23-4  
convert window to **NDC** coordinates 8-14  
    convert **NDC** to window coordinates 8-14  
    **Normalized** device coordinates (NDC) 2-2  
draw **Numbers** as text 7-3

## O

**Open** device window 21-2  
    **Open** picture library file 14-5  
    **Open** picture segment 3-1  
device **Options** 1-4  
device window **Options** 21-1, 21-3  
flush driver **Output** buffer 1-5, 16-3  
GPGS-F input / **Output** file number 1-4, D-1

## P

installation dependent **Parameters** A-1 to A-3  
inquire installation dependent **Parameters** 23-7  
    define line **Pattern** 9-2  
    define polygon **Pattern** 12-7  
        polygon **Pattern** definition, summary 12-9  
        select **Pattern** polygon texture 12-5  
        polygon **Pattern** reference point 12-9  
        polygon **Pattern** size 12-8  
        polygon **Pattern** table 12-4, 12-7  
inquire number of definable **Patterns** 23-6  
predefined polygon **Patterns** 12-7  
    polygon **Perimeter** colour 12-4  
    **Perspective** projection 6-12  
    get **Pick** event report 8-6  
    **Pick** input 20-1 to 20-7  
echo selection for **Pick** input 8-9  
    request **Pick** input 8-3

- sample **Pick** input 8-4
  - Pick** input and pseudo segments 20-7
- buffer / **Pick** simulation module 14-2, 16-1, 17-1, 20-5, E-4
  - Picture** element, *see* picture **Element**
  - Picture** mode transformations 6-16
  - Picture** segment, *see* **Segment**
- inquire **Pixel array** capability 23-6
  - Pixel array** drawing 12-12
  - Pixel array** readback 12-15
- software / hardware **Pixel array** selection 12-12
  - Polygon** definition 12-2
  - Polygon** drawing 12-3
  - Polygon** hatch density 12-8
  - Polygon** hatch reference point 12-9
  - define **Polygon** hatch style 12-8
  - inquire number of definable **Polygon** hatch styles 23-6
    - Polygon** hatch style definition, summary 12-9
    - Polygon** hatch style table 12-4, 12-7
  - predefined **Polygon** hatch styles 12-8
    - Polygon** interior style 12-4
    - define **Polygon** pattern 12-7
    - inquire number of definable **Polygon** patterns 23-6
      - Polygon** pattern definition, summary 12-9
      - Polygon** pattern reference point 12-9
      - Polygon** pattern size 12-8
      - Polygon** pattern table 12-4, 12-7
    - predefined **Polygon** patterns 12-7
      - Polygon** perimeter colour 12-4
  - software / hardware **Polygon** selection 12-6
    - 3D **Polygon** texture 12-11
      - Polygon** texture quality 12-5
      - Polygon** texture selection 12-5
    - Polyline** drawing 10-1
  - current **Position** 4-1
  - inquire current **Position** 23-2
  - device window **Position** 21-7
  - retained segment **Position** 18-1
    - current **Position** with circles 4-8
    - current **Position** with curves 10-4
    - current **Position** with elliptic arcs 4-11
    - current **Position** with lines 4-3
    - current **Position** with markers 4-12
    - current **Position** with pixel arrays 12-12
    - current **Position** with polygons 12-3
    - current **Position** with polylines 10-2

- current **Position** with text 7-1
- Predefined** colour table 11-6
- Predefined** linetypes 4-1
- Predefined** polygon hatch styles 12-8
- Predefined** polygon patterns 12-7
- graphic **Primitive**, *see* picture **Element**
- inquire number of segment **Priorities** allowed 23-6
- retained segment **Priority** 17-3
- axonometric **Projection** 6-12
- perspective **Projection** 6-12
- Proportional** spacing 7-12
- Pseudo** segment 14-1, 15-1
- clipping **Pseudo** segment 15-4
- colour of inserted **Pseudo** segment 15-2
- copy **Pseudo** segment 14-6
- delete **Pseudo** segment 14-7
- insert **Pseudo** segment from buffer 15-1
- insert **Pseudo** segment from library 15-2
- Pseudo** segment reference 15-4
- pick input and **Pseudo** segments 20-7

