The IRAF Simple Graphics Interface (SGI)

Doug Tody

National Optical Astronomy Observatories*
August 1986

ABSTRACT

The IRAF Simple Graphics Interface (SGI) provides a straightforward way of interfacing batch plotter devices such as laser printers and raster or pen plotters to the IRAF graphics i/o subsystem (GIO) to provide high quality graphics hardcopy. To minimize the knowledge of the IRAF system internals required to interface a new device, the device interface is implemented as a user written host Fortran or C program taking as input an SGI format metacode or raster plot file written by the IRAF graphics subsystem. To simplify the task of the host system program (and hence of the programmer who writes it), as well as to minimize the size and complexity of the interface, as much work as possible is done in the high level, device independent SGI kernel. The result is a surprisingly clean and efficient interface to which new devices can typically be added in a matter of hours, starting from the code for an existing SGI device interface.

June 6, 1990

*Operated by the Association of Universities for Research in Astronomy, Inc. under cooperative agreement with the National Science Foundation.

The IRAF Simple Graphics Interface (SGI)

Doug Tody

National Optical Astronomy Observatories*
August 1986

1. Introduction

The IRAF graphics subsystem defines three classes of graphics devices: interactive vector graphics devices, e.g., graphics terminals or bit-mapped graphics workstations (class `STDGRAPH`), semi-interactive image display devices (class `STDIMAGE`), and noninteractive or batch plotter devices (class `STDPLOT`). Our concern here is with the `STDPLOT` class of graphics devices, the output only, medium to hiqh quality plotter devices. The typical `STDPLOT` device is a laser printer, raster plotter, or pen plotter.

The SGI interface is not the only plotter interface provided by IRAF. Other plotter interfaces currently provided by IRAF are the *calcomp kernel*, useful when a Calcomp compatible plotter library is available, and the *nspp kernel*, useful when the NCAR graphics software is available on the host machine. Additional interfaces (GIO kernels) are planned for the *CGI*, *GKS*, and *Postscript* graphics standards. The principal advantage of the SGI interface is that it is vastly simpler than these alternative interfaces as well as highly efficient. In cases where the plotter device in question is not already supported on the local host by one of the standard graphics libraries, it will probably be much easier to build an SGI dispose task than to add a new device driver to the standard graphics library.

2. Overview

The Simple Graphics Interface (SGI) allows any plotter device to be interfaced to IRAF with a minimum amount of work, and with minimum knowledge of the IRAF system internals. This is achieved by having the IRAF graphics system do virtually all the work, producing as output a simple format *plot file* containing the information required to produce one or more graphics frames. A user supplied host system *dispose task* is then called to dispose of the output to a specific host system plotter device.

The architecture and dataflow of the SGI interface in the normal runtime multiprocess configuration is shown in Figure 1. Graphics output destined for an SGI device interface starts out as *GKI metacode* produced by an IRAF applications program running as a subprocess of the CL. The GKI metacode may be spooled in a metafile as is shown in the figure, but normally it is sent by the applications subprocess directly to the SGI kernel by writing to one of the IPC pseudofile streams (usually `STDPLOT` for a plotter device), with the IPC data being routed through the CL.

When `STDPLOT` becomes active the CL will fetch the graphcap entry for the plotter device specified by the user, e.g., the value of the environment variable *stdplot*. The graphcap entry for a plotter device contains an entry (*kf*) specifying the filename of executable containing the graphics kernel to be run. In our case the graphics kernel is the SGI kernel (`bin$x_sgikern.e`), a conventional portable and device independent GIO graphics kernel.

---

*Operated by the Association of Universities for Research in Astronomy, Inc. under cooperative agreement with the National Science Foundation.

The SGI kernel takes as input GKI metacode, translating all high level graphics and text drawing instructions into simple pen-up pen-down moves, writing the SGI plot file as output. When the maximum frame count has been reached, when the user logs out of the CL, or when the command *gflush* is entered, the SGI kernel issues a command to the host system to dispose of the plot file to the plotter. The host command interpreter executes the dispose command, calling the user supplied **dispose task** to dispose of the plot file to the plotter. The plot file is then deleted, either by the SGI kernel or by the dispose task itself.

