A GIPS Tutorial

Peter G. Ford

Center for Space Research
Massachusetts Institute of Technology
Cambridge, MA 02139

General Images
Image Processing

ABSTRACT

The *General Image Processing System* (GIPS) is a set of software tools that assists in creating, manipulating and displaying two-dimensional images within the UNIX† environment. All GIPS images are stored as single UNIX files, and may be piped from one process to another, or between nodes of a network. GIPS commands obey the familiar rules for UNIX command syntax, and image operations of arbitrary complexity may be built up from modular building blocks. Where necessary, the commands address specific output devices (e.g. frame buffers, film recorders), via a common interface.

This document does not describe all GIPS capabilities—for which you should consult the companion paper "*Guide to GIPS*" and the relevant manual entries. Instead, it prompts you to investigate some of the powerful GIPS techniques on simplified image files.

GIPS is currently implemented on Sun workstations and on VAX‡ computers running Berkeley UNIX and Ultrix‡ operating systems. Output devices include those supported by the SunView§ and XView applications interfaces, the AED 767 graphics station, and the Comtal Vision ONE/20 image workstation.

2/14/92

† UNIX is a trademark of Bell Laboratories.
‡ VAX and Ultrix are trademarks of Digital Equipment Corporation.
§ SunView is a trademark of Sun Microsystems Inc.

A GIPS Tutorial

Peter G. Ford

Center for Space Research
Massachusetts Institute of Technology
Cambridge, MA 02139

General Images
Image Processing

1. Introduction—Fun with *gin* and *git*

The fundamental data structure of the GIPS system is the *General Image.* For the moment you can think of it as a 2–dimensional data array with auxiliary bookkeeping information. To familiarize yourself with General Images, onto a UNIX system and run the following pair of commands:[1]

$ gin | git

The following output should appear at your terminal...

```
 0  1  2  3  4  5  6  7
 8  9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
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
```

The *gin* command writes a very simple General Image to the standard output stream, from which it is read by the companion *git* command. *Git* translates the GIPS pixels to ASCII numbers and writes them to the standard output. You should **never** write a GIPS image directly to your terminal—the binary pixels are guaranteed to make garbage of your screen! In due course, we'll discuss how to send a real image to an imaging display device; in the meantime, the examples will use *git* to display the image pixel values.

By default, *gin* creates an 8×8 image, filling each pixel with the integers in ascending order. You can alter the dimensions by including options in the *gin* command line, e.g. to create a 6×12 image...

$ gin –x 12 –y 6 | git

```
 0  1  2  3  4  5  6  7  8  9 10 11
12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 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 64 65 66 67 68 69 70 71
```

Notice that *git* did not have to be told explicitly the dimension of the image. General Images are self-defining via a *header* that is prefixed to the binary pixels. You can ask *git* to display this header by invoking it with the

[1] Throughout this document, GIPS command lines are printed in **boldface**, and prefixed with a "$" sign to denote the shell prompt.

–h option, *viz:*

```
$ gin –x3 –y4 | git –h
header = "Image Header"
version = vax
dtype = b
dscale = s
xnum = 3
ynum = 4
history = "Tue Jun 25 01:06:53 1985   gin: –x3 –y4"
FINIS=
  0   1   2
  3   4   5
  6   7   8
  9  10  11
```

Within a GIPS header, *xnum* and *ynum* specify the dimension of the image, *dtype* determines the type of pixel, and so forth. GIPS aims to take most of the housekeeping chores away from the user. For example, the *ihrot* command rotates any[2] GIPS image through 90°. Here is what it does to a *gin* image and its header:

```
$ gin –x 3 –y 4 | ihrot | git –h
header = "Image Header"
version = vax
dtype = b
dscale = s
xnum = 4
ynum = 3
history = "Tue Jun 25 01:15:49 1985   gin: –x 3 –y 4"
history = "Tue Jun 25 01:15:49 1985   ihrot:"
FINIS=
  2   5   8  11
  1   4   7  10
  0   3   6   9
```

Notice that the values of *xnum* and *ynum* have been swapped by passing the image through *ihrot* and that another *history* entry has been added to the header.

To summarize, the GIPS system consists of a number of UNIX commands and C–language subroutines that read and write files called *General Images*. Each such image begins with an ASCII *header*, followed by a binary data array. The header contains the complete description of the image, and GIPS commands automatically change it to keep track of any changes they have made. Finally, notice that General Images are never[3] updated "in-place", they are only transformed by being copied from file to file, or from process to process.

---

[2] A command such as *ihrot* reads the header of the input image, checks out how much memory and temporary disk space is available, and only then chooses an appropriate algorithm for performing its function, which in this case consists of rotating the image counter-clockwise. If you feel that a particular command is behaving inefficiently, you can often tune the algorithm by supplying command-line arguments. In this case, if you turn to the *ihrot*, manual pages, you will see that you can set reset the maximum amount of memory used, and specify a non-default spool directory. Many GIPS commands behave in this way—the larger the image, the more help is needed from the user to ensure that the commands take a finite time and use finite system resources. Now back to the tutorial!

[3] Never is a dangerous word to use in the software business—years after this paragraph was written, along came *ihpho*, a command that edits GIPS images in-place.

2. Filters

We learned in the previous section that *gin* writes an image to its standard output and *git* reads an image from its standard input. Most other GIPS commands do both, i.e. read an image from *stdin*, transform it in some way, and write it to *stdout*. In UNIX terms, such a command is known as a *filter*, by analogy with the function performed by a filter in a camera or electronic circuit.

2.1. The Pixel Arithmetic Filter—*iha*

The simplest GIPS filter to describe is *iha*, a program that transforms one or more input images, pixel by pixel, according to a "script" written in C language syntax,[4] in which the output pixel is represented by the variable *p*, and the corresponding input pixels from one or more images by *p1, p2, . . .* . For instance, to double the value of each pixel of an image, use the following:

```
$ gin | iha "p=2*p1" | git
  0   2   4   6   8  10  12  14
 16  18  20  22  24  26  28  30
 32  34  36  38  40  42  44  46
 48  50  52  54  56  58  60  62
 64  66  68  70  72  74  76  78
 80  82  84  86  88  90  92  94
 96  98 100 102 104 106 108 110
112 114 116 118 120 122 124 126
```

2.2. The Convolution Filter—*ihbox*

We call *iha* a *local* filter—each output pixel value depends only on the corresponding input pixel, not on the values of surrounding pixels. The simplest *non–local* filter, *ihbox*, replaces each input pixel by an average of itself and its surrounding pixels, each multiplied by an element of a so-called *convolution* array. The array may be stored in a UNIX file whose name is given to *ihbox*, or alternatively the array itself may be written on the command line. Here is a "before and after" example:

```
$ gin –x4 –y4 | git
  0   1   2   3
  4   5   6   7
  8   9  10  11
 12  13  14  15
```

```
$ gin –x4 –y4 | ihbox "3 3 1 0 2 0 4 0 2 0 1" | git
  0   1   2   2
  2   5   6   5
  5   9  10   7
  7   9  10   8
```

The *ihbox* argument string specifies the convolution array—in this case, the 3×3 array

```
1   0   2
0   4   0
2   0   1
```

Mathematically, if we write this array as A_{nm} and the input image as I_{ij}, the result of *ihbox* will be

$$I_{ij} \rightarrow \left(\sum_{n=-N/2}^{N/2} \sum_{m=-M/2}^{M/2} A_{nm}\ I_{i+n,\,j+m} \right) \Big/ \left(\sum_{n=-N/2}^{N/2} \sum_{m=-M/2}^{M/2} |A_{nm}| \right)$$

If this seems a rather complicated business, well—it is! But many powerful image operations may be represented by very simple A_{nm} arrays, for instance. . .

[4] *iha* is similar in principle to the UNIX command *awk*, but, instead of interpreting the command script pixel–by–pixel, it invokes the C compiler internally to generate in-line code. There is therefore a delay of a few seconds before an *iha* process initializes, but you make it back on any but the smallest images by the time the whole image has been filtered.

Low–pass Filter	*High-pass Filter*	*Directional Derivative*
0 1 0	1 0 1	1 0 0
1 4 1	0 0 0	0 0 0
0 1 0	1 0 1	0 0 –1

Leaving aside the behavior of *ihbox* at the boundary of an image, the transformation does not depend explicitly on either the *x* or *y* pixel coordinate. Such a transformation is therefore *separable*, in contrast with some others that we shall be considering later.

2.3. The Fast Fourier Transform Filter— *ihfft*

Another separable GIPS transformation is performed by the *ihfft* command, which takes a row-by-row Fourier transform of an image...

```
$ gin –x4 –y4 | ihfft | git –h
header = "Image Header"
version = vax
dtype = c
dscale = s
xnum = 4
xlfft = 4
ynum = 4
history = "Tue Jun 25 02:11:07 1985   gin: –x4 –y4"
history = "Tue Jun 25 02:11:07 1985   ihfft:"
FINIS=
(3  0)  (1.414 –1)  (1 0)  (1.414  1)
(11 0)  (1.414 –1)  (1 0)  (1.414  1)
(19 0)  (1.414 –1)  (1 0)  (1.414  1)
(27 0)  (1.414 –1)  (1 0)  (1.414  1)
```

In this example, *git* has displayed complex pixel values in the format "*(real,imaginary)*". You may have noted some changes in the header: *dtype* now has the value **"c",** indicating that the pixels have been transformed to complex floating-point format, and a new variable, *xlfft*, is created to preserve the pre-transformation x-dimension. The new *xnum* will always be a power of 2.

The **–i** option of *ihfft* takes the inverse Fourier Transform, so recalling that the inverse Fourier transform of a Fourier transform is just the original function, you shouldn't be very surprised at the following:

```
$ gin –x4 –y4 | ihfft | ihfft –i | git –a
 ~0   1   2   3
  4   5   6   7
  8   9  10  11
 12  13  14  15
```

where we have used the **–a** option of *git* to display the *moduli* of the complex pixels. By the way, that tilde sign, **"~"** is *git*'s way of showing that a floating-point pixel has a nearly zero value. In this case, the error is caused by arithmetic round-off inherent in taking a Fourier Transform.

2.4. Two-Dimensional Fourier Transforms

We can now put together the pieces required to perform a 2–dimensional Fourier Transform:

```
$ ... | ihfft | ihrot | ihfft | ...
```

and, with the additional knowledge that the **–ir** flags applied to *ihrot* effects an inverse rotation, we can verify the following identity transformation...

```
$ gin –x4 –y4 | ihfft | ihrot | ihfft | ihfft –i | ihrot –ir | ihfft –i | git –a
~0   1   2   3
 4   5   6   7
 8   9  10  11
12  13  14  15
```

In a "real life" situation, you would insert a filter, (e.g. *iha*) between the second and third *ihfft* commands to change the Fourier–space image components...

```
$ ... | ihfft | ihrot | ihfft | iha "

    float fact = fabs((1.0–(2.0*x)/xnum)*(1.0–(2.0*y)/ynum));
    r = r1*fact;
    i = i1*fact;

" | ihfft –i | ihrot –ri | ihfft –i | ...
```

The syntax of the *iha* script needs some explanation—temporary variables, in this case *fact*, can be declared and initialized using C syntax. The name of any header variable, e.g. *xnum* and *ynum*, will be replaced by its value in the output header. Mathematical routines such as *fabs* located in the UNIX "*libm.a*" library may be used without any special declaration. Finally, pixels within complex images (*dtype=c*) are not referenced by *p*, *p1*, *p2*, etc., but by *r*, *r1*, *r2*... for their *real* parts and *i*, *i1*, *i2*... for their imaginary parts. Therefore, in this example, the filter selectively down-weights pixels that have *x* and *y* addresses far from the borders of the Fourier-space image, i.e. those corresponding to high spatial frequencies.

3. Creating a GIPS Image

The *gin* command cannot create a meaningful image—its function is to help you learn GIPS, not to do serious work. There are four ways of creating an image from real data...

3.1. From a Binary Array

Create a binary data file, i.e. an array of $n \times m$ pixels, each of the same data type. Then create a GIPS header to describe that array. As headers contain only ASCII characters, you may use an editor for this task. Finally, invoke the *ihc* command to combine the header and the array. If necessary, *ihc* can be told to *invert* (top–to–bottom, the **–i** flag) or *reverse* (left–to–right, the **–r** flag) the array while creating the image. Each element of the array must be a numeric data type supported by GIPS (see Table 2). The binary array may be read from disk, from tape, or from the standard input stream, i.e. piped from the standard output of some other program. The array should consist of nothing but a stream of binary numbers—embedded newlines or other formatting fields are expressly forbidden.

In the following example, the header is read from the file "*gi.hdr*", and the array is read from the tape device named "*/dev/rmt1*":

```
$ ihc –h gi.hdr –4096 /dev/rmt1 | ...
```

where the **–4096** flag specifies that the physical tape block size is (at most) 4096 bytes. A sample header file is shown in Table 1. Header formats are discussed in detail in the *ihread*(3) manual entry and in the "*Guide to GIPS*" document. The variables that must be present in all General Image headers are shown in **boldface** in Table 1.

Table 1: A sample General Image header

header	**= "Image Header"**
version	**= ieee**
title	= "Pioneer Venus Radar Image"
owner	= "Peter Ford"
origin	= /data/ford/pv/cytglob.db
dtitle	= "Planetary Radius (km)"
dtype	**= b**
dscale	= n
dmax	= 6064
dmin	= 6048
dnull	= 0
dfact	= 0.0625
dzero	= 6048
xnum	**= 1024**
xscale	= w
xmax	= 240
xmin	= –120
xorg	= 0
xtitle	= "Longitude (\(deE)"
xfont	= times.r.10
xaxis	= "place=tbco font=R.6 by=60."
ynum	**= 512**
yscale	= m
ymax	= –75
ymin	= 75
yorg	= 0
ytitle	= "Latitude (\(de Mercator)"
yfont	= times.r.10r
yaxis	= "place=tbco font=R.6 from=60. by=–30."
FINIS=	

3.2. From Randomly-Supplied Pixels

Create an ASCII header as in the previous section, but invoke *ihc* with the **–w** flag and supply the pixels in any order. Specifically, *ihc* expects to ready triplets of values[5]

- the *x*–coordinate
- the *y*–coordinate
- the corresponding real-world data value

Ihc reads these fields until an end-of-file is reached, then sorts them and generates a General Image which may contain "gaps" in which no pixel values were supplied. The header variable *dnull* tells *ihc* what value to assign to such pixels. An image formed in this way may need to be subsequently filtered through "*ihbox –n*" in order to "fill in" these gaps from the values of neighboring pixels.

In the following example, the C–source for *a.out* writes to the standard output as follows:

[5] If the **–f** flag is used, *ihc* expects to read triplets of ASCII fields. Otherwise, the input stream must consist of an array of 12–byte structures, each containing a pair of *long* integer (INTEGER*4) coordinates followed by a *float* (REAL*4) pixel value.

```
struct {
        long      x, y;   /* pixel coordinates */
        float     f;      /* pixel value */
} data;

while ( ... ) {
        data.x = ...          /* x-coordinate */
        data.y = ...          /* y-coordinate */
        data.f = ...          /* pixel value */
        fwrite((char *)&data, sizeof(data), 1, stdout);
}
```

and the shell command syntax is

$ a.out | ihc –w –h gi.hdr –m "xnum=512 ynum=480" | ...

The **–w** flag tells *ihc* that the pixels are being supplied in random order. *Ihc* reads the GIPS header from the "*gi.hdr*" file, as in the previous section, but we have added a further twist—the **–m** flag, followed by its argument, "*xnum=512 ynum=480*"— which cause *ihc* to modify the values of a pair of header variables, *xnum* and *ynum*. Alternatively, these reassignments could have been placed in a file, e.g. "*gi.hdr.updt*", and the name of that file supplied to *ihc* via the **–m** option, e.g.

$ a.out | ihc –w –h gi.hdr –m gi.hdr.updt | ...

In general, the **–h** and **–m** flags of all GIPS commands may be followed either by a character string or by a file name. In all cases, the presence of an equals sign in the field signifies the former, and its absence the latter.

3.3. From a Foreign Image

GIPS comes supplied with several "input" filters that can automatically translate images into GIPS format, preserving as much of the foreign header information as can be recognized. The simplest of these programs, *ihfrompix* and *ihfromgif*, which convert Sun rasterfile and Compuserve GIF images, respectively, are straightforward because their input image headers contain little precise information about their contents beyond the image size and color map. In the following example, the contents of a Sun frame buffer is "captured" and saved as a General Image file, "*gi.sun*",

$ screendump | ihfrompix –a sun.cmap –o gi.sun

and the current Sun color map is copied to the file "*sun.cmap*". The other input filters, *fits*, *ihpds*, *ihstokes*, and *ihvicar* are considerably more complicated since they translate from images can contain much information. You should read the GIPS manual entries for a detailed description of these commands, since they have several limitations, which may change over time as the specifications of these foreign formats are developed.

3.4. From a User Program

Write your own program in C or Fortran, using the GIPS library routines to generate the header and binary array...

ihread	reads an existing header file or string into core.
ihmerg	updates an in-core header from a file or string.
ihget	accesses a header variable.
ihput	updates a header variable.
ihwrit	writes out the updated header.
ihdel	delete a header variable
ihtype	get format and description of header variable
ihbin	reads a binary image raster.
ihbout	writes a binary image raster.
ihbcl	closes an opened image file.

These routines are located in "*/usr/local/lib/libGI.a*", and may be loaded with your programs by including the **–lGI** flag in your *f77* (1), *cc* (1), or *ld* (1) commands. Some examples of Fortran and C programs are given in the "*Guide to GIPS*" document. If you decide to write one yourself, you should also consult the manual entries for *gips* (3) and *ihread* (3). As a general rule, you should not need to construct your own image header from scratch each time. Instead, call *ihread* to read a skeleton header file, or the header of an existing image, then call *ihmerg* or *ihput* to fix-up specific variables.

3.5. More About GIPS Headers

Each GIPS header must include three **required** fields: *header*, with its value "*Image Header*", identifies this as a GIPS image header; *version* identifies the binary pixel format, and prevents an image created by one CPU from being processed by another if their binary data formats differ; FINIS= marks the end of the header. In addition, the header must contain *xnum* to define the number of columns and *ynum* to define the number of rows that the image possesses.