## R

- Redraw** all retained segments 16-4
- Redraw** retained segments of device window 21-10
- pseudo segment **Reference** 15-4
- polygon pattern / hatch **Reference** point 12-9
- Release** device 1-2
- Release** device window 21-3
- Release** picture library 14-6
- Release** segment buffer 14-3
- hidden lines and surfaces **Removal**, *see* **HLHS** module
- Reparent** device window 21-8
- flush event **Reports** 8-5
- Request** mode input 8-2 to 8-4
- Retained** segment 14-2, 16-1
- copy **Retained** segment 14-6
- delete **Retained** segment 16-4
- Retained** segment attributes 17-1 to 17-4, 20-4
- Retained** segment blinking 17-3
- Retained** segment detectability 20-4
- Retained** segment position 18-1
- Retained** segment priority 17-3
- inquire **Retained** segment storage method 23-5
- Retained** segment visibility 17-1
- device windows and **Retained** segments 21-10

- redraw all **Retained** segments 16-4
- redraw **Retained** segments of device window 21-10
- copy **Retained** segments to background device 19-1, 21-13
- RGB** colour model 11-3
- character **Rotation** 7-6
- inquire current character **Rotation** 23-4
- coordinate **Rotation** 6-4

## S

- Sample mode input 8-4 to 8-5
- draw in true **Scale** 5-1
- coordinate **Scaling** 6-4
- linepattern **Scaling** 4-2
- clipping pseudo **Segment** 15-4
- close picture **Segment** 3-1
- colour of inserted pseudo **Segment** 15-2
- copy picture **Segment** 14-6
- delete pseudo **Segment** 14-7
- delete retained **Segment** 16-4
- inquire current picture **Segment** 23-1
- open picture **Segment** 3-1
  - pseudo **Segment** 14-1, 15-1
  - retained **Segment** 14-2, 16-1
  - retained **Segment** attributes 17-1 to 17-4, 20-4
  - retained **Segment** blinking 17-3
  - inquire **Segment** buffer status 23-8
  - picture **Segment** classes 14-1
  - picture **Segment** definition 14-1
  - retained **Segment** detectability 20-4
- insert pseudo **Segment** from buffer 15-1
- insert pseudo **Segment** from library 15-2
  - picture **Segment** identifier 3-1, 14-2
  - picture **Segment** introduction 3-1
  - retained **Segment** position 18-1
- inquire number of **Segment** priorities allowed 23-6
- retained **Segment** priority 17-3
- pseudo **Segment** reference 15-4
- picture **Segment** storage 14-2
- picture library **Segment** storage 14-4
- primary buffer for **Segment** storage 14-3
- inquire retained **Segment** storage method 23-5
- retained **Segment** visibility 17-1
- pick input and pseudo **Segments** 20-7
- redraw all retained **Segments** 16-4
- retained **Segments** and device window 21-10



- redraw retained **Segments** of device window 21-10
- copy retained **Segments** to background device 19-1, 21-13
  - Select** device 1-2
  - Select** device window 21-5
  - Select** picture library 14-5
  - Select** segment buffer 14-3
  - Set** transformation matrix 6-10
- character **Shearing** 7-5
- inquire current character **Shearing** factor 23-4
- coordinate **Shearing** 6-6
- buffer / pick **Simulation** module 14-2, 16-1, 17-1, 20-5, E-4
- character **Size** 7-4
- inquire current character **Size** 23-3
- device window **Size** 21-8
- inquire device window **Size** 21-12
- inquire device **Size** 5-1, 23-4
- marker **Size** 4-12
- inquire current marker **Size** 23-4
- polygon pattern **Size** 12-8
- circle **Smoothness** 4-10
- inquire current circle **Smoothness** 23-3
- Software** character selection 7-6
- Software** circle selection 4-10
- Software** fonts B-2 to B-9
- Software** pixel array selection 12-12
- Software** polygon selection 12-6
- Space** mode transformations 6-16
- character **Spacing** 7-4
- proportional **Spacing** 7-12
- device window **Stacking** order 21-6
- picture segment **Storage** 14-2
- inquire retained segment **Storage** method 23-5
- hidden lines and **Surfaces**, *see* **HLHS** module
- SURRENDER** C-4