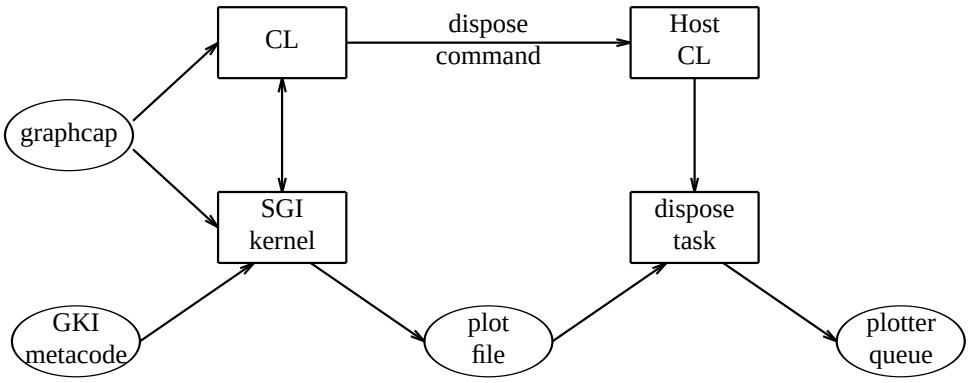

Figure 1. SGI Architecture and Dataflow

The SGI kernel can produce two types of plot files: **metacode files**, wherein all text and vector drawing instructions have been reduced to simple pen up and pen down drawing commands, and **raster files**, simple two dimensional boolean arrays (bitmaps). Metacode files are used to drive intelligent devices such as laser printers, non-raster devices such as pen plotters, and to interface to host graphics libraries such as the Calcomp and PLOT10 libraries. Raster files are used to drive raster devices, such as the Printronix, Trilog, Versatec, and so on.

Fortunately, one does not have to understand all this in any detail to implement an SGI interface for a new device. To interface a new graphics device via SGI one must [1] add an entry for the new SGI device to the IRAF *graphcap* (graphics device capability) file, and [2] write and test the host level SGI dispose program. The dispose task, a conventional host dependent Fortran or C program, is typically only several hundred lines of code and can usually be written by adapting the code for an existing SGI device. The graphcap entry defines the physical and control parameters for the device, and is typically only half a dozen lines of text.

The SGI interface is not called simple because the IRAF graphics subsystem is simple (it isn't), but because the interface defined by the SGI plot file is so small and well defined. Once the dispose task functions correctly when run manually on a test SGI plot file, it can be expected to run correctly in the multi-process configuration described above as well.

3. The SGI Graphcap Entry

All graphics devices supported by IRAF must have an entry in the graphics device capability file, `dev$graphcap`. Often a **physical device** will have more than one entry, e.g., when the device is supported by more than one GIO graphics kernel. The **logical device name** specified by the user determines which graphcap entry, and therefore which GIO graphics kernel, will be used to process the GKI metacode output by the applications program. The graphcap file format and usage is discussed in more detail in the paper *Graphics I/O Design*, but for completeness we will present the essential concepts here as well.

The graphcap entry for a device contains two distinct classes of device parameters. The parameters with lower case names, e.g., 'kf', 'xr', and so on, are the **generic device parameters**. The generic parameters are defined for all graphics devices, with default values if omitted from the graphcap entry for a device. The generic parameters are used to control the device independent part of the graphics system. The parameters with upper case names, e.g., 'DD', 'PX', and so on, are the **kernel control parameters**. These are defined only for one particular type of graphics kernel, and are used to control the device dependent operation of that kernel. Often the same kernel parameter will be used by two different kernels in a slightly different way, creating a possible source of confusion.