Table 2: Allowed *dtype* Values

dtype value	*Data Type*	*Length in bytes*	*Minimum value*	*Maximum value*
b	Unsigned byte	1	0	255
s	Signed byte	1	–128	127
h	Unsigned half-word	2	0	65535 (2^{16}–1)
u	Signed half-word	2	–32768 (-2^{15})	32767 (2^{15}–1)
l	Unsigned full-word	4	0	2^{32}–1
i	Signed full-word	4	-2^{31}	2^{31}–1
f	Single-precision float	4		
d	Double-precision float	8		
c	Complex (FFT)	2×4		

The *dtype* variable describes the type of pixel element. The allowable *dtype* values vary with the possible data types supported by your particular computer. The definitions for 32–bit machines such as the VAX, MIPS, SPARC, and MC680x0 processors are shown in Table 2.

Another header field, *dscale*, indicates the relationship between the *real* datum value, e.g. units of *kilometers*, *megahertz*, *democrats*, etc., and the value stored in the image pixel. The most usual values are *dscale=s,* indicating that the two are the same, and *dscale=n,* indicating that the real value is related to the pixel value via the auxiliary header variables *dfact* and *dzero* in the following manner...

$$\text{Real} = dfact \times \text{pixel} + dzero$$

There are a number of other possible *dscale* values described in "*Guide to GIPS*".

4. Displaying an Image—*gipstool* and *xgips*

This section describes the commands that display images via the SunView and X11/XView interfaces. Commands that use the simpler "device independent" interface—the *comsubs* subroutines in the library "*libFB.a*"—are described in the Appendix.

The *gipstool* command can be invoked on a Sun workstation that possesses a local frame buffer accessed by a SunView or OpenWindows display server. You can not run *gipstool* from a remote login session or shell. By contrast, *xgips* can run on any computer that can make a connection to an X window server, and you get to specify which server either by means of the $DISPLAY environment variable or by the **–display** command option. The penalty for this greater flexibility is that *xgips* is somewhat slower than *gipstool*, and the current version does not support interactive color or greyscale enhancement. However, these programs have very few other differences and, in what follows, we'll give examples of *xgips* usage with the understanding that, unless explicitly stated otherwise, everything will apply equally well to *gipstool*.

4.1. *Xgips* Command Syntax

The full syntax of *gipstool* and *xgips* commands is quite complicated, since many modes are supported...

xgips [**–KNSc**] [*–depth*] **[–C** *map***] [–Iaimo** *file***] [–XY** *size,left,right***] [–e** *coeffs***] [–n** *colors***]**
[–r *buffers***] [–xy** *size,org,off***] [–zZ** *zoom***] [***file***]**

These arguments are explained in the *gipstool* (1) manual pages. Here we merely point out that the first set of flags, **–KNSc**, take no arguments, and may be concatenated, i.e. **"xgips –K –N –S"** is identical to **"xgips –KNS".** Each other flag must be followed by an argument, or in the case of the **–X**, **–Y**, **–x**, and **–y** options, the argument may consist of up to three integer sub-fields, separated by commas.

4.2. Creating an Image Window

If *xgips* is given the name of a General Image, it reads its header, figures out how large it is, and creates a window in which to display it. If the image name is omitted, it reads it from the standard input stream, e.g. at the end of a pipe:

$ ihc –h gi.hdr /dev/rmt1 | ihbox bx1 | xgips –a topo

Xgips then translates the input pixels into values suitable for the display device, and writes them into the window, line by line. The pixels are first converted to "real world" values, as defined by the *dscale* header variable, and then linearly mapped to display values using the *dmin* and *dmax* header values—real-world pixels less than *dmin* are given the lowest display value, those greater than *dmax* are given the highest display value, and the remainder are linearly mapped between the two.

4.3. Halting *xgips* while the Window is being Written

Xgips cannot receive any input from the mouse or keyboard while it is initializing the window, although you can kill the entire process either by selecting "*Quit*" from the window's border menu, or by sending an interrupt signal (usually CTRL–C) to the window in which *xgips* was invoked. (Note: the *gipstool* border menu cannot be accessed at this time, but the second method of halting the program—CTRL-C in the shell window—should do the trick).

4.4. Magnifying the Image

The image can be "zoomed" by *x*– and *y*–factors supplied by the **–z** and **–Z** options, respectively. Zooming is performed by pixel replication, and is restricted to integer magnification factors. If the image is larger than the screen, only the top left portion will be displayed, but *xgips* will have read the entire image into memory so all parts can be inspected using subcommands. Less than the whole image can be read, and the size and position of the image in the display window can be adjusted by invoking *xgips* with suitable values of the **–X**, **–Y**, **–x**, and **–y** options.

4.5. Displaying Cellular Images

The situation is somewhat different if the input image is in "cellular" format, i.e. has been created by the *ihbidr*, *ihgfmt*, or *ihpho* commands. In these cases, *xgips* should be invoked with values of **–x** and **–y** specifying a window size that is not greater that the size of the display screen. Subsequent *xgips* motion subcommands will cause different parts of the image to be read and displayed, usually very rapidly in comparison with typical access times for non-cellular images. This action can also be specified for non-cellular images by invoking *xgips* with the **–c** option. Access efficiency to cellular images can be further improved by specifying a buffer count via the **–r** option—selecting the number of "ring buffers" used to store the input image cells, usually as many as possible provided sufficient memory is available.

4.6. Color Selection

By default, *xgips* displays an image in 64 shades of grey. Once allocated, the shades cannot be changed, nor can the assignment of a particular shade to a given range of pixel values. The number of shades can be altered by the **–n** option, which must be a power of 2, in the range from 2 to 256. By default, the mapping between pixel display value and shade is linear, but this can be altered by specifying the name of a "colormap" file in the **–a** option. A colormap file contains precisely 256 lines (not counting comments and null lines), each containing

three numbers in the range from 0 to 255—the red, green, and blue intensities in which to display up to 256 colors. If *n* shades are requested, their intensities are taken from every 256/*n*'th line of the colormap file.

GIPS comes with several colormap files in the "/*usr*/*gips*/*lib*/*cmap*" directory, and *xgips* looks here if the colormap file name cannot be resolved, i.e. in the above example, it first looks for "*topo*", then for "*topo.cmap*", then for "/*usr*/*gips*/*lib*/*cmap*/*topo*", and finally for "/*usr*/*gips*/*lib*/*cmap*/*topo.cmap*".

Once defined, colormaps can be "shared" between different applications accessing the same display, thereby helping to reduce the chance that the default color table will overflow, causing the window manager to "flash" colors depending on which window the mouse pointer is located in. Colormaps are shared by being given the same name. By default, this name is the same as the colormap file name (or "bw" if none is specified), but it can be set independently by the **–C** option.

4.7. *Gipstool* Colors

So far, everything said about *xgips* also applies to *gipstool*, except that the latter creates 128 shades by default, not 64, and that some dual color/BW frame buffers will not display color unless the **–8** flag is set. The major difference between the commands lies in the ease of changing color assignments once the image, or a portion of it, has been displayed. Because *gipstool* can directly change the local color lookup tables, it is much easier to change the pixel color values, and, once allocated, colors stay defined. When all 256 colors are required, *gipstool* should be invoked with the **–N** option, and, although any other windows on the display device may either vanish or change color in an unpredictable way, the General Image itself will be displayed in the chosen colors.

The same is not true of *xgips*, which must pass all color requests through a "window manager" program, which usually thinks it knows which colors you want to display better than you do! The result is that, if you want to change image colors from within *xgips*, you must invoke it with the **–N** option, which invariably leads to color "flashing" while the window manager reloads a "private" set of colors whenever the mouse pointer is moved into the image window.

4.8. Mouse Commands

Xgips assumes that you are using a 3–button mouse. Holding down the RIGHT button displays a multi-level menu. The menu items are the basic *xgips* subcommands, each of which can also be invoked by a single keystroke. In the menus, the keystroke codes are followed by brief descriptions of the functions they invoke.

Clicking the MIDDLE mouse button while pointing to an image pixel creates two "pop-up" windows, one showing a magnified view of the image in the area surrounding the chosen pixel, the other tabulating the address of the pixel and its real-world and display values. Since each pixel spans a certain finite range of real-world *x*–, *y*–, and data–values, the upper and lower limits of each value is displayed opposite the corresponding *xtitle*, *ytitle*, and *dtitle* header value.

The user can assign the LEFT mouse button to any other *xgips* subcommand—when the button is clicked, the subcommand is executed. Some subcommands, e.g. **p** and **^V**, will respond in a special way when assigned to the LEFT button, as detailed below.

While the mouse buttons are usually clicked within the image window, there are three special sub-windows in which the mouse will act differently—the color palette (**C** key), the information window (TAB key), and the tags window (**"{"** key).

4.9. Control Commands

The following subcommands perform useful functions while not actually changing the displayed image.

^C Quit the *xgips* command, after prompting the user to confirm this intention.

^H (Back-Space) Get help on any *xgips* subcommand—type any subcommand keystroke and read a brief explanatory message.

^I Display an information pop-up window to the right of the image window. The pop-up shows the settings of several *xgips* variables and modes, which can be changed by clicking in a series of command buttons with the LEFT mouse button.

^L Redisplay the image, repairing any damage.

^M (Return) Remove all pop-up windows and other inserts from the screen.

^Q Toggle the cross-hairs on and off (in *gipstool* only).

/ Display or set a keyword value in the header of the General Image being displayed.

! Hide the cursor, execute a UNIX shell command, beep the screen, and de-display the cursor. This is particularly useful when using *screendump*(1) to make a copy of the current display.

1 Assign any *xgips* subcommand to the LEFT mouse button.

4.10. Changing Colors

The colors assigned to image pixels and graphics can be changed at any time by *gipstool* subcommands, but this is only true for *xgips* if it was invoked with the **–N** option.

^W Display a color wedge—once displayed, the wedge can be re-sized and, by holding down the RIGHT mouse button within the wedge, a menu will appear which can cause the shades to run from to-to-bottom, right-to-left, etc.

B Select the background R,G,B values, i.e. a triplet of integers, each in the range 0–255, which will replace the lowest value in the colormap.

H Write a histogram of the area defined by the image mark (see "Graphic Drawing Subcommands", below) and the current cursor location to a disk file. The histogram will be in the form of a 7×256 floating-point General Image containing a count of real-world and display pixel values and their R-G-B colormap values for each of the possible 256 display values. Alternatively, if the disk file name is omitted, the image is written to a *gipsmongo* socket (see the **^G** subcommand, below).

K Write the current colormap to a disk file.

M Read a new colormap from disk.

S Apply an adaptive pixel stretch—the algorithm is based on a pixel-frequency histogram that is re-calculated whenever a fresh image or part of an image is read. The stretch is specified as a percentage. (See also the **s** subcommand, below).

V Select the foreground R,G,B values, i.e. a triplet of integers, each in the range 0–255, which will replace the highest value in the colormap.

s Apply a linear pixel stretch—unlike the **S** command, this linearly re-stretches the pixel values between two real-world limits supplied by the user.

4.11. Image Motion Subcommands

If you want to "roam" through a cellular image, or if *xgips* has read an image that is larger that your display window, and you want to display more of it, you should use the following subcommands...

P Set the pan increment—the number of pixels by which the image window will move as a result of the "*move left*", "*move right*", etc. subcommands. If set to zero, the increments will default to half the window width or height.

^A Move to the left border.

^B Move left (on a Sun keyboard, the left arrow key will also perform this function).

^E Move to the right border.

^F Move right (on a Sun keyboard, the right arrow key will also perform this function).

^N Move down (on a Sun keyboard, the down arrow key will also perform this function).

^P Move up on a Sun keyboard, the up arrow key will also perform this function).

^T Move to the top border.

^U Move to the bottom border.

^X Set the top left corner of the image to the current cursor location—unless this is the top left window pixel, it will introduce left and/or top borders, which you might want to use for axis scale marks, etc.

^Y Set the bottom right corner of the image to the current cursor location—unless this is the bottom right window pixel, it will introduce right and/or bottom borders, which you might want to use for axis scale marks, etc.

4.12. Zoom Window Subcommands

The following commands affect the pair of pop-up windows that show a zoomed area surrounding the cursor location, and the real-world and displayed values of that location. These have already been mentioned above when discussing the "Mouse Commands".

^V Display the pop-ups—this is the identical function to that of the MIDDLE mouse button.

Z Set the dimensions of the zoom pop-up; the user must supply three values—the number of pixels to be displayed in the pop-up, their magnification factor, and the color index of the central cross-hairs.

4.13. Pixel Location Subcommands

Individual image pixels can be located either by their pixel addresses, their real-world addresses (e.g. latitude and longitude), or by their presence in a *tag* list that is maintained on an external disk file and can be created, read, and updated by *xgips* subcommands.

. Locate a given pixel by its address or real-world coordinates. If the pixel isn't currently on the screen, but can be displayed either by panning through an in-core image or by reading more of an external image file, the program attempts to center the pixel in the re-written image window. Once located, the cursor is pointed at the pixel.

^R Display the tag window.

R Recall a named tag—the user enters the tag name, and *xgips* locates the pixel with those real-world coordinates.

T Create a new location tag—inserting the real-world coordinates corresponding to the cursor location, and prompting the user to specify a tag name.

{ Read a tag file from disk.

} Write the current tag list to disk.

4.14. Graphic Plane Subcommands

All graphic items are assigned to a particular "*graphic plane*", numbered from 0 to 255. Plane 0 is special in that it doesn't move with the image as it is panned. The advantage of planes is that they can be turned on and off at will.

^K Clear the current graphic plane, deleting all items defined in it.

D Remove a particular graphic plane from the screen—it can be re-displayed with a subsequent **G** subcommand.

G Set the index of the current graphic plane—if the plane already contains any items, they will be displayed. When *xgips* starts up, only plane 0 is defined.

u Undo the most recent graphic action, i.e. remove the effect of the last graphic subcommand.

4.15. Graphic Drawing Subcommands

The following subcommands cause one or more graphic objects to be added to the current graphic plane, according to the values of the graphic parameters described in the next section. Several subcommands refer to the "*image mark*", which is a pixel location set by the **^D**, **^O**, **^S**, and **R** subcommands, and used by subsequent subcommands.

Refer to the "*Guide to GIPS*" document for details on how to use the axis-drawing subcommands **a**, **m**, **x**, and **y**.

^D Draw a line from the image mark to the cursor location; then re-set the image mark to this location.

^O Set the image mark to the current cursor location. If the *LEFT* mouse button is then held down while the cursor is "dragged", a temporary "rubber band" rectangle is stretched from the image mark to the cursor, outlining the area that will be selected by a subsequent **H** or **I** subcommand.

^S Set the image mark to the current cursor location. If the *LEFT* mouse button is then held down while the cursor is "dragged", a temporary "rubber band" line is stretched from the image mark to the cursor. This line can be made permanent by hitting the **^D** key.

a Draw separable *x* – and *y* –axes, with scales and titles, according to the current values of header variables *xaxis* , *xtitle* , *xfont* , and *yaxis* , *ytitle* , and *yfont* .

h Insert the character-string value of a General Image header variable as graphics text at the cursor location.

m Draw non-separable axes, using the current values of the General Image header variables *mapping* , *xaxis* , and *yaxis* .

t Insert a text item at the current cursor location. All graphics text can contain *troff* –format escape sequences as defined in the *comlabel* (3) manual entry in order to display special characters or to change font type, size, or location within a text line.

v Insert the current cursor address or pixel value as a text item at the current cursor location.

x Draw separable *x* –axes according to the current values of header variables *xaxis* , *xtitle* , and *xfont* .

y Draw separable *y* –axes according to the current values of header variables *yaxis* , *ytitle* , and *yfont* .

4.16. Specifying Graphics Parameters

The following subcommands determine the style in which the graphics commands described in the previous section function.

A Set the text drawing angle—a value in degrees counter-clockwise from horizontal.

C Display a color palette—a grid of possible colors with which to draw subsequent graphic items. Select a color by pointing to it with the mouse and clicking the LEFT button. When the **C** subcommand is invoked via a script, note that, if less than 256 colors have been specified in the *xgips* **–n** option, some color indices will represent the same value, e.g. if 32 colors are requested, values 0–7 represent the same (background) color, 8–15 the same, etc.

F Set the current text drawing font—all font files in *vfont*(5) format are acceptable. If a file of that name is not found, *xgips* will look in "/*usr*/*lib*/*fonts*/*fixedwidthfonts*", and then in "/*usr*/*lib*/*vfont*".

N Set the current variable precision—affecting the number of significant digits used to display all floating point values within *xgips* displays.

O Set the current text origin—see the table in section A.2.6.

4.17. Painting Subcommands

In addition to the graphic drawing planes, *xgips* supports a single *paint* overlay, an 8-bit image with the same dimensions as the stored image, whose non-zero-valued pixels will overlay those of the image itself. The paint pixels can be set with a "paint-brush" or with an area-fill subcommand.

W Set the paint-brush width in pixels, i.e. the size of the area that will be filled by the **p** subcommand.

o Fill the paint overlay—set all like-values pixels surrounding the cursor location to the current graphic color. Use this subcommand with care since it is possible for the color to "leak" out of almost closed regions, and there is no "undo painting" subcommand!

p Paint a pixel—set the "paint-brush" region surrounding the cursor location to the current graphic drawing color. If this subcommand is assigned to the LEFT mouse button, the mouse may be dragged across the image with the LEFT button depressed, and the color "painted" into the overlay.

r Read a paint overlay from disk.

w Write a paint overlay to disk—overlays are saved as simple General Images with 8-bit pixels and little additional information.

4.18. Subcommand and Pixel Logging

One reason the *xgips* and *gipstool* user interfaces have the characteristics they do—apart from a certain desire on the author's part not to be sued into penury by Apple Computer Inc.—is that *all* subcommands can be logged to an external file, which can be edited and "played back" at a later time. The format of these "log scripts" is identical to the command input except that (a) a subcommand keystroke can be separated from any additional text by one or more tabs or spaces, (b) single keystroke commands may be followed by newline characters, and (c) control keys may be specified with carat prefixed, i.e. CTRL–C may be entered as the two characters **"^C".** Note that this applies only to input from log scripts—if you type a carat from the keyboard, *xgips* will respond "*Key ^ not assigned*".