## T

- colour **Table** 11-1
- default colour **Table** 11-6
- define colour **Table** 11-2 to 11-6
- polygon hatch style **Table** 12-4, 12-7
- polygon pattern **Table** 12-4, 12-7
- inquire colour **Table** length 23-6
- inquire colour **Table** type 23-6
- Text**, *see also* **Character**
- draw numbers as **Text** 7-3

inquire hardware **Text** capability 23-5  
    get **Text** event report 8-6  
    inquire **Text** extent 7-13  
echo selection for **Text** input 8-9  
    request **Text** input 8-3  
    sample **Text** input 8-4  
proportional **Text** spacing 7-12  
3D polygon **Texture** 12-11  
    polygon **Texture** quality 12-5  
    polygon **Texture** selection 12-5  
    input **Tool** numbers 8-1  
combine **Transformation** matrices 6-10  
    set **Transformation** matrix 6-10  
system **Transformation** matrix 6-1  
restore **Transformation** matrix from stack 6-9  
    save **Transformation** matrix in user array 6-10  
    save **Transformation** matrix on stack 6-9  
    reset **Transformation** matrix to identity 6-1  
        **Transformation** mode 6-16  
inquire current **Transformation** mode 23-3  
    image **Transformations** 18-1  
inquire image **Transformations** capability 23-5  
    modelling **Transformations** 6-1 to 6-17  
    coordinate **Translation** 6-3  
retained segment **Translation** 18-1  
    draw in **True** scale 5-1

## U

**Update** device 16-3  
inquire device window **Update** state 21-10  
    **User** coordinates 2-1  
convert window to **User** coordinates 8-14  
    convert **User** to window coordinates 8-14

## V

    get **Valuator** event report 8-7  
echo selection for **Valuator** input 8-9  
    request **Valuator** input 8-3  
    sample **Valuator** input 8-4  
    automatic **Value** increment 10-2, 10-5, 12-3  
        **Vanishing** point 6-7, 6-16  
inquire GPGS-F **Version** number 1-6  
    eye position for **Viewing** 6-12  
    input echo **Viewport** 8-8

Viewport definition 2-2  
inquire current Viewport limits 23-2  
window to Viewport mapping 2-1 to 2-3, 21-2  
Viewport of background device 19-2  
inquire maximum Viewport size 23-4  
default Viewport values 2-3  
device window Visibility 21-6  
retained segment Visibility 17-1

## W

line Width 9-3  
line Width scaling factor 13-1  
inquire current line Width scaling factor 23-2  
device Window 21-1  
current device Window 21-5  
input from device Window 21-13  
redraw retained segments of device Window 21-10  
reparent device Window 21-8  
device Window and background device 21-13  
device Window and retained segments 21-10  
device Window and window to viewport mapping 21-2  
Window coordinates 2-1  
convert NDC to Window coordinates 8-14  
convert user to Window coordinates 8-14  
device Window identifier 21-9  
inquire current Window limits 23-2  
device Window management 21-2 to 21-5  
device Window options 21-1, 21-3  
device Window position 21-7  
device Window size 21-8  
inquire device Window size 21-12  
device Window stacking order 21-6  
convert Window to NDC coordinates 8-14  
convert Window to user coordinates 8-14  
Window to viewport mapping 2-1 to 2-3, 21-2  
inquire device Window update state 21-10  
default Window values 2-3  
device Window visibility 21-6  
inquire number of device Windows available 23-6