Of the generic parameters, only two are absolutely required for a kernel to function (`kf` and `tn`). A couple more generic parameters are required for text to come out the right size. Several more are usually added to define the physical characteristics of the plotter. The recommended minimum set of generic parameters for an SGI device are shown in Figure 2.

parameter	*recommended default*	*description*
ch	0.0294	character height in NDC units
cw	0.0125	character width in NDC units
kf	bin$x_sgikern.e	executable containing kernel task
tn	sgikern	name of the kernel task to be run
xr,yr	-	device resolution in X and Y (pixels)
xs,ys	-	device scale in X and Y (meters)
zr	1	device resolution in Z (greylevels)

Figure 2. Generic graphcap parameters for an SGI device

The SGI kernel parameters fall into two subclasses, those parameters which pertain to both metacode and raster output, and those used only for raster output. There are currently no special parameters for controlling the format of metacode output (since there is only one format). The parameters common to both metacode and bitmap devices are defined in Figure 3.

The function of most of the SGI control parameters should be self explanatory. Since the graphcap entry is read at runtime, new values for any of these parameters take effect immediately (unless the graphcap entry is cached somewhere), making it easy to tune the graphcap entry for a device.

The most critical parameter in an SGI graphcap entry is the DD parameter, the device-dispose control string. The DD entry has the following three fields:

device, plotfile, dispose_command

The *device* field should be the logical device name; it is required as a placeholder but is not actually used at present. The *plotfile* field is the root name for the temporary plot file; the SGI kernel will use this to construct a unique temporary file name. This is normally set to `tmp$sgk`, causing the plot files to be placed in the IRAF logical directory `TMP`. Lastly, the *dispose_command* field is the command to be sent to the host system command interpreter to dispose of the plot file to the plotter queue. The dispose command can be almost anything; some examples will be given later. The sequence **$F** appearing anywhere in the dispose command string is replaced by the host equivalent of the computer generated plot file name before the dispose command is executed.

DB	have the kernel print debug messages during execution
DD	host command to dispose of metacode file ($F)
FE	output a frame instruction at end of plotfile
FS	output a frame instruction at start of plotfile
MF	multiframe count (max frames per job)
NF	store each frame in a new file (rather than all in one file)
RM	boolean; if present, SGK will delete plot files
RO	rotate plot (swap x and y)
YF	y-flip plot (flip y axis) (done after rotate)

Figure 3. SGI kernel control parameters

The integer parameter MF sets a limit on the maximum number of graphics frames to be buffered by the kernel before the output is disposed of. Typical values are 1, 8, or 16. If the boolean parameter NF is present in the graphcap entry for the device each frame will be stored in a separate file, otherwise all frames are concatenated together in the same file. If multi-file output is enabled, i.e., if MF is greater than 1 and NF is asserted, then the individual plot file names will have the root name `$F`, e.g., `tmp$sgk1234a`, and a numeric extension, e.g., ".1", ".2", and so on. For example, on a UNIX host, the DD string might contain a file template such as `$F.[1-8]` to match up to eight frame files in multi-file output mode.

```
sgiver|uver|UNIX/SGI versatec interface:\
    :kf=bin$x_sgikern.e:tn=sgikern:xs#.200:ys#.200:\
    :xr#1536:yr#1536:zr#1:cw#.0125:ch#.0294:\
    :BI:YF:BF:LO#1:LS#2:PX#2112:PY#1576:\
    :XO#300:YO#40:XW#1536:YW#1536:MF#8:NF:\
    :DD=uver,tmp$sgk,!{ lpr -Pvup -s -r -v $F.[1-8]; }:
```

Figure 4. Sample SGI graphcap entry

A complete example of an actual SGI graphcap entry is given in Figure 4. Since this is a runtime file the syntax (which is patterned after the UNIX *termcap*) is concise and unforgiving, but it matters little since it is so simple. Note that the entire entry is actually a single logical line of text, using the backslash convention to continue the entry on multiple physical lines. The entry consists of a list of logical device name aliases, followed by a list of device parameters delimited by colons, with the whole delimited by the first unescaped newline encountered. Whitespace immediately following an escaped newline is ignored, elsewhere it is data. Numeric constants must be preceded by the character # to be interpreted as such.

The remaining SGI kernel parameters are those used to control the generation of raster output. Discussion of these parameters is deferred to the description of the SGI raster file format.