Once *xgips* and *gipstool* have displayed their input images, they look for a file named "*.gipstoolrc*" in the user's home directory, and execute it as a command script. This can be overridden by the **–I** command-line option. Unlike an identical script invoked interactively by the **"<"** subcommand, if an error is detected while executing a start-up script, the program will immediately terminate. Similarly, the program can be made to begin logging immediately after displaying the image by specifying the **–o** command line option.

< Read commands from an external file. While the subcommands are being executed, *xgips* will suppress all pop-up windows, although they will be displayed as soon as the file has been read. If the file name begins with **"|",** it is interpreted as a shell command to be run by "/*bin*/*sh*", and the subcommands will be read from its standard output.

Enter a comment—lines in log files beginning with a **"#"** character will be ignored.

> Begin or end command logging—if followed by a file name, subsequent subcommands will be written to that file. If the name begins with a **"|",** it will be interpreted as a shell command to be run by "/*bin*/*sh*", and the subcommands will be passed to its standard input. Finally, specify **">"** without arguments to terminate subcommand logging.

L Begin or end pixel logging—when in effect, each pixel selected by the **l** subcommand will be written to a named file or pipe.

l Write the real-world and/or pixel coordinates and values of the pixel at the cursor location to the log file.

4.19. Miscellaneous Subcommands

^G Invoke the *gipsmongo* graphic display program—it will start up in its own window, and wait for a socket connection from *xgips* or *gipstool*. Currently, the **H**, **I**, and **J** subcommands can write objects to the *gipsmongo* socket, for subsequent display using the powerful *mongo* command language.

f Change or remove the text displayed in the top stripe of the image window. If a zero-length string is specified, *gipstool* will remove its top stripe entirely. This is not the case with *xgips*, since X window stripes are maintained by the window manager.

I Write a sub-image—the rectangular area from the image mark to the current cursor location are written to a disk file as a General Image. Alternatively, if the file name is omitted, the image is written to a *gipsmongo* socket.

J Write a line trace—the *n* pixel values and addresses that lie along a straight line from the image mark to the cursor are written to a disk file as an $n \times 7$ floating point General Image, where the y–dimension represents the line index, and the real-world and pixel values of the x–, y–, and data–coordinates. Alternatively, if the file name is omitted, the image is written to a *gipsmongo* socket.

5. Resampling an Image or Merging Images—the *ihm* Command

ihm is one of the most powerful of the GIPS commands, but its syntax is one of the simplest. You merely tell it what the *output* image header is to look like—it goes ahead and reads one or more input files, transforming them to that output specification. By default, *ihm* uses the header of the first input image to describe the output, but you can augment this with the **–m** flag, or use the **–h** flag to specify a completely fresh header.

5.1. *ihm* as a Filter

ihm invoked as a filter without any options is a null operation...

```
$ gin | ihm | git

 0  1  2  3  4  5  6  7
 8  9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
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
```

A simple use of *ihm* would be to change the size of an image...

```
$ gin | ihm –m "xnum=4 ynum=5" | git –h

header = "Image Header"
version = ieee
dtype = b
dscale = s
xnum = 4
ynum = 5
history = "Sun Jun 30 21:26:22 1985  gin:"
history = "Sun Jun 30 21:26:22 1985  ihm: –m xnum=4 ynum=5"
FINIS=
  0   1   2   3
  8   9  10  11
 16  17  18  19
 24  25  26  27
 32  33  34  35
```

Observe that no pixel interpolation has taken place. This is because, without further information, GIPS routines assume that *xnum* and *ynum* define both the number of *x* and *y* pixels and the real coordinate interval spanned by the image. To tell *ihm* otherwise, you must include variables *xmin*, *xmax*, *ymin*, and *ymax* in the header. Contrast the above with the following...

```
$ gin –m "xmin=0 xmax=8 ymin=0 ymax=8" | ihm –m "xnum=4 ynum=5" | git –h

header = "Image Header"
version = ieee
dtype = b
dscale = s
xnum = 4
xmax = 8
xmin = 0
ynum = 5
ymax = 8
ymin = 0
history = "Sun Jun 30 21:32:43 1985  gin: –m xmin=0 xmax=8 ymin=0
ymax=8"
history = "Sun Jun 30 21:32:44 1985  ihm: –m xnum=4 ynum=5"
FINIS=
  3   5   7   9
 15  17  19  21
 28  30  32  34
 41  43  45  47
 53  55  57  59
```

By supplying the "real-world" *x* and *y* coordinate limits, we have changed the operation asked of *ihm* from

- Transform an 8×8 image with $0 \le x \le 8$ and $0 \le y \le 8$ to a 4×5 pixel image with $0 \le x \le 4$ and $0 \le y \le 5$.

to the following...

- Transform an 8×8 image with $0 \le x \le 8$ and $0 \le y \le 8$ to a 4×5 pixel image with $0 \le x \le 8$ and $0 \le y \le 8$.

i.e. the first operation results in part of the image being removed, while the second performs a resampling.

5.2. *ihm* as a Merger of Images

ihm operates in the same way with multiple input images, inspecting the bounds of their real-world *x* and *y* coordinates and performing the required interpolations. Whenever possible, the operation is synchronous—the first output raster is written as soon as sufficient input data has been read to fill it, but if any input image must

be read in the reverse direction, and if that image comes from an "unseekable" device, e.g. a pipe or a tape, *ihm* will first copy it to a temporary disk file.

ihm usually merges images by averaging the contributions of each input pixel to each output pixel, weighted by the area of overlap. This is a rather inefficient process, unless the **–q** flag is specified, in which case *ihm* merely copies the value of the first input pixel that contributes to each output pixel.

ihm assumes that the images to be merged describe the same physical quantities, transforming the pixels to floating-point "real world" values if necessary before averaging them. If this is not the case—if, for instance, one image represents a mask to be applied to another—you should use the *iha* command instead, e.g.

$ . . . | iha "p = p2 ? p1 : 0" – gi.mask | . . .

in which the piped image is filtered so that pixels corresponding to zero values in "*gi.mask*" are themselves zeroed out. Note that, unlike *ihm*, *iha* requires that all input images have the same dimension as the output image.

The resampling performed by *ihm* is of a *separable* nature—the relation between x–pixel number and real-world x–coordinate must not depend on the y–coordinate value, and *vice versa.* In the next section, we'll describe some GIPS routines that relax these restrictions.

6. Non–Separable Image Transformations

A non-separable image transformation is one in which an input pixel P_{ij} is mapped into an output pixel $P'_{i'j'}$ such that i' depends on both i and j, and so does j'. Such a transformation cannot efficiently be performed on images in the usual GIPS format, and *ihm* cannot be used. Instead, the image must be reformatted and copied to disk by the *ihgfmt* command in a special "cellular" format, optimized for random access. Such an image file can then only be accessed by a subset of GIPS commands—*ihnull* to translate it back into the usual sequential format, the display routines *gipstool* and *xgips*, which can efficiently "roam" around truly enormous cellular images, and the *ihgeom* command, which specializes in non-local transformations.

You can think of *ihgfmt* and *ihgeom* as black boxes, and concentrate on how to supply *ihgeom* with commands that define the transformation to be performed. These may be generated by the *ihgrid* command, so a typical sequence of operations would be. . .

$ . . . | ihgfmt –o /tmp/image
$ ihgrid *options* **/tmp/image | ihgeom /tmp/image | . . .**
$ rm /tmp/image

ihgrid operates in one of three modes. . .

6.1. Rubber Sheet Mode

If no analytic transformation is specified, *ihgrid* reads a series of numbers specifying pairs of x– and y–pixel coordinates called "tiepoints". . .

- input x–pixel address and
- input y–pixel address corresponding to
- output x–pixel address and
- output y–pixel address

If sufficient pairs of input and output tiepoints are supplied, *ihgrid* will write a stream of commands to *ihgeom* to "smoothly" interpolate the image so that the input tiepoints map to the output coordinates. *ihgrid* uses a bi-linear polynomial algorithm and you must supply the order of polynomial to use via the **–n** option. The tie points themselves may be supplied either as binary floating point fields or as ASCII characters, the latter signaled by means of the **–f** option.

6.2. Analytic Mode

If you invoke *ihgrid* with a numeric flag, e.g.

$ ihgrid –9 . . . | ihgeom . . .

it will transform the image according to one of a set of built-in mappings, as shown in Table 3. In the above example, the image will be transformed to a polar stereographic projection. The transformation is defined by a pair of analytic functions, $F_x(x,y)$ and $F_y(x,y)$, that describe the transformation of x and y coordinates. These functions and their inverses are described in the "*Guide to GIPS*" document. The last column in Table 3 shows which other *ihgrid* flags specify the transformation parameters. For example, you would convert an image to a two-parallel Lambert projection via the command

$ ihgrid –7 –l *long* **–p** *lat1* **–P** *lat2* **–r** *scale* **| . . .**

where *long* specifies the central image longitude, *lat1* specifies the first standard parallel, *lat2* the second standard parallel of the projection, and *scale* represents an overall scaling factor. If *scale* is omitted, *ihgrid* attempts to compute a satisfactory value that would result in the entire input image filling 100% of the output area. *Ihgrid* is not sufficiently clever to do this precisely, but you can repeat the command supplying a "fudge-factor" via the **–R** flag—a scaling factor which will be multiplied by *ihgrid*'s initial guess of a suitable **–r** value.

6.3. User-Defined

It is also possible to supply *ihgrid* with your own transformation. An example of this is given in the manual entry for *ihgrid*(1).

Table 3: Analytic Transformations of *ihgrid*

Image Transformation				*Allowed*
Number	*Name*	*Type*	*Description*	*ihgrid options*
1	Gnomonic	Azimuthal	r=tan(d)	–rlp
2	Stereographic	Azimuthal	r=2tan(d/2)	–rlp
3	Orthographic	Azimuthal	r=sin(d)	–rlp
4	Equidistant	Azimuthal	r=d	–rlp
5	Equivalent	Azimuthal	r=2sin(d/2)	–rlp
6	Lambert #1	Conformal	One parallel	–rlp
7	Lambert #2	Conformal	Two parallels	–rlpP
8	Mercator	Conformal	Equatorial	–rl
9	Stereo Polar	Conformal	Stereographic	–rl
10	Mercator	Conformal	Oblique	–rlp
11	Albers #1	Equivalent	One parallel	–rlp
12	Albers #2	Equivalent	Two parallels	–rlpP
13	Lambert Cyl.	Equivalent	Cylindrical	–rl
14	Lambert Polar	Equivalent	Polar azimuthal	–rl
15	Bonne	Equivalent	Pseudo-conical	–rlp
16	Sinusoidal	Equivalent		–rl
17	Werner	Equivalent		–rl
18	User-Defined			
19	Aitoff	Equivalent	Non-conic	
20	Hammer	Equivalent	Non-conic	
21	Briesemeister	Equivalent	Non-conic	
22	Cylindrical	Cylindrical		

7. Scripts, Shells and Nodes

7.1. Executing GIPS Commands from a Script

Because of the ease with which GIPS commands can be strung together to make pipelines, interactive commands can soon outgrow even the history mechanism of */bin/csh*. Where possible, you should therefore run GIPS processes from within a shell script. Consider the following example from a previous section...

$ gin –m "xmin=0 xmax=8 ymin=0 ymax=8" | ihm –m "xnum=4 ynum=5" | git –h

The same commands written as a script, are shown in Table 3, followed by the resulting script output. Notice that the header history entries preserve the structure and readability of the shell commands themselves. We have used the *sh* shell to execute the script. Had we used *csh*, it would have been necessary to escape each newline within the quoted text, which is less readable.

When on-line storage space is limited, the script becomes the record of a given output image, as well as a "workbench" on which to experiment with particular filters. One of the more productive ways of executing GIPS software is to remain in an editor with a copy of the script on the screen, invoking a sub-shell to process the script, waiting until a small portion of the output image appears on your display device, then, if you're not satisfied with the result, hitting CTRL–C to terminate the sub-shell, returning to the editor and changing some of the GIPS flags and parameters.

7.2. Processing Statistics

The **–S** flag is accepted by nearly all GIPS routines. When a routine that has been invoked with this flag completes normally, it writes a single line of processing statistics to the standard error stream, *stderr*, for example...

$ ihc –S –h gi.hdr /dev/rmt1 | ihbox –S bx1 | xgips –S

```
ihc: 8.1u 4.5s 0:12 8% 30+41k 31+22io 51pf+0w 90ss+0m+0r
ihbox: 6.4u 1.2s 0:11 19% 37+38k 23+17io 31pf+0w 75ss+0m+0r
xgips: 3.6u 0.3s 0:11 4% 34+45k 35+16io 25pf+0w 79ss+0m+0r
```

The fields have the following significance...

*x.x***u**	User CPU time used (seconds)
*x.x***s**	System CPU time used (seconds)
x:xx	Wall-clock time elapsed (minutes:seconds)
*xx***%**	Percentage of CPU used by this process
xx+*xx***k**	Size of shared and unshared memory used (kbytes)
xx+*xx***io**	Number of read and write i/o operations (blocks)
*xx***pf**+*x***w**	Number of page faults and swaps
*xx***ss**	Number of system calls
+*x***m**+*x***r**	Number of system swaps and reclaims

7.3. Using GIPS within a Network

Because of its stream architecture, GIPS is ideally suited for *distributed* image processing provided the network transmission rate is high enough. Consider the following scenario...

> Your own node of a local area network has a processor capable of fast floating point operations, but no tape drive or image display. The tape drive is located on the node named *server*, and the image display on node *aed2*. All three nodes support GIPS.

The following code would read an image from tape, add a header, filter it, and display it...

$ cat gi.hdr | rsh server ihc /dev/rmt1 | ihbox bx1 | rsh aed2 ihs –o1

The first *rsh* command invokes *ihc* on the *server* node. *Ihc* reads a binary image array from *server's* tape unit "*/dev/rmt1*" and a header from *server's* standard input, which, by virtue of the *cat* command, is the file "*gi.hdr*"

on your local node. *Rsh* directs the *server* node to return the standard output of *ihc* back through the network, where your own system pipes it into *ihbox*. After filtering, the image is piped through the network to the *aed2* node, where *ihs* is invoked to transform the image to a form suitable for the available display device.

7.4. Like and Unlike Processors

When passing General Images between nodes of a network, it is important to prevent a binary image created by one machine from being mis-interpreted by another processor that uses a different internal representation for binary fields. For this reason, the value of the *version* field that is required of each GIPS header must be the same for every GIPS image processed by a particular system. The only exception to this is the *ihtrans* filter that explicitly transforms an image from a "foreign" version to the local one. For example, the above example running on three unlike processors would be

$ rsh server ihc /dev/rmt1 < gi.hdr | ihtrans | ihbox bx1 | rsh aed2 ihtrans "|" ihs –o1

where the first *ihtrans* is executed on your local node and the second on the *aed2* node. Because the second *rsh* must execute both *ihtrans* and *ihs* on the *aed2* node, it is necessary to quote the **"|"** in its argument list to prevent it from being interpreted by the local shell.

Table 4. A Sample Shell Script

```
#! /bin/sh

gin –m "

    xmin = 0
    xmax = 8
    ymin = 0
    ymax = 8

" | ihm –m "

    xnum = 4
    ynum = 5

" | git –h
```

Output from the above script

```
header = "Image Header"
version = vax
dtype = b
dscale = s
xnum = 4
xmax = 8
xmin = 0
ynum = 5
ymax = 8
ymin = 0
history = "Tue Jul  2 10:32:38 1985  gin: –m

    xmin = 0
    xmax = 8
    ymin = 0
    ymax = 8

"
history = "Tue Jul  2 10:32:38 1985  ihm: –m

    xnum = 4
    ynum = 5

"
FINIS=
  3    5    7    9
 15   17   19   21
 28   30   32   34
 41   43   45   47
 53   55   57   59
```

Appendix
The Generic Display Interface

A.1. The *ihs* Command

This section describes the *ihs* command—a program that uses the "device independent" interface to access a set of raw frame buffers. GIPS users with access to SunView or X11 servers will probably ignore this section. They have their own special *gipstool* and *xgips* commands, documented in Section 4 of this tutorial.

Ihs assumes that you have access to a *generic* image display device, with rather modest capabilities. All access to the device is made via the *comsubs* subroutines in the "/*usr*/*gips*/*lib*/*libFB.a*" library. Only two GIPS commands use this interface—*ihs* for displaying images, and *iged* for adding axes and titling. *iged* will be dealt with in Appendix B. Because *ihs* is so important, we'll go into its various options and modes in considerable detail. It usually appears at the end of a pipeline of GIPS processes, e.g.

$ ihc –h gi.hdr /dev/rmt1 | ihbox bx1 | ihs –o1

A.1.1. The Input Stream

In common with many other GIPS commands, *ihs* takes an optional "*file*" argument, i.e. an argument that does not begin with a minus sign, or immediately follow one of the flags that expects an argument. If "*file*" is specified, *ihs* expects to read a General Image from the file of that name. Otherwise, it reads a General Image from the standard input stream.

Ihs translates the pixels of the input image into a form suitable for the display device. It may also truncate the image at left, right, top, or bottom if the image won't fit, and will add blank borders if necessary. Alternatively, the **–i** option instructs *ihs* to insert the image in the display without blanking out the borders.

Table 5: *ihs* flags and header variables

ihs flag	*Variable*	*Description*
–x	xorg	Display address of x–origin of image
–X	xlim	Display address of right edge of image
–y	yorg	Display address of y–origin of image
–Y	ylim	Display address of bottom edge of image

A.1.2. The Output Streams

The **–o** option of *ihs* specifies the destination of the output image pixels. When followed by a numeric argument, e.g. **–o1** or **–o 1**, *ihs* writes the image to plane number 1 of the display device. If the argument is non-numeric, *ihs* writes to a file of that name. In either case, the header has been stripped away—*ihs* does not write a General Image. The location of the pixels on the screen is determined by a set of GIPS header variables, which may be overridden by *ihs* flags, as shown in Table 5.

The display device is assumed to be rectangular, and display addresses are measured with respect to a (0,0) origin at the top left[6] of the visible screen.

The **–m** option is used to make final changes to the GIPS header before *ihs* determines how to display the image. As we mentioned previously, the **–m** flag can either specify the changes themselves, or the name of a file containing those changes, depending on whether the string following **–m** contains an equals sign. Of course, if the argument following **–m** contains any blanks, newlines, or tabs, it must be enclosed in quotes.

[6] it is easy to forget that the *y*–origin is at the **top** of the display, not the bottom. GIPS uses this definition because most image frame buffers are addressed from top to bottom.

Ihs can also be told to save the GIPS header so that the header can be used as the skeleton for a subsequent *ihc* command, or as input to *iged*. For example...

$ ... | ihs –h– | iged

A.1.3. The Inverted Histogram

Another optional function of *ihs* is to construct an inverted histogram of pixel values that may be used to apply an ''optimal'' contrast stretch to the displayed image. The inverted histogram is a lookup table that, when applied to the image, will result in an even distribution of displayed pixel densities. The histogram is generated by the **–f** option. If your display device features pipelined contrast stretches, *ihs* will interpret a numeric **–f** argument as a command to write the histogram to the lookup table of that number within the display device itself. Otherwise, the **–f** argument is interpreted as the name of the file to which the lookup table is to be written.

A.1.4. Data Flow

The **–c** flag instructs *ihs* to terminate once the displayable portion of the image has been written to the display device. This will cause all prior GIPS routines in the *ihs* input pipe to terminate abnormally because their write–to–*stdout* requests will be unsatisfied. If these processes are run under *csh*, the C–shell, this would normally result in an error message "*Broken Pipe*" from each such process. This will not occur for GIPS routines—they always tell the system to ignore this error condition. If you don't specify the **–c** flag, *ihs* will read the input stream until an end of file.

ihs may write up to three separate output files, as specified by the three flags...

- **–o** The image display pixels
- **–f** The inverted histogram image enhancement table
- **–h** The image header

If **–o** is given a numeric value, (i.e. if the image display is to be updated), the data sets are written in the order listed above. This allows the header to be piped into *iged* after the image has been written to the display, without any possible i/o conflict if *ihs* and *iged* were to write to the display device simultaneously.

A.1.5. Miscellaneous Flags and Options

The remaining *ihs* flags are quickly described: **–n** suppresses the generation of any display file whatever. It implies **–h–** i.e. that the image header will be written to the standard output, but the destination may be overridden by an explicit **–h** specification. The **–l** option specifies the maximum line length to use when writing the image header file. If omitted, **–l1** is assumed, and each header item is followed by a newline character. The **–s** flag tells *ihs* to copy the input image without any conversion whatever. It may be used to remove the header from an image that is to be used outside the GIPS environment. Finally, the **–q** flag suppresses the message

ihs: warning—image truncated on: *left right top bottom*

that *ihs* would otherwise write to the standard error stream when less that all image pixels can be accommodated within the display device.

Table 6: *iged* **Sub–Commands**

!	*command(s)*	1,5	pass a command to a UNIX sub-shell
"	*[char [value]]*	5	store or execute a macro
.	*[command(s)]*		issue a command to the display processor
<	*[[</|] name]*		execute sub-commands from file or pipe
=	*[command(s)]*	4	reissue an *iged* sub-command
>	*[[>/|] name]*		redirect output to a file or pipe
?	*[char]*		get help about *iged* sub-commands
C		2,5	clear the current graphic plane
D	*loc loc ...*	2,5	draw and fill a polygon

F	*[type [name]]*		specify an alternate character font
G	*# oper [^]*	2	merge a graphic into the current graphic
H	*[code]*	4	start or stop histogram mode
L	*loc name [**f**]*	2,4,5	insert a logo into the current graphic
N	*loc*	4,5	report a rounded pixel value
O	*loc loc*	2,4,5	draw a circle
P	*loc #*	4,5	change an image pixel value
R	*[#] command ...*	1,4	repetitively execute a sub-command
S	*loc loc ...*	2,4,5	draw a smooth curve through points
T	*loc code*	2,3,5	write x, y, or data coordinate into graphic
U		1,2,4,5	undo the previous image update
[	*loc loc*	2,5	draw an open rectangular box
]	*loc loc*	2,5	draw and fill a rectangular area
^	*text^ [text^]*	4	edit and reissue a sub-command
a	*[params]*	2,3	draw both axes
b		1	enter block-update mode
c	*[colorcode]*		color & redisplay the current graphic
d	*loc loc ...*	2,5	draw a straight line
e	*loc loc*	2,5	create a grey-scale in the current image
f	*[*****] [name]*		specify a character font name
g	*[#]*	2	select a (current) graphic plane
h	*[name]*		read a new GIPS image header
i	*[#]*	2	select a (current) image plane
j	*loc name*	2,5	insert a header value into graphic
k	*[linetype]*		select a line-drawing pattern
l	*[*****] name ...*		display header variable(s)
m	*[name]*	3	merge a new GIPS image header
n	*loc*	5	report an image pixel value
o	*[#]*		set the relative text-drawing origin
p	*[#]*		specify the numeric precision
q		4	return from *iged* (repeated)
r	*[name/#]*	2	read a graphic
s		1,2	enter single-update mode
t	*loc [text]*	2,5	position target or write a graphic title
u		1,2,4,5	undo the previous graphic update
v	*name [= value]*	3	change or delete a header variable(s)
w	*[name/#]*		write the current graphic
x	*[params]*	2,3	draw x-axis
y	*[params]*	2,3	draw y-axis
z	*name loc ...*	4,5	scan and report pixel values

A.2. The *iged* Command

This command serves the function of a graphic editor for display devices that are accessed via the *consubs* routines located in the GIPS library "*/usr/gips/lib/libFB.a*". Once a General Image has been created, filtered, resampled, merged, and sent to a display device by *ihs*, it can be enlivened with some axes, titles, colors, and so forth. It is also often necessary to *interact* with the image display, to determine the coordinates and values of particular pixels, and to compare pixels within different images. All this and more is available through the *iged* program (Interactive Graphic Editor). SunView and X11/XView users have their own *gipstool* and *xgips* commands that combines the functions of *ihs* and *iged*, so they can omit this Appendix.

Iged may be used in two quite different modes—the sub-commands have the same effect, but they are specified in different ways. The full list of sub-commands and their options is shown in Table 6, and the abbreviations used in that table are explained in Table 7.

Table 7: *iged* **Sub–Command Options**

(a key to the terms used in the second column and the notes in the third column of table 6)

#	positive decimal integer item.
...	one or more repetitions of the preceding field(s).
/	choice of options.
char	single ASCII character.
code	character or characters special to this sub-command.
colorcode	color code peculiar to your display device.
command	UNIX command or another *iged* sub-command,
linetype	decimal or octal integer or dot-pattern string.
loc	address on the display device.
name	a file, a font, or a header variable.
oper	C–like arithmetical operation code.
params	sequence of axis sub-parameter specifications.
text	sequence of ASCII characters
type	*troff* font specification – R,I,B,S.
value	numeric, character or (quoted) string item.
Note 1	Valid only in interactive mode or .igedrc
Note 2	The sub-command updates the current graphic
Note 3	Valid only after a GIPS header has been read
Note 4	Invalid for execution from the *.igedrc* file
Note 5	May be executed repeatedly via the **R** sub-command

A.2.1. *Iged* in Batch Mode

If any arguments follow the *iged* command, it executes in *batch* mode. This means that it expects to read a valid General Image header from the standard input stream, and it executes one or more sub-commands specified by its arguments, i.e.. . .

$ . . . | iged –c_1 *opt*$_1$ **[–**c_2 *opt*$_2$

where c_1, c_2, . . . c_n are a set of single character sub-commands, each of which may be followed by an optional argument. If that argument must begin with a minus sign, you should omit the space between it and the preceding sub-command character.

In batch mode, *iged* executes its arguments as sub-commands until an error condition is encountered or until the argument list is exhausted. *iged* then performs any necessary updates of the image display before exiting. Batch mode is most useful in a fully automated image-production environment. For example, the end of a processing pipe might be. . .

```
$ ihs –o1 –h– –m "

    xtitle = "Longitude (deg)"
    xfont = I.8
    xaxis = "from=–180. to=180. by=60. font=R.6 place=bco scale=d"

    ytitle = "Latitude (deg)"
    yfont = I.8r
    yaxis = "from=60. to=–60. by=–30. font=R.6r place=bco scale=d"

" | iged –C –a
```

in which the six header variables merged by the **–m** option of *ihs* define the titles, scales, and tick marks to be

written along each axis, and the fonts (character styles and sizes) in which to write them. The **–h–** option of *ihs* causes that program to write the updated header to its standard output, from which *iged*, executing in batch mode because it is invoked with options, reads it. The first *iged* sub-command, **–C**, is an instruction to clear the *current graphic plane*,[7] and the second, **–a**, writes the axes, scales, and titles into that plane. The sub-fields of *xaxis* and *yaxis* are discussed below.

Table 8: A Sample Interactive *iged* Session

$ iged	
iged v1.0	
: h gi.one	*read a GIPS header*
: m gi.extra	*merge more header fields*
m: open error: gi.etxra	*an error message*
? m gi.extra	*specify correct file name!*
: a	*draw axes*
: u	*unsatisfactory – undo the axes*
: x from=30.0	*draw x–axis with origin change*
: y by=-15.0	*same with the y–axis*
: u	*unsatisfactory – undo the y–axis*
: y from=60.0 by -20.0	*draw the y–axis again*
: c yel	*color the graphic yellow*
: g 2	*save graphic #1, switch to #2*
: << iged.script	*execute file of sub-commands*
: f B.16	*choose a 16–point bold font*
: o 2	*select the cursor origin*
: t % Image One	*specify a title*
#	*move the cursor, then hit return*
: u	*unsatisfactory – remove the title*
: t +0 +50 Image One	*rewrite the text lower down*
: G 1	*merge graphic 1 into graphic 2*
: w gi.one.graphic	*save graphic in disk file*
: q	*quit from iged*
q: repeat 'q' to exit	*iged asks for confirmation*
? q	*you confirm, and exit*
$	

A.2.2. *Iged* in Interactive Mode

If *iged* is invoked without arguments, it executes in *interactive* mode, reading sub-commands from the standard input. Each sub-command is entered on a separate line. The first byte on each line that isn't either a blank or a tab is interpreted as a sub-command character, although, for compatibility with *batch* mode, *iged* will not complain if that character is immediately preceded by a minus sign.

In *interactive* mode, each sub-command is executed until it completes, until an error is detected, or until you hit the **CTRL–C** key. *Iged* will then issue a prompt and read the next sub-command line from the terminal. To exit, you must enter the single sub-command **q**, followed immediately by a carriage return. You will be prompted to repeat this procedure before you are allowed to exit *iged*. Table 8 shows a sample session, with *iged* replies in **bold** and comments in *italics*.

[7] GIPS assumes that your image display device contains, in addition to the image frame buffer(s), one or more one–byte–per–pixel graphic overlay planes. These planes are referred to by numbers. The default plane is number 1. The **g** sub-command selects the current graphic plane.

A.2.3. *Iged* Program Startup

In *batch* mode, *iged* first reads a General Image header from its standard input stream. Then, in either mode, *iged* attempts to open the display device. In batch mode, failure to open the display will cause *iged* to exit. In interactive mode, the failure will be reported (once), but *iged* will keep going. The program next attempts to read a file "*.igedrc*" in your $HOME directory. If found, it reads and interprets the contents as a series of *iged* sub-commands, one per line. In either mode, an error while executing this startup file will cause *iged* to exit immediately.

A.2.4. Commands that Access General Image Headers

When *iged* is running in interactive mode, it is particularly useful for you to be able to translate automatically between pixel and real-world values. This information must be supplied by a General Image header.

h read a new GIPS image header
l display header variables
m merge a file of GIPS header variables
v change or delete header variables

Reading GIPS headers is the function of the **h** sub-command, which may be augmented by **m**, in the same way as the **–m** option of many other GIPS commands. Individual header variables may then be added, replaced, and deleted by **v**, and displayed by **l**. The **"l *"** sub-command displays the entire header, and "**l *** *name* " writes the current header to disk file *name*.

Table 9: *iged* **Pixel Location Formats**

@	read the location of the display cursor.
%	prompt, then read the location of the display cursor.
x y	*x*– and *y*–coordinates, as follows...
:*n*	pixel address within display area.
n	pixel address within original General Image.
f	pixel corresponding to this "real-world" coordinate value.
±*n*	increment from last pixel address.
>*n*	prompt, then increment *n* from cursor location.
<*n*	prompt, then decrement *n* from cursor location.
	where *n* is an unsigned integer, *f* is a floating-point number, e.g. *1.2*, *–1e3*, etc.

A.2.5. Commands that Reference the Current Graphic

The **g** sub-command selects a particular graphic plane in your display device as the *current graphic,* and the remaining graphic commands refer to this plane. *Iged* keeps copies of the graphic planes that it updates.

C clear the current graphic plane
D draw and fill a polygon
G merge a graphic into the current graphic
L insert a logo into the current graphic
O draw a circle
S draw a smooth curve through specified points
T write x, y, or data coordinate into graphic
[draw an open rectangular box
] draw and fill a rectangular box
c color & redisplay the current graphic
d draw a line
e create a pixel-value scale box

g select a graphic plane
j insert header item value into graphic
r read a graphic
t position the cursor or enter text
w write the current graphic

Many of these sub-commands refer to one or more pixel locations. It is possible to specify these in many ways. If *iged* has already read the GIPS header that was written by the *ihs* command that write the image to the display device, you can specify pixel locations in "real-world" coordinates, as well as absolute pixel addresses. The syntax is shown in Table 9. The characters **"%"** and **"@"** specify an *x*–*y* pair; the other formats specify one or the other, and must therefore appear in pairs: *x*–coordinate followed by the *y*–coordinate. The pair need not be of the same type, but to use floating-point (real-world) values, *iged* must already have read a valid GIPS header.

The **r** and **w** sub-commands may be used to read and write the current graphic in a disk file or frame-buffer plane. If you work in **block** mode (via the **b** command) you must explicitly issue a **w** sub-command to update your displayed graphic.

A.2.6. Commands that Select the Drawing Mode

The manner in which the graphic drawing sub-commands operate may be controlled by the following sub-commands...

F specify an alternate font
f specify a font name
k select line type
o set relative label origin
p specify numeric precision

The **f** sub-command specifies the style and size of font to be used by **t** and **T**, although the size and weight (i.e. Roman, Bold, or Italic) may be altered within the text stream via *troff*–style escape codes. Thus, the string **"\fB"** within a text string writes subsequent string characters in a boldface font, **"\s+2"** increases the point-size by 2, etc. Unlike *troff*, these escape codes only apply within the current text string—if the string ends with boldface selected, *iged* does not remember this when you write the next string.

The **t** and **T** subcommands write their text relative to a specified pixel address. The precise location of that address relative to the text string may be defined by the **o** sub-command. By default, the address represents the location of the top left corner of the text string, but this may be changed as follows:

o 1	top left	**o 2**	top center	**o 3**	top right
o 4	middle left	**o 5**	middle center	**o 5**	middle right
o 6	bottom left	**o 7**	bottom center	**o 8**	bottom right

When *iged* draws lines, it does so according to a "mask" specified by the **k** sub-command. **k** may be given a decimal or octal value, or a character pattern, each signifying a series of 8 bits, some of which are to be set "off", some "on", in all lines subsequently drawn by *iged.* For instance, if any of the following equivalent sub-commands are executed...

k – –xx– –xx
k 51
k 063

subsequent lines will be dashed, with two pixels off, then two pixels on, and so on.

All floating-point numbers written to the terminal or inserted in the current graphic are written with a precision that may be set by the **p** sub-command. The default is 6 decimal digits. To change this to, for instance, 12, use **"p 12".**

A.2.7. Commands that Draw Axes

You may draw the x– and y–axes with the following sub-commands...

a draw both x– and y–axes
x draw the x–axis
y draw the y–axis

The axes are drawn according to the specifications of the header variables *xaxis* and *yaxis*, supplemented by *xtitle*, *ytitle*, *xfont*, and *yfont*, **and** by an option list following the sub-command. The allowed options are as follows:

by=	axis scale increment[9]
font=	character size, orientation and style for scale numbers
form=	*printf* format for axis scale numbers
from=	starting axis coordinate
line=	type of line joining scale marks across the image
loc=	location of an additional axis
mark=	maximum number of scale labels along axis
place=	code describing the position of axes, scales and titles
scale=	code for special scale number formatting
tic=	number of tick marks per scale interval
ticlen=	length of axis scale marks and tick marks
to=	ending axis coordinate

Place= determines where the particular axis is to be placed. Its value is a string containing one or more of the following characters:

b draw an axis at bottom of image
c center the axis titles
i draw scales and titles inside the image area
l left justify the axis titles
m draw axis in mid-image (defined by *loc=*)
o draw scales and titles outside the image area
r right justify the axis titles
t draw an axis at top of image

The **i** and **o** flags are mutually exclusive, as are **c**, **l**, and **r**, but **b**, **m**, and **t** may appear together. The axis lines themselves always stretch the full length or width of the image, but scale marks and values are placed according to the *from*, *to*, and *by* fields, subject to the limit on the total number of scale marks per axis specified by *mark*.[10] If the *by* field is unspecified, *iged* automatically determines an appropriate scale interval, defined as one in which the scale numbers occupy the least number of characters.

Form and *scale* are useful for fancy scale annotations. *Form* controls the appearance of scale notations, in the manner of the C library function *printf*(3). Thus, a *y*–axis representing degrees Kelvin might be displayed as

yaxis=" ... form=%g\(deK ... "

where **"%g"** specifies the output of a floating-point variable, and **"\(de"** represents the degree sign. *Iged* recognizes most of the special character and in-line formatting commands of *troff*. The *scale* field is used when *form* alone cannot specify the correct scale value. *scale* can be given one of the following one–character values...

[9] Coordinates specified as integers will be interpreted as pixel addresses. Coordinates containing a period and/or exponent specify real-world values, in which case *iged* must have already read a valid GIPS header.

[10] The default value of the *mark* operand is 100.

	Default form=	*Description*
s	%.6g	Simple floating-point value
l	%.6g	Logarithm base 10 of value
d	%d\(de%d\´%.12g\´\´	Degrees, minutes, seconds
h	%dh%dm%.12gs	Hours, minutes, seconds

If *scale=l*, the logarithm of the scale value is taken, and *that* value is output in the format specified by *form*, e.g. "*scale=l form=10\u\s–2%.6g*" might generate the following scale...