4. Plot File Formats

The SGI kernel can generate either metacode output, wherein the plot file contains a series of simple vector drawing commands, or raster output, wherein the plot file is a two dimensional raster (bitmap) with all the vector drawing commands already processed and the corresponding bits set in the bitmap. The two plot file formats are described in detail in the next two sections.

4.1. SGI Metacode Format

The SGI metacode file format is a sequence of 16 bit integer words containing binary opcodes and data. The metacode is extremely simple, consisting of only two drawing instructions (pen up move and pen down draw), a frame instruction, and an optional set line width instruction. All text is rendered into vectors by the SGI kernel hence there are no text drawing instructions (some other GIO kernel should be used if fancy hardware character generation, etc., is desired). The SGK metacode instruction formats are summarized in Figure 5.

opcode	*data words*	*instruction*
1	0, 0	frame instruction
2	x, y	move to (x,y)
3	x, y	draw to (x,y)
4	w, 0	set line width (>= 1, 1=normal, 2=bold)

Figure 5. SGI Metacode instruction formats

All opcodes and data words are 16 bit positive integers encoded in the machine independent MII format, i.e., most significant byte first. Only 15 bits of each 16 bit word are actually used. Coordinates are specified in the range 0 to 32767. All instructions are zero padded to 3 words to simplify metacode translation programs. Note that whether the frame instruction precedes or follows each graphics frame is a device option controlled by the FS and FE kernel parameters. Note also that the *sgidecode* task in the *plot* package may be used to examine the contents of SGI metacode files.

4.2. SGI Raster File Format

The raster file format written by the SGK consists of a binary raster file containing one or more bitmaps (rasters) with no embedded header information. All bitmaps in a raster file are of the same size. The size is specified in the graphcap entry for the device and may be passed to the host dispose task on the foreign task command line if desired. Page offsets may also be passed on the command line, e.g., to position the plot on the plotter page. The SGI kernel control parameters defined for bitmap devices are summarized in Figure 6.

BI	boolean; presence indicates a bitmapped or raster device
LO	width in device pixels of a line of size 1.0
LS	difference in device pixels between line sizes
PX	physical x size (linelen) of bitmap as stored in memory, bits
PY	physical y size of bitmap, i.e., number of lines in bitmap
XO,YO	origin of plotting window in device pixels
XW,YW	width of plotting window in device pixels
NB	number of bits to be set in each 8 bit output byte
BF	bit-flip each byte in bitmap (incredible but true)
BS	byte swap the bitmap when output (swap every two bytes)
WS	word swap the bitmap when output (swap every four bytes)

Figure 6. SGI graphcap control parameters for bitmap devices

The output raster will consist of PY lines each of length PX bits. If PX is chosen to be a

multiple of 8, the *physical* length of each line of the output raster will be PX/8 bytes. Note that the values of PX and PY are arbitrary and should be chosen to simplify the code of the translator and maximize efficiency. In particular, PX and PY do not in general define the maximum physical resolution of the device, although if NB=8 the value of PX will typically approximate the physical resolution in X. If the byte packing factor is less than 8 then PX must be increased to allow space for the unused bits in each output byte. If there are multiple bitmap frames per file, each frame will occupy an integral number of SPP char units of storage in the output file, with the values of any extra bits at the end of the bitmap being undefined (an SPP char is 16 bits on most IRAF host machines).

The plot will be rasterized in a logical window XW one-bit pixels wide and YW pixels high. The first YO lines of the output raster will be zero, with the plotting window beginning at line YO+1. The first XO bits of each output line will be zeroed, with the plotting window beginning at bit XO+1. The bytes in each output line may be bit-flipped if desired, and all of the bits in each output byte need not be used for pixel data. If the bit packing factor NB is set to 8 the plotting window will map into XW bits of storage of each output line. If fewer than 8 bits are used in each output byte more than XW physical bits of storage will be used, e.g., if NB=4, XW*2 bits of storage are required to store a line of the plotting window. The unused bits are set to zero.

While the SGI kernel may not always produce a plotter ready output raster, it will likely come pretty close. The translator can later *or* a mask into the zeroed bits, flip the data bits, complement the data bits, or perform any other bytewise operation using a simple and highly efficient table lookup vector operator (i.e., use the *value* of the raster byte as the *index* into a 256 element lookup table, passing the value of the indexed element to the plotter; prepare the lookup table once when the program first fires up).