$10^{1.2}$ $10^{1.3}$ $10^{1.4}$ $10^{1.5}$

If *scale=d* or *scale=h*, the scale value is split into integer degrees (hours) and minutes and floating-point seconds, and these three items are output according to *form*.

A.2.8. Commands that Refer to the Current Image Plane

Unlike graphic planes, *iged* keeps no in-core copy of the current image plane. Each of the following requests results in I/O between *iged* and your display device. If you make a mistake with the **P** or **e** sub-commands, you can restore the original image once only by the **U** sub-command.

N report truncated pixel value
P change image pixel value
e create a pixel-value scale box
i select an image plane
n report pixel value
z scan and report pixel values

The remaining sub-commands write pixel values to the terminal (unless re-directed by the **">"** sub-command). The format is...

x: *xpixel xmin xmax* **y:** *ypixel ymin ymax* **d:** *dpixel dmin dmax*

for the remainder, where *xpixel* represents the integer x–coordinate, *xmin* and *xmax* the corresponding real-world x–coordinate limits, and so on. **N** reports a single "rounded" value instead of minimum and maximum values. The rounded value is determined by an algorithm that seeks to find the shortest unambiguous decimal number that represents the pixel value or coordinate, e.g. if the **n** sub-command reported a pixel as

x: 20 101.31 101.57 y: 119 –17.21 –16.0 z: 6 231.92 278.21

N would report it as

x: 20 101.5 y: 119 –17 z: 6 250

The *y* coordinate is rounded to –17, not –16 because the latter value would be ambiguous with the pixel below.

A.2.9. Histogram Mode

The **H** sub-command is used to switch *iged* into and out of various "histogram" modes. Within such a mode, each time that any *iged* line–, curve–, or figure–drawing sub-command sets a bit in the current graphic plane, the corresponding pixel in the current image is "accumulated". The precise function is governed by one or more key letters that follow the **H** sub-command, i.e...

s *[n]*	accumulate partial horizontal histogram
t	accumulate vertical histogram
v	write each pixel value (output as **n** sub-command)
r	write each pixel value (output as **N** sub-command)
n	don't include *null* pixel values
z	include *null* pixel values

Histogram mode may be terminated by the **H** sub-command entered without options. Otherwise, all sub-modes may be active simultaneously. The effect of the **s** and **t** sub-modes only becomes apparent when you terminate histogram mode, when they display, respectively, horizontal and vertical histograms of the pixel values that have been selected during the time that *iged* was in that sub-mode. Here is an example of the **s** sub-mode used to histogram the pixels within an interactively-defined polygonal boundary...

```
: Hs                                    select s sub-mode
: D%%%%%%%%%%                  select corners of polygon
: H                          terminate mode, draw histogram

14:                          *
13:                         ** ***
12:                       ***     **
11:                     **         *
10:                    *           **
 9:                  *              **
 8:                **                 *
 7:               *                   *
 6:             **                     **
 5:            *                         *
 4:           *                           **
 3:         **                               *  * *
 2:  *    ** *                                  ** * *
 1:   ****  *                                        *
    |---+----+----+----+----+----+----+----+----+----+----+----+--|
    0    20   40   60   80 100 120 140 160 180 200 220 240
:
```

The **t** sub-mode operates in the same way except that the histogram is written vertically, allowing all pixel values to be displayed.

The **r** and **v** sub-modes are designed to be used with the output re-direction subcommand **">".** For instance, the following code would report the average value of a set of pixels within a user-defined square...

```
: Hr                                                  select histogram mode
: >|awk '{s += $9; n++} END {print "avg=" s/n}'      pipe through awk
: ]%%                                                 specify rectangle
: H                                                   terminate mode
: >                                                   terminate awk pipe
avg = 211.2                                           output from awk
:
```

where *awk*(1) is used to accumulate and report the average of the "real-world" pixel values. In practice, if you wanted to many such histograms, you might assign the above **">"** command to an *iged* macro, to save typing it each time.

A.2.10. Meta-Commands

The following sub-commands are used to control the way in which other sub-commands execute.

! pass a command to the shell
" store or execute a macro
. issue a command to the display processor
< read sub-commands from file or pipe
= reissue a sub-command
> redirect output to file or pipe
? obtain help about *iged* sub-commands
R repeatedly execute a sub-command
U undo the most recent image update
^ edit and reissue a sub-command
b enter block-update mode
q return from *iged*
s enter single-update mode
u undo the most recent graphic update

The most recent instance of each sub-command is saved, and may be re-executed by prefixing the command letter with an equals sign. Alternatively, the **"^"** sub-command may be used to edit and then re-execute the most recent sub-command. The syntax is similar to the history mechanism of *csh*. Here is a simple example...

: t % Degree Centigrade	*Insert title into graphic*
#	*user moves cursor, hits return*
: u	*user not satisfied, undoes the title*
: =t	*Re-execute the title command*
t % Degree Centigrade	*iged repeats the command*
#	*user repositions cursor, hits return*
: u	*still not satisfied, removes the title*
: t^ee^ees^	*edits the command and reexecutes*
t % Degrees Centigrade	*iged* repeats the command
#	*user hits return*
:	*the title is inserted correctly*

In this example, the user has made several attempts to write the string "*Degrees Centigrade*" into a graphic plane. *Iged* responds to the presence of the **"%"** character in the **t** sub-command by prompting the user to reposition the cursor on the display device. The user does this, then hits the return key and *iged* accepts the cursor location as the pixel address at which to write the title string.

The **b** and **s** sub-commands switch between *block-update* and *single-step* modes during interactive editing. In block-update mode, the current graphic plane is only updated by a specific **w** sub-command, or when *iged* terminates. In single-step mode, the graphic plane is re-written after each sub-command that alters it. Block-update mode uses less CPU time, but does not give you the "feel" of an interactive editor.

The **"** sub-command is used to define and execute *iged* macros—groups of sub-commands that are to be executed sequentially. For instance, the following command defines the macro **"x"...**

"a –o0 –fR.10 –T%xy –fI.6 –T>0>20d

If you subsequently enter the macro command **"a,** the following sub-commands will be executed:

–o0 Set text origin code = 0, i.e. subsequent text will be centered about the target location.

–fR.10 Use the **"R.10"** font (Roman 10–point) for subsequent text.

–T%xy Prompt the user to move the display cursor, then to hit the enter key. The "real-world" *x* and *y* coordinates of the cursor will be written to the current graphic plane.

–fI.6 Change the current font to 6–point italic.

–T>0>20d Move the cursor 20 pixels below its previous location and enter the "real-world" value of that pixel into the graphic plane.

Glossary

The following paragraphs define technical terms used in this tutorial and in other GIPS documents and manuals. Terms peculiar to UNIX itself, e.g. *file, pipe, script, shell*, etc., have not been included, so you should read (and understand) a UNIX guide before you tackle GIPS!

Address: GIPS recognizes three types of coordinate address—three ways of specifying a location within an image...

name	coordinate system
pixel	image array indices
display	display-screen addresses
real-world	"real" *x,y* coordinates

Thus, *pixel* coordinates are inherently tied to the image array, and are often integer quantities, (although they may take on floating-point values for routines such as *ihgrid* and *ihgeom* which are concerned with non-local transformations), *display* coordinates are only meaningful to those GIPS routines that address a display device, and *real-world* coordinates are inherently floating-point.

Array: a two-dimensional collection of like elements. All GIPS images, except those reformatted by *ihgfmt* for use by *ihgeom*, are represented by an ASCII header followed by an array of binary fields. The dimension of the array is defined by header variables *xnum* and *ynum*. The array is stored with the x–coordinate varying most rapidly, i.e. the first pixel following the header has *x=0*, *y=0*, the next *x=1*, *y=0*, and so on.

Axis: the Cartesian x– and y–directions of the GIPS image array. Axis lines and scales may be drawn and titled via the display commands—*gipstool*, *iged*, or *xgips*.

Azimuthal: a transformation that preserves correct directions (azimuth) from at least one point in the image to all other points.

Beautification: the process of adding color, axes, scales, and titling to an image—the function of a display command.

Borders: the area of a display device surrounding a GIPS image.

Boxcar: a type of separable image transformation using a $(2n+1)\times(2m+1)$ array, replacing each pixel by the convolution of the array with that pixel and its neighbors.

Column: a "vertical" cross-section of a GIPS image, i.e. the set of all pixels with the same x–coordinate value.

Complex: the pixel format resulting from the Fourier transform filter *ihfft*, representing pairs of *cosine* and *sine* coefficients.

Conformal: a transformation that preserves the shape of image pixels.

Coordinate: see *address*, above.

Cursor: a real-time movable pointer on your display device.

Default: the value assumed by a GIPS command for some flag or option that you do not explicitly specify at execution time.

Display: the device on which you view your images. The GIPS routines *ihs* and *iged* assume that the image is stored in a rectangular "frame buffer" within the display device. *Iged* further assumes that this device is capable of displaying 1–bit colorable graphic overlays, and has a movable cursor. *gipstool* and *xgips* are more flexible—they access the display device through the SunView and X11/Xview interfaces, which isolate the programs from the physical characteristics of the display devices. Graphic overlays are implemented in software.

Efficient: is used in the sense of maximum pixel throughput. Most GIPS routines rely on algorithms that yield processing times directly proportional to the number of pixels in a General Image. *Ihfft* is an exception—it varies as $n \log m$ where n represents the number of pixels and m the length of each raster. The efficiency of GIPS commands that use external scratch files is system-dependent.

Equivalent: a transformation that preserves pixel areas.

Escaping: the action of prefixing with a back-slash a character that would otherwise have some special significance; *cf.* **quotation**.

Fast Fourier Transform: a transformation of each raster of an image into its *cosine* and *sine* frequency components. This transform, and its inverse, are performed by the *ihfft* command. Each pixel is transformed into a pair of floating-point numbers, and the header variables *dtype*, and *xlfft* are given the values **c** and *xnum*, respectively.

Filter: any UNIX command whose principal function is to read a data stream from the standard input and write a data stream to the standard output.

Flag: an argument of a GIPS command, signified by a minus sign followed by a single character. A particular command word may be followed by several flags, denoting particular choices of processing modes. Some flags must also be followed by an additional value argument and are referred to as *options*.

Float: the default binary floating point format of the processor that is executing the GIPS command.

Font: one of a set of type-styles available to a display program when constructing axis labels and titles. Under all versions of Berkeley UNIX, these are synonymous with the file names of Berkeley-supplied Versatec or Varian fonts. See *vfont*(5) for more details.

Fourier, J.–B.: 18th century French mathematician. See *Fast Fourier Transform*, above.

Full-word: the largest integer format available on the current processor, synonymous with the "*long int*" definition of its C compiler.

Gaps: image pixels whose data value is undefined. These are given the same numeric value as the *dnull* header variable. The *–n* flag of the *ihbox* command may be used to provide interpolated values for such pixels.

General Image: a stream of data, treated by GIPS commands as a single entity, consisting of an ASCII header followed by binary data, defining a 2–dimensional image.

GIPS: acronym for the *General Image Processing System.*

Graphic: a one–bit–per–pixel overlay to an image frame buffer display, used by the display routines to hold axes, scales, and titles.

Half-word: a binary arithmetic format available on your local processor, synonymous with the "*short int*" definition of its C compiler.

Header: the string of ASCII bytes that begins each GIPS image file, whose sub-fields define the characteristics and processing history of the binary pixel fields that follow.

Histogram: a tally of the frequency of occurrence of pixel values within an image. *cf* **inverted histogram,** below.

History: a variable within a General Image header that is added automatically by each GIPS command that has processed that image. Unlike other header variables, a header may contain multiple *history* entries.

.igedrc: a file in the user's $HOME directory containing *iged* sub-commands to be executed whenever that program is invoked.

Image: within GIPS–any two-dimensional array of data values (pixels); *cf.* **General Image**.

Interpolation: a process for estimating a pixel value at specified *real-world* coordinates, given the values of neighboring pixels. The various GIPS interpolation algorithms and the commands that use them are...

command	algorithm
ihbox	boxcar convolution
ihmed	median of neighbors
ihm *ihgeom*	nearest neighbor
ihm *ihgeom*	bi-linear averaging

Inversion: the reflection of an image about its x–axis, i.e.

Inverted Histogram: that transformation which, when applied to the pixel values of a given image, results in a "flat" pixel-value histogram. If the pixels of a General Image allow a greater dynamic range than your display device, such a transformation will optimize the information content of the displayed image.

Label: within a display program, a numeric indicator inserted at appropriate locations along an axis drawn by the **a**, **x**, and **y** sub-commands. The label format may be specified by the *font=*, *form=*, and *scale=* sub-fields of the *xaxis* and *yaxis* header variables.

Local Transformation: a transformation of an image in which the output value of each pixel depends on its input value, but not on the value of neighboring pixels; *cf* **non–local transformation.**

Lookup Table: an efficient way of performing a local transformation $p' = f(p)$ on byte–size pixels, used by the *iha* command. As p can only take on 256 values p_i, *iha* will precalculate $f_i = f(p_i)$ and use this table to transform the image. A 2–dimensional table $f_{ij} = f(p_i, q_j)$ is used when merging pairs of input images.

Macro: within *iged*, a sequence of sub-commands assigned a single alphabetic byte name, invoked by the **'"'** *sub-command.*

Mapping: another name for an image transformation.

Merge: the combination of two or more images. When performed by *iha*, the images must be of the same size; when by *ihm*, the output image header specifies the output pixel array size and the real-world x– and y–coordinate

Network: in GIPS usage, a means of moving data and commands between processors, especially *ethernet*.

Node: the name by which a particular processor name is known to software that accesses a network.

Non–Local Transformation: an image transformation in which the value of each output pixel may depend on the values of several input pixels; *cf* **local transformation.**

Non–Separable Transformation: a non-local image transformation in which the x– and y–coordinate addresses do not transform independently; i.e. $x' = f_x(x,y)$ and $y' = f_y(x,y)$.All such transformations can, at least in principle, be performed by the *ihgeom* command; *cf* **separable transformation.**

Origin: of the *display* device, the top-left-most pixel of the display buffer; of a GIPS image, the first pixel following the header, with coordinates $x=0$, $y=0$.

Pixel: a single element of the binary array portion of a General Image—the smallest spatially resolved element within an image or display device.

Processor: a CPU capable of executing GIPS commands.

Projection: An image transformation defined by an analytic function or functional. *ihgrid* recognizes 22 built-in projections and allows you to supply one of your own.

Pseudo-color: the transformation of image display pixels to a set of hue–intensity–saturation combinations, usually performed within the display device.

Quotation: the means of including blanks, tabs and newline characters within a single GIPS command argument by enclosing the string within single or double quotation marks; *cf* **escaping**.

Raster: a "horizontal" cross-section of a GIPS image, i.e. the set of all pixels with the same y–index value.

Real-world: a coordinate system in which the x and y pixel indices are physically meaningful, e.g. latitude and longitude, right ascension and declination, stress and strain, etc.

Rectangular Transformation: see **separable transformation,** below.

Resample: an image transformation that results in a change in the number of pixels that represent all or part of the original. This is a rather loose term, used in GIPS documentation to describe the action of *ihm* in changing the size of an input image.

Reversal: the reflection of an image about its y-axis, i.e.

Rotation: the counter-clockwise transformation of an image changing x–axis to y–axis and *vice versa.* i.e.

$$xmax \rightarrow ymin$$
$$xmin \rightarrow ymax$$
$$ymax \rightarrow xmax$$
$$ymin \rightarrow xmin$$

Row: see **raster**, above.

Rubber Sheet Transformation: a non-local non-separable mapping defined by specifying the output coordinates of a set of tie-points *(qv.)* and the input coordinates of corresponding points to be "smoothly" transformed into them.

Scale: the line drawn by a display program into a graphic along the x– or y–direction; the tick-marks and coordinate numbers that may be drawn along that line.

Scratch File: a temporary disk file with a unique name that is automatically allocated, used, and then removed, by several GIPS commands. The user may specify the directory in which the file is to be entered via the **–d** option of that command.

Screen: the visible part of the user's display device; the latter may also contain frame-buffers, pipeline processors, etc.

Separable Transformation: an image transformation in which the x– and y–coordinate addresses transform independently, i.e. $x' = f_x(x)$ and $y' = f_y(y)$. All such transformations can, at least in principle, be performed by the *ihm* command; *cf* **non-separable transformation.**

Stream: a sequential string of bytes—the format of all GIPS images (and most UNIX files), well suited to distributed processing.

Stretch: a transformation applied to the values of pixels in an image display so as to increase the visibility of particular details. See also **inverted histogram.**

Target: see **cursor**.

Terminal: the device by which you interact with the command shell of your local processor, as distinct from the display device that you use to view images.

Tick: within a display program, a short vertical or horizontal bar extending at right angles from an axis line in the direction of an axis **label** *(qv.)*

Tie-Point: a location within an image that can be identified by its pixel value in relation to neighboring pixels, without specifying its coordinates. Tie-points are typically used to overlay or merge images that have been defined with respect to different *x* and *y* coordinate systems, or with respect to the same coordinate system but subject to varying systematic errors. See also **rubber-sheet transformation,** above.

Title: within a display program, a character string that indicates the meaning of an axis, automatically inserted into a graphic via the **a**, **x**, and **y** sub-commands according to header variables *xtitle* , *xfont* , *xaxis* , *ytitle* , *yfont* , and *yaxis* .

Transformation: within GIPS, an operation performed on each pixel of an input image that re-locates it with respect to an output image, often accompanied by a change in pixel value. Transformations that affect only the pixel value, without regard to the values of "nearby" pixels are termed **local**. Of non–local transformations, those that involve separate transformations on the *x* – and *y* –axes are termed **separable**.

Version: a GIPS header variable denoting the type of processor that wrote the image. You must use the *ihtrans* command to translate external images if their version value differs from that acceptable to your local processor.

GUIDE TO GIPS

Peter G. Ford

Center for Space Research
Massachusetts Institute of Technology
37–601, Cambridge, MA 02139

ABSTRACT

The *General Image Processing System* (GIPS), is a software package that has been developed for the UNIX† operating system. It is highly modular, takes full advantage of pipeline processing, and has interfaces to SunView, X11/XView, and "generic" display devices. A variety of image-processing operations are supported—merging, resampling, filtering, etc.—and new functions may be added with ease.

This document describes in some detail the mathematical and computational aspects of GIPS commands and their associated data structures. A companion document, the *GIPS Tutorial*, is written in a more conversational tone and will be more easily accessible to the beginning GIPS user.

February 4, 1992

† UNIX is a trademark of Bell Laboratories.

GUIDE TO GIPS

Peter G. Ford

Center for Space Research
Massachusetts Institute of Technology
37–601, Cambridge, MA 02139

INTRODUCTION

The manuals that describe the many excellent image processing systems available today would have us believe that each is more powerful, more exquisitely versatile than the next. You might be left with the impression that their authors wrote the documentation before the source code. Baloney! Faced with the reluctance of hardware vendors to supply versatile software for their high-resolution image displays, said authors just put together some plausible code, bounced it around their colleagues, wrote up everything that survived, and are now trying to sell it. I claim no superior lineage for the current product.

With this understanding, I shall proceed to describe GIPS as if it had sprung fully clad from the keyboard: beginning with some mathematical definitions, then a description of image formats and of image processing commands, and ending with a survey of the gut-level software. More details are contained in the individual manual files distributed with GIPS, which can be browsed and printed by the *cman* (1) command.[1]

Work on the *General Image Processing System*, a.k.a. GIPS, was begun in 1984 to support a variety of users on a VAX/780 mini-computer and attached Comtal Vision ONE/20 image processor. I picked the name *general* because everybody associated with the hardware suggested a slightly different image format. Their needs also varied with respect to the size of image and whether their had to retain a knowledge of the physical values and locations of processed image pixels. The original version possessed only a very simple display interface, which was implemented by a single set of subroutines called from separate UNIX commands. This has since been augmented by high-level interfaces to SunView and X11 display servers. Nevertheless, the present system still retains its separation between functions that *process* images and those that *display* them.

The intent behind GIPS is that images should be considered as *objects* to be passed from one UNIX command to another accompanied by enough information about their contents that the user need only supply a minimum of additional instructions to perform most image processing operations on them. This philosophy places some restrictions on the range of operations that can be defined, and also limits the degree to which others can be optimized. The payback is a certain elegance of form and ease of use.

Although we shall talk of "transforming" an image, of "re-projecting" it, and so on, most GIPS commands do not *update* image files. Instead, they act as *filters*, writing completely new images that can be passed to other GIPS commands via pipes or network sockets. This may seem to be a waste of storage space, but in practice it results in considerable savings since the intermediate steps need not be saved, and, because the images march from process to process in lock-step, it is often possible to display the first results of a multi-step process, and possibly kill it and start over, before more than a small percentage of the data has been processed.

---

[1] We follow the UNIX documentation conventions—references to manuals are italicized and suffixed with the section number, in parentheses. Section 1 describes UNIX commands, and section 3 describes subroutines.

CONCERNING IMAGES

The digital images that can be processed by GIPS are represented by arrays[2] A_{ij} with integer indices i and j. The array A is usually an approximation to a real function $F(x,y)$, in which coordinates x and y range over some continuous 2-dimensional surface, e.g. latitude and longitude on a sphere. i,j are related to x,y by a pair of transformations, $i=X(x,y)$ and $j=Y(x,y)$, that determine what portion of the (x,y) surface is to contribute to each element A_{ij}. Image processing operations are simplest if these transformation functions are *separable*, i.e. if $i=X(x)$ and $j=Y(y)$, and if they are *invertible*, i.e. if the arrays $\bar{X}$ and $\bar{Y}$, defined by $x=\bar{X}_{ij}$ and $y=\bar{Y}_{ij}$, exist and are single-valued. GIPS does not require that the X and Y mappings be either separable or invertible, but the range of processing possibilities for other types of image will be somewhat restricted.

Another important characteristic of the X and Y mapping functions is their *locality*, i.e. whether $Lim\,|X(x+\delta,y+\varepsilon)-X(x,y)| \rightarrow 0$ as $\delta,\varepsilon \rightarrow 0$ for all x,y. With the exception of cartographic re-projection and Fourier transformation, most image *enhancement* operations are also *local* in nature; that is, they replace each A_{ij} by some expression involving elements A_{kl} where the values of k and l are always close to i and j, respectively. As a general rule, it is much easier to implement local operations on local images than on non-local ones. A simple example of a non-local image, and one that GIPS treats with particular care, is that of an equatorial image of the whole Earth—the software must understand that 360° and 0° represent the *same* longitude, and resampling and filtering operations must be made to "wrap around" the International Date Line.

An image operation can also be non-separable, e.g. the mapping of an image defined by A_{ij}; $i=X(x)$, $j=Y(y)$ to a coordinate system defined by $i=\tilde{X}(x,y)$, $j=\tilde{Y}(x,y)$, where $\tilde{X}$ and $\tilde{Y}$ are explicitly non-separable functions of x and y. An example would be the transformation of a planetary image from Mercator to a polar azimuthal projection. GIPS supports such operations, although it may first be necessary to transform the input image to a "cellular" format that is optimized for non-local access, a topic that we shall return to later in this document.

Compared to the mathematical complexities of coordinate transformation, the relation between the value of the array element A_{ij} and the value of the physical function $F(x,y)$ is much simpler. GIPS distinguishes three separate quantities:

- the *pixel* value, i.e. the value of the stored array element A_{ij}, which may be a fixed or floating point quantity, real or complex.
- the *realworld* value, i.e. the value of the function $F(x,y)$ that is meaningful to the person viewing the image, e.g. *lumens*, *decibels*, etc.
- the *display* value of the element, i.e. the *inherently* fixed-point value to be given to this element when it is represented as a pixel in an appropriate display device.

Many image processing systems do not distinguish between *pixel* values and *display* values, as defined above. The big advantage of the three-level system is that operations involving image *enhancement* can be cleanly separated from those of image *display*, and both can be modularized.

GENERAL IMAGE FORMAT

Most GIPS commands are optimized for *local* operations on local images, in which an $n \times m$ array is physically represented as a sequential set of pixels, beginning with $A_{0,0}$, then $A_{1,0}$ through $A_{n-1,0}$, then $A_{0,1}$ through $A_{n-1,1}$, and ending with the $A_{n-1,m-1}$ element. Such arrays can be passed between UNIX processes as data *streams* and *pipes*. To implement true pipeline processing, the entity called a "image" must be larger than the number of bytes necessary to define the A_{ij} array—since processes "down the pipe" must inherit information about what has happened to the image. In the GIPS system, this is accomplished by prefixing the binary image array with a *header* — a series of ASCII characters that describes the image. The result of appending "image array" to "image header" is a data stream called a *General Image*, which can be stored on disk or tape just like any other UNIX stream, but which

[2] Although we refer to the image structure an array, this does *not* imply that all GIPS images are physically stored as numerical arrays, only that each pixel can be uniquely characterized by two integer indices.

can also be passed (piped) between UNIX processes and eventually displayed, stored, or perhaps thrown away.

There are currently two ways of importing images from the non-GIPS world— either by processing an existing image file through an input *filter*, i.e. one of the programs described in Table 1, or by running the *ihc* (1) command to merge a user-supplied image description with a binary array. Here are a couple of examples:

```
$ ihfromfits /dev/rmt1 –o /usr/me/fits_image.1
$ a.out | ihc –h /usr/me/hdr.1 | . . .
```

In the first line, a FITS[3] image is copied from the tape device "*/dev/rmt1*", and is written to disk as "*/usr/me/fits_image.1*" in General Image format. The second line shows how the *ihc* command can be used to prefix a binary array (the assumed standard output of *a.out*) with a General Image header located in file "*/usr/me/hdr.1*" and pipe the resulting General Image to other processes, represented here by the ellipsis ". . .".

Some general rules governing GIPS command arguments are presented in Table 2. If a command is invoked with a single "=" argument, it writes a usage note detailing its valid options and arguments, e.g.

```
$ ihmed =
Usage: ihmed [–S] [–m file] [–o file] [–x xdim] [–y ydim] [file]
```

Arguments that are not part of recognized flags or option values are interpreted as input file names. There are several ways of specifying a General Image to a GIPS command:

- as a UNIX file name.
- as a dash, "–", to denote the standard input stream, *stdin* (or, as the value of an **–o** option, the standard output stream, *stdout*).
- as a vertical bar, "|", followed by a single-line shell command, which will be passed to the Bourne shell, "*/bin/sh*", and whose standard output stream must itself be a General Image.[4]
- in the case of **–m** and **–h** options, their value can be a General Image header itself, enclosed in quotes to prevent any white space from being interpreted by the command shell.

Here are some examples:

```
$ # Apply a median filter and display it with a linear stretch
$ ihmed /usr/me/gi.test | xgips –m "dmin=20 dmax=22.5"

$ # Display a GIF image merged with a VAX-format GIPS image
$ ihfromgif /usr/me/gif.img | ihm – "| ihtrans /usr/me/gi.vax" | xgips
```

The other processing routines will only read genuine General Images, either to transform them in some way (the commands listed in Table 3), or to write them to a variety of display devices (see Table 4). All the commands described in this document are explained in greater detail in the individual UNIX manual pages accessed by *cman* (1).

For a GIPS image to take part in a *non-local* transformation, it must be stored in a special *cellular* format, which is optimized for easy random access. Such images cannot be piped between UNIX commands—they must exist as disk files. The GIPS commands that create and access them are listed in Table 5.

---

[3] A popular astronomical image format—see the Bibliography.

[4] or the standard input stream, *stdin*, in the case of the **–o** option, *stdout*).

Table 1. GIPS Input Filters

Name	*Function*
ihfromfits	Transform any 2–dimensional FITS image file to General Image format.
ihfromgif	Transform any image from the CompuServe GIF format, saving any accompanying colormap information, but discarding image borders, etc.
ihfrompix	Transform any Sun rasterfile, saving accompanying colormap information.
ihpds	Transform any 2–dimensional image described by a NASA Planetary Data System label. Images whose PDS labels show them to be written in VICAR2 or AIRSAR format will actually be processed by the *ihvicar* or *ihstokes* commands, respectively.
ihstokes	Transform any image from JPL's AIRSAR radar image format.
ihvicar	Transform any 2–dimensional image from JPL's VICAR2 format.

Table 2. Common GIPS Command Options

Option	*Description*
–S	Instructs the command, upon successful termination, to write a one-line message to the *stderr* stream detailing UNIX resource usage.
–a file	The specified file contains auxilliary processing data.
–b	The binary input array is to be *byte-swapped*—consecutive pairs of bytes are to be interchanged.
–d dir	Specifies the name of the directory in which scratch files will be allocated. The default is always "/*tmp*/".
–h file	Supplies a new General Image header (possibly overriding that suplied by an input image). If *file* is specified as a dash, "–", the header will be read from the standard input, *stdin*. The header file may contain an entire Generalized Image or just the header portion.
–m file	Supplies one or more General Image header fields that are to be merged with those from other sources to generate the output header.
–o file	specifies the destination of output from the command. If **–o** is omitted or specified as a dash, "–", output is written to the standard output stream, *stdout*.

Table 3. GIPS Filters

Name	*Function*
iha	Generate an output General Image from an arithmetic operation performed pixel-by-pixel on one or more input images.
ihbox	Apply a spatial filter to a General Image, the filter being defined by an $n \times m$ array.
ihfft	Apply a Fast Fourier Transform filter to each row of a General Image.
ihm	Merge or resample one or more General Images into a single output image.
ihmed	Apply a median transform to a General Image.
ihnull	Copy a General Image, optionally updating its header fields.
ihrot	Rotate (by 90°), reverse (left–right), or invert (top–bottom) a General Image.
ihs	Strip the header from a General Image.
ihtrans	Translate a General Image between processors with incompatible pixel formats.

Table 4. GIPS Display Programs	
Name	*Function*
gipsmongo	Display small General Images using the *mongo* interactive graphics program.
gipsNeWS	Display any General Image image on a NeWS server.
gipstool	Display any General Image image on a SunView server.
iged	Add graphics and text to an image display device that is accessed through the *com-subs*(3) subroutine library.
xgips	Display any General Image on an X11 server.

	Table 5. Commands that operating on Cellular Images
Name	*Function*
gipstool	Display a cellular General Image image on a SunView system.
ihbidr	Describe a NASA Magellan Basic Image Data Record (BIDR) file as a cellular General Image.
ihgeom	Transform an existing cellular General Image according to parameters supplied by *ihgrid*.
ihgrid	Generate a "control stream" of mapping commands so that *ihgeom* can transform a General Image to a particular mapping projection.
ihgfmt	Translate a sequential (non-cellular) General Image to cellular format.
ihnull	Translate any General Image to cellular format.
ihpho	Edit a cellular General Image "in place".
ihproj	Combine the operations of *ihgfmt*, *ihgrid*, and *ihgeom* into one command.
xgips	Display a cellular General Image on an X11 server.

GENERAL IMAGE HEADERS

The header portion of a GIPS image file is written in a simple character string format (e.g. Table 6). It may be created by one of the UNIX editors, (e.g. *vi*(1), or *ex*(1)), or you may write a Fortran or C program to generate it automatically. When writing your own code, you can make use of the header manipulation routines described in *ihread*(3), which also perform many tests on the validity of your parameters. Sample programs are illustrated in Tables 9a and 9b.

The general header format consists of a stream of items—"*name=value*"—where *value* may be a single character, an integer, a floating-point number, or a character string. The *name*, "=", and *value* fields may be surrounded by any number of spaces, tabs, or newline characters. If a space, tab, or newline character appears within a character-string *value* field, the field must be enclosed in double quotation marks. If the field itself contains any double quotation marks, they must each be preceded by a back-slash character.

The allowed header variable names and the type of data that they can represent is defined by a "profile" file. All GIPS commands interpret General Image headers against the definitions in a file named "*cprofile*" in the current working directory, or, if this file doesn't exist, against the file "*$GIPSDIR*/*tables*/*cprofile*".[5] The contents of this file is shown in Table 7. Not all variables defined in "*cprofile*" need appear in any given header (e.g. see Table 6). The significance of the "type" column is described in *ihread*(3). Note that most of the variable names appear in triplets—beginning with the letters *d*, *x*, and *y*—referring to the pixels themselves and to the *x*– and *y*–axes, respectively. There follows a description of the principal header variables.

[5] If the environment variable $GIPSDIR isn't defined, its default value is "/*usr*/*gips*".

Table 6. Contents of a typical GIPS header.

```
header = "Image Header"
version = 1.0
title = "Pioneer Venus Orbiting Radar"
owner = Ford
date = 08/05/83
time = 18:41:19
dtitle = "Planetary Radius (deg)"
dfont = B.10
dtype = b
dscale = s
dnull = 0
xtitle = Longitude
xfont = B.8
xscale = s
xnum = 1024
xorg = 0
xmax = 240
xmin = –120
xaxis = "mark=10 tic=2 font=R.6 place=bco"
ytitle = Latitude
yfont = B.8r
yscale = s
ynum = 512
yorg = 0
ymax = –65
ymin = 75
yaxis = "mark=10 tic=2 font=R.6r place=bco"
history = "Thu Oct 13 10:01:21 1983 ihc: –h i.1"
history = "Thu Jan  5 09:12:13 1984 ihs: –n /data/gi.radius"
FINIS=
```

FINIS is required, and denotes the end of the General Image header. The end is actually composed of 7 characters, "FINIS" followed by an equals sign followed by a newline character. What follows that is the binary image itself.

d/x/yaxis[6] are used by the display commands (see Table 4) to generate axes and labels.[7] The value of each is a character string containing a number of sub-items in the format **"item=value",** separated by blanks or tabs:

- **mark** the maximum number of scale labels that will be written alongside that axis line.
- **tic** the number of tick marks to be inserted between each scale label.
- **font** the font to be used to write the scale labels.
- **place** one or more of the following characters: **"lcriotmb",** according to whether the titles should be **l**eft, **c**enter, or **r**ight justified, **i**nside or **o**utside the image area, and at the **t**op, **m**iddle, and/or **b**ottom of the image.

[6] the slash notation is used throughout this section to denote a choice of leading letters "d", "x", "y", for variables that relate to data pixels, x-axis, and y-axis, respectively. If the choice is restricted to "x" or "y", the variable is written with a leading "x/y", e.g. *x/ynum*.

[7] *daxis* is not currently implemented.