Contrary to normal IRAF practice, storage for the raster buffer is statically allocated in the SGI kernel. This was done for efficiency reasons, since rasterization is an unusually expensive operation (the Fortran compiler can generate tighter code for a static buffer). The only disadvantage is that since the dimensions of the raster buffer are fixed at compile time, *there is a builtin upper limit on the size of a raster*. The SGI kernel comes configured ready to rasterize the output for any device with a resolution of up to 2112 by 2048 pixels. If this is too small, change the values of the *define*-ed parameters `LEN_FBUF` and `LEN_OBUF` in the file `gio$sgikern/sgk.x`, and do a `mkpkg; mkpkg install` on the package to recompile and install a new SGI kernel (obviously, you must have the source on line). Note also that efficient rasterization requires sufficient working set to avoid excessive page faulting (thrashing) on the raster buffer when drawing vertical vectors. Typical *glabax* vector plots require about 5 seconds to rasterize on our UNIX 11/750.

5. Implementing a New SGI Device Interface

As noted earlier, to implement a new SGI device interface you must [1] add an SGI graphcap entry for the device to `dev$graphcap`, and [2] write a translator program to dispose of the SGI plot file to the device.

If the device in question requires a metacode type plot file, there should be no problem **setting up the graphcap entry**, except for the DD string, which we can deal with later when the dispose task is better defined. If the device requires raster input, some investigation will be required to determine how best to format the output raster, e.g., how many bits per byte (NB), how many bits per physical output line (PX), and how many output lines (PY). The values of the various flip, swap, rotate, etc., parameters are most easily set by trial and error once the dispose task is installed and running. The *showcap* task is useful for debugging graphcap entries. The new graphcap entry can either be added directly to the installed graphcap file (`dev$graphcap`), or it can be developed in a test graphcap file and installed later, changing the value of the *graphcap* environment variable to point to the private version of the file.

Once a graphcap entry for the new device is available, it may be used in conjunction with the SGI kernel to **generate an SGI plot file** to be used to test the translation (dispose) task. To do this, get into the CL and make a GKI metacode file, e.g., using the command '`:.write mc`' while in cursor mode with any old plot displayed on the graphics terminal. Next, run the SGI kernel (task *sgikern* in the *plot* package) to turn the GKI metacode file into an SGI metacode or raster file. Note that the plot file is normally deleted automatically, so to generate the test plot file you must set the graphcap entry up with a null dispose command of some sort, also making sure that the *RM* switch is omitted from the graphcap entry.

Given the test plot file, we are ready to **code and test the translation program**, normally implemented as a host Fortran or C program (note that it may not be necessary to write such a program at all - the host system may already provide something which can be used directly, such as the UNIX *lpr* task in the sample graphcap entry for the versatec, shown in Figure 4). The IRAF system comes with a number of such programs ready made for the more common plotter devices; you will find these in the directory `host$gdev/sgidev`. Each device interface is contained in a single file. Pick the one for the device most closely resembling the device you are interfacing, make a copy of it with a new name, and modify the code as necessary for the new device. Test the program on the test plot file, go back and change the graphcap if necessary and generate a new test plot file, and iterate until the process converges.

The final products of this exercise are the graphcap entry and the dispose task. It is probably simplest if the graphcap entry is installed directly in `dev$graphcap`. The dispose task may either be installed directly in the system, like the SGI device interfaces that come with IRAF, or it may be installed in the `LOCAL` directory or some place outside of IRAF, changing the graphcap entry to point to the task wherever it has been installed. If you choose to install the dispose task and its source directly in the system, remember to save it and merge it back in when future versions of IRAF are installed.

The procedure for adding a new dispose task to the standard system is as follows:

[1] Install the source file in `host$gdev/sgidev`.

[2] Make an entry for the device in all *mkpkg* files in the same directory, e.g., `mkpkg`, `mkpkg.com`, `mkpkg.csh`, and so on, depending upon the host system.

[3] Type `mkpkg update` to compile, link, and install the new executable in the `HLIB` directory (where host dependent runtime files go).

[4] On a VMS host, add an entry for the device to `hlib$sgiqueue.com`, and set up the graphcap entry to submit this file to run in a batch queue, like the other VMS/SGI devices. On a UNIX host, *lpr* provides queueing as a builtin feature, and can easily be used to set up new queues. Alternatively, appending an '`&`' to the DD dispose command will at least allow the translation to be carried out in the background, and prefixing the command with *nice* will cause it to be run at a reduced priority.

Note that while IRAF does not (currently) provide a builtin network capability for SGI devices, the UNIX *lpr* provides full network access on UNIX hosts, and on a VMS host something can usually be cobbled together using DCL command files and DECNET. The principal advantage of using the IRAF network capabilities with SGI will be the ability to easily access devices on a remote node running a different operating system than that on the local node.

6. Site Dependencies in the SGI Graphcap Entry

As the number of plotter devices supported by SGI translators in the standard IRAF system grows, many sites will find that support for their local plotter device is already provided by the standard IRAF system. In most cases the only changes required to interface to the local device will be modifications to the graphcap entry for the device, and possibly modifications to

any associated script files in `HLIB`. The parameters the user is most likely to need to change in the graphcap entry are the logical device name and the dispose command (DD parameter), e.g., to change the name of the plotter queue, if passed as a command line argument in the dispose command. If multiple copies of the same device exist on the local network, a separate graphcap entry must be provided for each.

7. Some Examples

We have already seen one example, i.e., an SGI interface for a versatec raster plotter on a UNIX node (Figure 4). This example is unusual because no special SGI translation task is required; the raster file format required by the UNIX *lpr* is sufficiently device and system independent that the SGI kernel can produce it directly. As a bonus, *lpr* provides full network access and queued execution, without any extra work on our part.

Life is never quite so simple on VMS, but the VMS SGI interface for the versatec plotter is not very difficult either. To dispose of a raster to the versatec on a VMS host, one must reformat the raster file produced by the SGI kernel to produce an RMS fixed format record structured file, blocked 132 bytes per record (130 data bytes plus a 2 byte record type code, used to signal formfeeds and such).

The primary function of the VMS/SGI dispose task for the versatec, therefore, is to copy the SGI raster plot file, reformatting it into the peculiar record structure required by the VMS print queue. Once the record structured file has been generated, the file is disposed of to the print queue and the SGI plot file is deleted. When plotting is complete, the VMS print queue deletes the queued file. The graphcap entry for all this is shown in Figure 7. Note the use of the VMS batch queue facility in the DD string; this makes *gflush* appear considerably faster that it would otherwise be. The *FAST* queue is used since the reformatting operation is expected to take only a few tens of cpu seconds.

```
sgiver|Basic SGI/SGK interface to the versatec:\
    :kf=bin$x_sgikern.e:tn=sgikern:xs#.200:ys#.200:\
    :xr#1536:yr#1536:zr#1:cw#.0125:ch#.0294:\
    :BI:YF:BF:LO#1:LS#2:PX#2112:PY#1576:XO#300:YO#40:XW#1536:YW#1536:
vver|SGI/SGK interface to the versatec on the local VMS node:\
    :DD=vver,tmp$sgk,sub/que=fast/noprint/nolog \
    /para=\050"vver","$F","2112","1576","versatec","$F.ras"\051 \
    irafhlib\072sgiqueue.com:\
    :MF#8:tc=sgiver:
```

Figure 7. VMS/SGI graphcap entry for the versatec raster plotter

We have concentrated on raster plotters in our examples because the raster plotter interface is more complex than the metacode file interface, and because at the time this was written, there was not yet a working example of a laser printer SGI interface (SGI interfaces for the *imagen* and *QMS* were under development). The SGI interface for a metacode device is very similar, the principal difference being the omission of the `:BI...` etc. line in the graphcap entry. The dispose task for a laser printer simple reads successive 3 word SGI instructions from the SGI metacode file until EOF is reached, translating each instruction into the plotting instructions required by the laser plotter, with rasterization being performed by the plotter itself.