Table 7. Contents of the *cprofile* file
(see *ihread*(3) for a description of the *Type* column)

Variable	*Type*	*Description*
header	**R**	Image Header
version	**R**	ieee msb-754 sun NeWS SANE-D
title	**s**	Title of Image
owner	**s**	Owner of Image
origin	**s**	Source of Image
date	**s**	Creation Date
time	**s**	Creation Time
geometry	**s**	Random-access image descriptors
mapping	**s**	Non-local transformation parameters
history	**h**	History entry
FINIS	**E**	End of header identifier
dtitle	**s**	Description of data variable
daxis	**s**	Full description of data titling (unimplemented)
dfont	**s**	Font in which to write dtitle
dtype	**c**	Pixel length *b*=byte *h*=half *i*=full *f*=float *d*=double
dscale	**c**	Type of data mapping: *s*=std *f*=pixfile *p*=polynomial
dnum	**i**	Number of data components
dmax	**d**	Maximum float data value
dmin	**d**	Minimum float data value
dfile	**s**	Name of file containing byte→pixel mapping
dpoly	**i**	Order of polynomial if datatype=*p*
dparm	**s**	Polynomial coefficients if datatype=*p*
dnull	**d**	Data value denoting none available
dfact	**d**	Pixel scaling factor for dscale=n
dzero	**d**	Pixel baseline value for dscale=n
xtitle	**s**	Description of *x*-variable
xfont	**s**	Font in which to write *xtitle*
xtype	**c**	X-variable type (always *d*)
xscale	**c**	Type of *x*-axis mapping: *s, f, p, l, m, w*.
xnum	**i**	Number of *x*-axis pixels.
xorg	**i**	Lower pixel *x*-limit
xlfft	**i**	Number of *x*-axis pixels before *ihfft*
xlim	**i**	Upper displayed pixel *x*-limit
xmax	**d**	Maximum float *x*-axis value
xmin	**d**	Minimum float *x*-axis value
xwrap	**d**	Limit of *xscale=w* x-axis value
xfile	**s**	File for *xscale=x* mapping.
xpoly	**i**	Order of polynomial if *xscale=p*
xparm	**s**	Polynomial coefficients if *xscale=p*
xaxis	**s**	Full description of *x*-axis
and similarly for *y*-axis variables		

loc defines the location of "middle" titles—i.e. within *xaxis*, *loc* specifies that *y*–value at which the *x*–axis is to be drawn.

from,to,by specify the steps to be used in drawing the axis scale notations. If specified in floating-point form, realworld values are assumed. Otherwise, they are assumed to represent pixel values. The same convention is used to interpret the **loc** value.

line specifies an 8-bit mask used to construct lines connecting the top and bottom of the image at each scale mark.

ticlen specifies the length in pixels of the axis scale and tick marks.

scale specifies a code letter which causes the default format for axis scale numbers to be:

- **s** "*%.6g*"—simple fixed or float numbers.
- **l** "*10\s-2\u%.12g*"—ten to a power
- **d** "*%d\(dg%d'%.12g''*"—degrees, minutes, and seconds; if you override this format via the *form* subfield, you must include (at least) three "%" fields.
- **h** "*%dh%dm%.12gs*"—hours, minutes, and seconds; if you override this format via the *form* subfield, you must include (at least) three "%" fields.

form specifies a non-default format in which to draw the axis scale numbers. The format is the same as that used by the C subroutine *printf*(3), with *troff*-style escape codes described in *comlabel*(3). It should contain no imbedded whitespace characters (blanks, tabs, or newlines).

d/x/yfont file names used to instruct the display commands which fonts are to be used to draw data and axis titles.

d/x/max
d/x/min specify the range of real-world pixel values to be viewed on a display device, and the limits of real-world *x*–axis and *y*–axis coordinates. The precise definitions are shown in Tables 8a and 8b. Note the difference between *dmax* and *dmin* on the one hand, which are ignored by all GIPS filters and *only* influence the displayed image values, and *x*/*ymax* and *x*/*ymin* on the other, which, in the absense of a *mapping* variable, determine the real-world coordinates of each pixel.

d/x/yscale specify the relationship between pixel, row, or column values and the real-world values that they represent.[8] Many GIPS commands ignore the *xscale* and *yscale* values when the image header contains a *mapping* variable.

- **s** *dscale=s* indicates a *standard* pixel mapping, i.e. the pixel values and "real-world" values are identical. *Xscale=s* or *yscale=s* indicates that the corresponding axis pixels linearly span the interval specified by header variables *xmin* to *xmax* or *ymin* to *ymax*.
- **n** defines a *normalized* linear mapping between pixel and real-world value. The relationship is defined by header variables *dzero* and *dfact* (see table 8a). The "n" value is *only* permitted for *dscale*, i.e. *xscale=n* and *yscale=n* are illegal.
- **w** applies only to *x*–axes, indicating that the mapping is linear, but that *xmax* and *xmin* represent the same value, e.g. when *x* is an azimuthal variable (such as longitude) that spans a range of values that *contain* 360°. A different "wrapping" value may be specified by a header variable names **x/ywrap**, whose default values are each 360.
- **l** indicates that the mapping is logarithmic between *d*/*x*/*ymin* and *d*/*x*/*ymax* (which must therefore be positive!).

---

[8] *dscale=p* and *dscale=f* are not acceptable to many of the filter routines, e.g. *ihbox*(1), *ihmed*(1), and *ihfft*(1). Images in these formats must be converted to another *dscale* by filtering them through *ihm*(1) beforehand.

m applies only to y-axes (i.e. *yscale=m*), denoting a Mercator projection between *ymin* and *ymax*. *xscale=f* and *xscale=l* are illegal.

p the mapping between pixel, row, or column and real-world value is defined by the combination of header variables *d/x/ypoly*, *d/x/yparm*, *d/x/ymin*, *and d/x/ymax*. *d/x/ypoly* specifies a non-negative integer n, and *d/x/yparm* specifies a string of coefficients p_i. The real-world value is related to the x, y, or pixel value by the transformations listed in Tables 8a and 8b. If an image with *dscale=p* is to be filtered, it should first be piped through *ihm*(1) to translate the pixels to a linear mapping and so reduce the considerable CPU overhead necessary to apply the polynomial to each pixel.

f as an ultimate in flexibility, *d/x/yscale=f* implies that there exists an auxiliary file, whose name is specified by header variable *d/x/yfile*, that contains a "look-up table". For *dscale=f*, the table contains 257 ASCII numbers (256 if *dtype=b*) that define the intervals between the possible 256 output display values. The intervals are rounded *down* so that a pixel with value equal to the ith table entry will be displayed as pixel value i. For *xscale=f* or *yscale=f*, the file must contain one more entry than the number of rows (columns) of pixels (header variables *xnum* and *ynum*). Any user-supplied header variables *d/x/ymin* or *d/x/ymax* will be overridden by the respective maximum or minimum values of the corresponding coordinates.

Table 8a. Summary of *dscale* definitions

dscale	*Pixel*	*Realworld Value*	*Displayed Image Value*
s	p	$d = p$	$0 \le 256 \times \frac{d - dmin}{dmax - dmin} < 256$
n		$d = dzero + p \times dfact$	
p		$d = \sum_{n=0}^{dpoly} f_n p^n \; ; \; f \in dparm$	
l		$d = p$	$0 \le 256 \times \frac{\log_{10} d - \log_{10} dmin}{\log_{10} dmax - \log_{10} dmin} < 256$
f			$n \; ; \; f_n \le d < f_{n+1} \; ; \; f \in dfile$

Table 8b. Summary of *xscale* definitions

xscale	*Pixel Address i,j*	*Realworld Value x,y*
s, w	$0 \le i < xnum$ $0 \le j < ynum$	$x = xmin + i \times (xmax - xmin)/xnum$
m		$y = \text{atan exp}\{y_0 + j \times (y_1 - y_0)/ynum\} - \pi/2$ $y_0 = \ln \tan(ymin + \pi/2)$ $y_1 = \ln \tan(ymax + \pi/2)$
f		$x = f_i \; ; \; f \in xfile$
p		$x = \sum_{n=0}^{xpoly} f_n i^n \; ; \; f \in xparm$
l		$x = xmin \times 10^{(\log_{10} xmax - \log_{10} xmin) \times \frac{i}{xnum}}$

d/x/ytitle are character strings used by the display programs (see Table 4) to label data and axes.

dnull is used by many GIPS commands, e.g. *ihm*(1), *ihmed*(1), *ihbox*(1), and the display programs, to identify null pixel values. Although *dnull* may be represented in the header by a double-precision quantity, it will be converted to *dtype* before comparing with input pixels.

dtype specifies the format of binary pixels in the image array, as follows:

- **b** unsigned 8-bit integers (bytes) in the range 0 through 255.
- **s** signed 8-bit integers (bytes) in the range –128 through +127.
- **u** unsigned 16-bit integers in the range 0 through 65535.
- **h** signed 16-bit integers in the range –32768 through +32767.
- **l** unsigned 32-bit integers in the range 0 through 4294967296.
- **i** signed 32-bit integers in the range –2147483648 through +2147483647.
- **f** 32-bit floating-point numbers.
- **d** 64-bit floating-point numbers.
- **c** pairs of 32-bit floating-point numbers, typically representing Fourier cosine-sine pairs generated by *ihfft*(1).
- **G** the image is in "cellular" format. Information on the pixel type is contained in the *dtype* sub-field of the *geometry* variable.
- **M** the image is in "unnormalized cellular" format. This is a combination of *dtype=c* and *dtype=G*, but, instead of the pair of floats denoting a complex number, they store an accumulator and weight. If such an image is opened for writing by *gwrite*, both numbers may be updated. However, when opened by *gopen* for reading, each subsequent call to *gcell* causes the accumulator to be divided by the weight and the weight set to zero. Such an image is ideal for many varieties of merge and resample operation.

Whenever a GIPS command changes the *dtype* of a General Image, pixels whose values lie outside the possible range of the new *dtype* are set to the largest or smallest possible value, i.e. when transforming from *dtype=s* to *dtype=b*, pixels with negative values will be transformed to zeroes.

geometry is used to save information about the format of a cellular image, i.e. one created by *ihgfmt*(1) or *ihpho*(1). It will be ignored unless *dtype=G*. Its value is a character string containing the following required sub-fields in the form **"item=value" ,** separated by single spaces:

- **type** the pixel type—see the description of the **dtype** header variable, above.
- **block** the size of each storage cell, in bytes.
- **xdim** the *x*–dimension of each storage cell, in pixels.
- **ydim** the *y*–dimension of each storage cell, in pixels.

header is required in each General Image header, and must have the same value as its description in the "*cprofile*" file. It announces that this is a GIPS image.

history which may appear more than once within a given header, is generated by each GIPS command to create an "audit trail" showing what has happened to the image,

mapping is added to any General Image created by the *ihgeom*(1) command, and sometimes by *ihpds*(1) and *ihvicar*(1), to define the relationship between (*x,y*) pixel addresses and latitude and longitude. Its value is a character string containing the following required subfields in the form **"item=value" ,** separated by single spaces:

- **name** the name of the projection. See *ihgrid*(1) for a list.
- **type** the projection code. See *ihgrid*(1) for a list.
- **origin** a pair of floating-point numbers, separated by a comma, denoting the pixel address in the image of the projection origin. This address may, of course, lie

outside the physical image limits.

radius the projection scaling factor, equal to the angle in radians subtended at the center of body by a pixel at the center of projection, i.e. the radius of the body divided by the pixel width.

euler a set of three floating-point numbers, separated by commas, denoting the three Euler angles (in degrees) through which the body is rotated before applying the mapping projection. The first angle is therefore the east longitude of the projection origin, the second its latitude, and the third is an azimuthal angle through which the body is rotated (counter-clockwise) about the projection axis.

lat a pair of floating-point numbers, separated by a comma, denoting the first and second standard latitudes (in degrees) of the projection. Note that not all projections use these values. Consult *ihgrid*(1) for detauls.

xyrot an optional integer which, if non-zero, performs a 90° counter-clockwise rotation of the projected image. This is most useful when describing transverse projections.

For more information on the use of the *mapping* sub-fields, see the section on "Cartographic Transformations", below.

version is required in each General Image header, and must have the same value as one of the keywords that make up its description in the *cprofile* file. Its value describes the binary format of the image pixels that follow the header, and is defined for a particular CPU architecture, e.g. **vax**, **ieee**, etc. Only the *ihtrans* and *ihnull* routines will read General Image files with **version** values that do not correspond to the local *cprofile* description: *ihtrans* because its function is to translate such images into the local format, and *ihnull* to give the user the chance to over-ride an incorrect **version** value.

x/ylim are used by the display commands (see Table 4) to specify the size of image to be displayed. The top left of the image will be located at display address (*xorg*,*yorg*), and the bottom right at (*xorg+xlim*,*yorg+ylim*). If these addresses lie outside the display area, less than the full image will be displayed.

x/ynum specifies the number of pixel columns (*xnum*), *ynum*). The variable *dnum* is not currently implemented. Note that both *xnum* and *ynum* can be larger (or smaller) than the dimensions of the display device.

x/yorg are used by the display commands (see Table 4) to specify the $x-$ and $y-$pixel addresses in the display area at which the first (top left) General Image pixel is to be written.

Table 6 gives an example of a typical GIPS header—taken from an actual image file. Note that not all variables described in the *cprofile* file need actually appear in the header. Sometimes the absence of a variable causes GIPS routines to assume a default value; other variables have no default (e.g. *dnull*). Finally, some variables, i.e. **title, owner, origin, date,** and **time**, are for annotation purposes only, and do not play any role in GIPS processing.

IMAGE-PROCESSING COMMANDS

The GIPS commands that handle General Images have all been written in the same "style"—many of them accept the same options and all expect to read and/or write honest GIPS-format headers and binary arrays. Table 4 lists some of the options that are common to several GIPS commands. Not all commands accept all options, and the same option may cause somewhat different behavior from command to command. When in doubt, consult the detailed command description in the appendix.

The commands themselves are listed in Table 5. In all cases, a command invoked with a single equals sign as it only argument will generate a "usage" message listing its allowed options and/or other arguments. In addition, the graphic editors, *gipstool*, *gipsmongo*, *iged*, and *xgips* possess interactive help functions.

Creating an image: The simplest input image would consist of an array of binary numbers. You must create a GIPS header, e.g. with an editor, and pass the header and image to *ihs*. You can also use the

–m option to edit the header of an existing image. In the following example, a binary array is in "/*usr*/*me*/*array*", and an almost-suitable General Image is in "/*usr*/*me*/*gi.old*", except that its *ynum* value is 1024 instead of 1280, as required to describe the binary array. You might procede as follows:

$ ihc –h /usr/me/gi.old –m ynum=1280 /usr/me/array | . . .

or alternatively, using *ihnull* (1) to edit the header:

$ ihnull –m ynum=1280 /usr/me/gi.old | ihc –h– /usr/me/array | . . .

If the input array is written in an incompatible format, e.g. VAX integers, and GIPS is executing on a high-order-byte-first machine, you must specify *version=vax* in the header, and then pipe the image through *ihtrans* (1) to translate the pixels to the local format before you can use the image as imput to any other GIPS command.

Ihc will also permit the pixels to be supplied in random order. When the **–w** flag is set, it expects to read triplets of *x* and *y* coordinates and associated pixel values. It will sort and re-sample them according to the definitions in the general Image header that it has been given. Refer to the *ihc* (1) manual for more details.

If the original array contains a header prefix, you will want to convert it to GIPS format. If an input filter already exists for that format, use it. Otherwise use the **–s** option of *ihc* (1) to skip over the input header.

Finally, you may wish to create the image directly from a program. You can do this yourself quite easily by calling the necessary *ihread* (3) routines to supply header values. You don't have to generate the entire header with subroutine calls—the *ihread* routine will first read an existing header from a disk file which you might keep around expressly for the purpose, and all your program has to do is fix it up a bit with calls to *ihput*. Then call *ihwrit* to write the header, *ihbin* to write the rasters, and *ihbcl* to close the file or pipe.

Merging an Image: This is the function of the *iha* (1) and *ihm* (1) commands. *Iha* requires that each input image have the same number of *x* pixels and *y* pixels, whereas *ihm* will merge partially overlapping images together. These commands also differ in the way they transform and merge each pixel. *Iha* is given a script written in C–language, specifying an algorithm relating the output pixel, p, to the corresponding input pixels p_1 through p_n of the n input images. In contrast, *ihm* either performs a straight averaging of the real-world pixel values (taking account of the percentage of each input pixel's contribution to each output pixel), or it performs a logical bit-string assignment of input pixels to output.

In the simplest case, only a single input image is specified. This doesn't sound very exciting (to merge an image with itself?), but it is a very useful operation since you can change the image format in the process. For instance, *ihm* will change *x*– or *y*–axis scales from linear to logarithmic or *vice-versa*, will apply polynomial corrections to pixel data values, and so on. Sometimes a format change is merely cosmetic—to "beautify" a display. At other times, it is essential—the filter commands *ihbox* (1), *ihmed* (1), and *ihfft* (1) will only accept images in *dscale=s* or *dscale=n* format, so any other image must be filtered through *ihm* before *ihbox*, e.g.

$ . . . | ihm –m "dscale=s dtype=f" | ihbox fill.3x3 | . . .

When *iha* and *ihm* are invoked with multiple input image arguments, they combine them into a single output image, whose characteristics are inherited from the header of the *first* specified input image, but you can change the this by **–h** and **–m** options.

Filtering an Image: The simplest filter command is *ihbox* (1). It applies an $n \times m$ convolution array B to each General Image array element a_{ij} according to the transformation

$$a_{ij} \rightarrow a'_{ij} = \left\{ \sum_{k=-n/2}^{k=n/2} \sum_{l=-m/2}^{l=m/2} B_{kl}\, a_{i+k\ j+l} \right\} \Big/ \left\{ \sum_{k=-n/2}^{k=n/2} \sum_{l=-m/2}^{l=m/2} |\, B_{kl}\, | \right\}$$

Note that n and m must be odd and greater than 1. If header variable *dnull* is defined, pixels with that value will be dropped from the summations. If this leads to a zero denominator on the right-hend side,

a_{ij} will be replaced by *dnull*. *Ihbox* must be passed the name of a file containing the array dimensions and elements in the order

$$n, m, B_{11}, B_{12}, \ldots, B_{1n}, B_{21}, \ldots, B_{22}, \ldots, \quad etc$$

Alternatively, it can be passed the *values* themselves, enclosed in quotation marks to protect them from shell interpretation.

The simplicity of the boxcar filter is deceptive—very powerful operations can be easily defined, as the following examples will show:

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{low–pass filter}$$

$$B = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \quad \text{high–pass filter}$$

$$B = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \quad \text{partial–derivative}$$

Another filtering routine, *ihfft*(1), takes a Fourier transform of a General Image, i.e.

$$a_{ij} \rightarrow a'_{ij} = \left[\frac{1}{\sqrt{n}} \sum_{l=0}^{n} a_{ik} \cos\frac{l\pi}{k}, \quad \frac{1}{\sqrt{n}} \sum_{l=1}^{n} a_{ik} \sin\frac{l\pi}{k} \right]$$

Whatever the pixel-type of the input image, *ihfft* will transform it to the special *dtype=c* complex format—pairs of single-precision floating point numbers. If you want to take a 2-dimensional Fourier transform, you must follow the first *ihfft* by *ihrot*(1) command to rotate the image counter-clockwise, then use a second *ihfft*. The output may be transformed back to spatial coordinates by passing it through *ihfft* again, this time using the **–i** option to specify the inverse Fourier transform. An entire pipeline sequence might look like this:

. . . | ihfft | ihrot –t | ihfft | iha filt01 | ihfft –i | ihrot –ri | ihfft –i | . . .

where other processes are assumed to the left and the right, and the file *filt01* is assumed to contain a script that applies some Fourier-domain filtering to the image.

Non-local operations: The *ihgeom*(1) command performs non-local General Image mappings. The transformation itself must be supplied to *ihgeom* in the form of a *control grid* file. Also, the input General Image must be re-formatted by the *ihgfmt* command so that *ihgeom* can efficiently access its pixels in "random" sequence. These commands are described in detail in the section dealing with Image Projections.

Displaying an Image: After a General Image is created, merged, and filtered, it may be displayed by one of the commands listed in Table 5. If you possess a SunView or X11 window server, you should use *gipstool* or *xgips* to display GIPS images as windows on your screen. You don't need to read the remainder of this section. Otherwise, you must use the *ihs* and *iged* commands, which interact with a "generic" frame buffer through the *comsubs*(3) interface routines.

The *ihs*(1) command converts the input pixels to real-world values according to the conventions governing the *dscale* header variable, discussed above. It then converts them to 8–bit integers by mapping pixels with value equal to header variable *dmin* to zero and those equal to *dmax* to 255, and maps the remainder linearly between these limits. The integers are written to the frame buffer. The *ihs* command can also write the image header to the destination of your choice by means of its **–h** option. The header is still useful to you—it contains much information about the image you are displaying, and the *iged*(1) graphic editor is specifically designed to take advantage of this.

Iged may be used in one of two distinct modes—*pipeline* or *interactive*. It enters pipeline mode whenever it is invoked with arguments. Otherwise, it is in interactive mode. In pipeline mode, it

expects to read a GIPS header from its standard input. It interprets its arguments as a series of commands. Each command begins with a dash followed by a command letter. In interactive mode, it accepts the *same* command syntax from the standard input (usually the user's terminal), except that the command letters do not have to be preceded by a dash. Instead, each command ends with a carriage return.

CARTOGRAPHIC PROJECTIONS

The projections that are generated by *ihgrid* (1), and applied to General Images by *ihgeom*, may be grouped under the headings of *Azimuthal*, *Conformal*, and *Equivalent*, according to whether they preserve (some) directions, shapes, or areas. Each transformation maps the latitude θ (the y–coordinate) and longitude λ (the x–coordinate) of a spherical body of radius R into Cartesian map coordinates X and Y. Before performing the transformation, *ihgrid* rotates the (λ,θ) coordinate system according to Euler angles (λ_0, θ_0, and ρ_0), such that the origin of longitude becomes λ_0, the origin of latitude becomes θ_0, and ρ_0 defines an additional clock-wise rotation about the new origin. In addition, Lambert conic, Albers, and Bonne projections depend on one or two *standard parallels*, θ_1 and θ_2.

AZIMUTHAL PROJECTIONS

1. Gnomonic:

$$X = R\tan\lambda, \quad Y = \frac{R\tan\theta}{\cos\lambda}, \quad \theta = \tan^{-1}\frac{Y}{\sqrt{R^2+X^2}}, \quad \lambda = \tan^{-1}\frac{X}{R}$$

2. Stereographic:

$$X = \frac{2R\cos\theta\sin\lambda}{1+\cos\theta\cos\lambda}, \quad Y = \frac{2R\sin\theta}{1+\cos\theta\cos\lambda}, \quad \theta = \sin^{-1}\frac{4RY}{4R^2+X^2+Y^2}, \quad \lambda = \sin^{-1}\frac{X\tan\theta}{Y}$$

3. Orthographic:

$$X = R\cos\theta\sin\lambda, \quad Y = R\sin\theta, \quad \theta = \sin^{-1}\frac{Y}{R}, \quad \lambda = \sin^{-1}\frac{X\tan\theta}{Y}$$

4. Azimuthal Equidistant:

$$X = R\,\delta\cos\alpha, \quad Y = R\,\delta\sin\alpha, \quad \text{where} \quad \cos\delta = \cos\theta\cos\lambda, \quad \tan\alpha = \tan\theta\,\mathrm{cosec}\,\lambda$$

$$\theta = \sin^{-1}\left[\frac{Y}{\sqrt{X^2+Y^2}}\,\sin\frac{\sqrt{X^2+Y^2}}{R}\right], \quad \lambda = \sin^{-1}\frac{X\tan\theta}{Y}$$

5. Azimuthal Equivalent:

$$X = \frac{\sqrt{2}R\cos\theta\sin\lambda}{\sqrt{1+\cos\theta\cos\lambda}}, \quad Y = \frac{\sqrt{2}R\sin\theta}{\sqrt{1+\cos\theta\cos\lambda}}, \quad \theta = \sin^{-1}\frac{Y\sqrt{4R^2-X^2-Y^2}}{2R^2}, \quad \lambda = \sin^{-1}\frac{X\tan\theta}{Y}$$

CONFORMAL PROJECTIONS

6. Lambert Conical *(One Standard Parallel, θ_1):*

$$X = \rho\sin(\lambda\sin\theta_1), \quad Y = R\cot\theta_1 - \rho\cos(\lambda\sin\theta_1)$$

$$\text{where} \quad \rho = R\cot\theta_1\left[\tan\left(\frac{\pi}{4}-\frac{\theta}{2}\right)/\tan\left(\frac{\pi}{4}-\frac{\theta_1}{2}\right)\right]^{\sin\theta_1} = \sqrt{X^2+(Y-R\cot\theta_1)^2}$$

$$\theta = \frac{\pi}{2} - 2\tan^{-1}\left[\tan\left(\frac{\pi}{4}-\frac{\theta_1}{2}\right)\left\{\frac{\rho}{R\cot\theta_1}\right\}^{1/\sin\theta_1}\right], \quad \lambda = \frac{\tan^{-1}(X/(Y-R\cot\theta_1))}{\sin\theta_1}$$

7. Lambert Conical *(Two Standard Parallels, θ_1 and θ_2):*

$$X = \rho\sin(\lambda\sin\theta_1), \quad Y = \rho_0 - \rho\cos(\lambda\sin\theta_1)$$

$$\text{where}\quad \rho = C\left\{\tan\left(\frac{\pi}{4}-\frac{\theta}{2}\right)\right\}^{\sin\theta_1},\quad \theta_1 = \sin^{-1}\frac{\log(R\cos\theta_1)-\log(R\cos\theta_2)}{\log\tan\left(\frac{\pi}{4}-\frac{\theta_1}{2}\right)-\log\tan\left(\frac{\pi}{4}-\frac{\theta_2}{2}\right)}$$

$$\text{and}\quad \rho_0 = Y-\sqrt{\rho^2-X^2},\quad C = \frac{R\cos\theta_1}{\sin\theta_1\{\tan\left(\frac{\pi}{4}-\frac{\theta_1}{2}\right)\}^{\sin\theta_1}} = \frac{R\cos\theta_2}{\sin\theta_1\{\tan\left(\frac{\pi}{4}-\frac{\theta_2}{2}\right)\}^{\sin\theta_1}}$$

$$\theta = \frac{\pi}{2}-2\tan^{-1}\left\{(\rho/C)^{1/\sin\theta_1}\right\},\quad \lambda = \frac{1}{\sin\theta_1}\tan^{-1}\frac{X}{Y-\rho_0}$$

8. Mercator:

$$X = R\lambda,\quad Y = R\log\tan\left(\frac{\pi}{4}+\frac{\theta}{2}\right),\quad \theta = 2\tan^{-1}\exp\frac{Y}{R}-\frac{\pi}{2},\quad \lambda = \frac{X}{R}$$

9. Polar Stereographic:

$$X = 2R\sin\lambda\tan\left(\frac{\pi}{4}-\frac{\theta}{2}\right),\quad Y = 2R\cos\lambda\tan\left(\frac{\pi}{4}-\frac{\theta}{2}\right),\quad \theta = \frac{\pi}{2}-2\tan^{-1}\frac{\sqrt{X^2+Y^2}}{2R},\quad \lambda = \tan^{-1}\frac{X}{Y}$$

10. Mollweide:

$$X = \frac{2\sqrt{2}}{\pi}R\lambda\cos c,\quad Y = \sqrt{2}R\sin c\quad \text{where}\quad 2c+\sin 2c = \pi\sin\theta$$

$$\theta = \sin^{-1}\frac{2c+\sin 2c}{\pi},\quad \lambda = \frac{\pi X}{2\sqrt{2}R\cos c}\quad \text{where}\quad c = \sin^{-1}\frac{Y}{\sqrt{2}R}$$

EQUIVALENT PROJECTIONS

11. Albers *(One Standard Parallel,* θ_1*):*

$$X = \frac{R}{\sin\theta_1}\sin(\lambda\sin\theta_1)\sqrt{1+\sin^2\theta_1-2\sin\theta\sin\theta_1}$$

$$Y = \frac{R}{\sin\theta_1}\cos(\lambda\sin\theta_1)\{\cos\theta_1-\sqrt{1+\sin^2\theta_1-2\sin\theta\sin\theta_1}\}$$

12. Albers *(Two Standard Parallels,* θ_1 *and* θ_2*):*

$$X = \rho\sin\left\{\frac{(\sin\theta_1+\sin\theta_2)\lambda}{2}\right\},\quad Y = \rho_0-\rho\cos\left\{\frac{(\sin\theta_1+\sin\theta_2)\lambda}{2}\right\}$$

$$\text{where}\quad \rho = \left\{\rho_1^2+\frac{4R^2(\sin\theta_1-\sin\theta)}{\sin\theta_1+\sin\theta_2}\right\}^{1/2} = \left\{\rho_2^2+\frac{4R^2(\sin\theta_2-\sin\theta)}{\sin\theta_1+\sin\theta_2}\right\}^{1/2}$$

$$\rho_1 = \frac{2R\cos\theta_1}{\sin\theta_1+\sin\theta_2},\quad \rho_2 = \frac{2R\cos\theta_2}{\sin\theta_1+\sin\theta_2},\quad \rho_0 = Y-\sqrt{\rho^2-X^2}$$

$$\theta = \sin^{-1}\{\sin\theta_1+\frac{1}{4R^2}(\rho_1^2-\rho^2)(\sin\theta_1+\sin\theta_2)\},\quad \lambda = \frac{2}{\sin\theta_1+\sin\theta_2}\tan^{-1}\frac{X}{\rho_0-Y}$$

13. Lambert Cylindrical:

$$X = R\lambda,\quad Y = R\sin\theta,\quad \theta = \sin^{-1}\frac{Y}{R},\quad \lambda = \frac{X}{R}$$

14. Lambert Polar Azimuthal:

$$X = 2R\sin\left(\frac{\pi}{4}-\frac{\theta}{2}\right)\sin\lambda,\quad Y = 2R\sin\left(\frac{\pi}{4}-\frac{\theta}{2}\right)\cos\lambda,\quad \theta = \frac{\pi}{2}-2\sin^{-1}\frac{\sqrt{X^2+Y^2}}{2R},\quad \lambda = \tan^{-1}\frac{X}{Y}$$

15. Bonne Pseudo-Conical *(One Standard Parallel, θ_1):*

$$X = \rho\sin\left(\frac{R\cos\theta}{\rho}\lambda\right),\quad Y = R\cot\theta_1 - \rho\cos\left(\frac{R\cos\theta}{\rho}\lambda\right)$$

where $\rho = R\cot\theta_1 - R(\theta-\theta_1) = \sqrt{X^2+(Y-R\cot\theta_1)^2}$

$$\theta = \cot\theta_1 + \theta_1 - \rho/R,\quad \lambda = \frac{\rho}{R\cos\theta}\tan^{-1}\frac{X}{Y-R\cot\theta_1}$$

16. Sanson-Flamsteed Sinusoidal:

$$X = \lambda R\cos\theta,\quad Y = R\theta,\quad \theta = \frac{Y}{R},\quad \lambda = \frac{X}{R\cos\theta}$$

17. Werner:

$$X = R(\frac{\pi}{2}-\theta)\sin\{\lambda\cos\theta/(\frac{\pi}{2}-\theta)\},\quad Y = R(\frac{\pi}{2}-\theta)\cos\{\lambda\cos\theta/(\frac{\pi}{2}-\theta)\}$$

$$\theta = \frac{\pi}{2} - \frac{\sqrt{X^2+Y^2}}{R},\quad \lambda = \frac{1}{\cos\theta}(\frac{\pi}{2}-\theta)\sin^{-1}\{X/R(\frac{\pi}{2}-\theta)\}$$

MISCELLANEOUS NON-CONIC PROJECTIONS

19. Aitoff

$$X = 2|\alpha|\cos\theta\sin\frac{\lambda}{2},\quad Y = |\alpha|\sin\theta,\quad 2\alpha^{-2} = 1+\cos\theta\cos\frac{\lambda}{2}$$

$$\theta = \sin^{-1}\rho y,\quad \lambda = 2\tan^{-1}\frac{\rho x}{2(2\rho^2-1)},\quad \rho = \sqrt{1-\frac{x^2}{16}-\frac{y^2}{4}}$$

20. Hammer

$$X = 2\rho\sin b,\quad Y = \rho\cos b,\quad \cos b\sin\alpha = \sin\theta$$

where $\cos\alpha = \cos\theta\cos\lambda$, $\cos\beta = \cos\theta\cos(\lambda/2)$, and $\rho = 2\sin(\beta/2)$

21. Briesemeister

$$X = \frac{7}{4}\rho\sin b,\quad Y = \rho\cos b,\quad \cos b\sin\alpha = \sin\theta$$

where $\cos\alpha = \cos\theta\cos\lambda$, $\cos\beta = \cos\theta\cos(\lambda/2)$, and $\rho = 2\sin(\beta/2)$

USER SOFTWARE WITHIN GIPS

General Image headers and data arrays are easy to manipulate within user programs written in Fortran 77 or C and linked with the "*/usr/gips/lib/libGI.a*" library. Tables 9a and 9b show Fortran and C versions of a simple image-copying program. The first step is to read the image header into memory, using *ihread*, which returns a handle (an INTEGER*4 in Fortran, and a pointer to a FILE structure in C). Header variables can then be inspected and changed via calls to *ihget* and *ihput*, respectively. Fortran callers may then read and write image lines by passing the appropriate image handle to *ihbin* and *ihbout*. The same routines may also be called from C, although the individual image pixels may also be read or written using the handle in calls to the standard C I/O library, e.g. *getc*, *fread*, *putc*, *fwrite*, etc. Finally, it is *essential* to close the image with call to *ihbcl*, which performs the appropriate action according to whether the image is a file, string, or pipe. These subroutines are fully documented in the *ihread*(3) manual entry.

Once you have read General Image pixels into a program, you can use another set of "canned" subroutines to convert back and forth between "image" and "real-world" coordinates. These routines are called *gips* and *egips*, and both are to be found in the "*libGI.a*" library. They are described in *gips*(3).

Table 9a. Sample GIPS program written in Fortran

```
      character*4097 buf
      kin = ihread("gi.test", "cprofile")
      if (kin.eq.0) goto 4
      call ihput("history", "Copied to ""gi.test2""")
      kout = ihwrit("gi.test2", 1)
      if (kout .eq. 0) goto 4

1     in = ihbin(buf, kin, 4097)
      if (in) 4, 3, 2
2     if (in .gt. 4096) goto 5
      iout = ihbout(buf, kout, 4097)
      if (iout) 4, 4, 1

3     call ihbcl(kout)
      call ihbcl(kin)
      stop
4     call iherrm
      stop 1
5     write(0, '("raster GT 4096 bytes")')
      stop 2
      end
```

Table 9b. The same program written in C

```
#include    <stdio.h>
#include    "/usr/gips/include/gips.h"
main()
{
    char  buf[4097];
    FILE  *kin, *kout;
    int   i, in, out;

    if ((kin = ihread("gi.test", "cprofile")) == NULL)
        gipserr();
    ihput("history", "Copied to \"gi.test2\"");
    if ((kout = ihwrit("gi.test2", 0)) == NULL)
        gipserr();
    for (i = 0; ; i++)
            if ((in = ihbin_(buf, &kin, 4097)) <= 0 || in > 4096 ||
                (iout = ihbout_(buf, &kout, 4097)) <= 0)
                    break;
    if (in < 0 || iout < 0)
        gipserr();
    if (in > 4096)
        fprintf(stderr, "raster GT 4096 bytes\n");
    else
        fprintf(stderr, "copied %d rasters\n", i);
    exit(0);
}
```

Another subroutine that is often useful in working with pixels is *gconv*, which converts from one pixel type to another. For instance, to convert an input raster of *xnum* pixels of *dtype* in the array *inbuf* to an array of real-world values in the *double* array *dbuf*, use

gconv(dbuf, inbuf, xnum, CVT_D, ihdcvt(dtype));

In this way, you can write GIPS commands that will work correctly with any valid type of input image pixel.

The *comlabel*(3) routines in "*libGI.a*" are based on routines written for a 3M Comtal image processor by Arthur Olson of NCI, Bethesda. They are used to generate character string titles and special symbols using a set of digitized type fonts from U. Cal. Berkeley. The subroutines cannot be called from Fortran—only from C.

At a yet higher level are the routines used for non-local image operations: *gopen*, *gcell*, and *gclose*. A cellular General Image should be opened by a call to *ihread*, and the returned FILE pointer should be passed to *gopen*, which returns a *gimage* pointer. Subsequent calls to *gcell* with this pointer will return pointers to individual pixels. The retrieval is optimized for "local" access, i.e. each pixel that is accessed is in the neighborhood of the previous request.

The "*/usr/gips/lib/libFB.a*" library contains a simple device-independent software package described in *comsubs*(3). These routines, which are only callable from C programs, serve to insulate the user from the physical display device. The *ihs* and *iged* programs use this package when interacting with a frame buffer, so that device-dependent features may be localized. To customize GIPS for your installation, you have only to rewrite these routines, and to recompile *ihs* and *iged*. The physical dimensions of the display, number of image and graphic planes, etc. are defined in "*gips.h*".

If errors are detected by GIPS subroutines, C callers can inspect the external variable *gipserrno*. "*gips.h*" contains symbolic definitions of the error codes and macros for formatting messages and for exiting gracefully. Since *gipserrno* is inaccessible to Fortran callers, they have their own error routine, *iherrm*, which should be called if *ihread* or *ihwrit* return zero-valued handles. This routine writes an error message to the standard error stream.

Finally, here is a "complete" GIPS application, i.e. one that should execute correctly on any supported CPU and accept any valid General Image. The program copies an image that is described by a *mapping* header variable, setting to null (i.e. to the value of *dnull*) any pixel that is "outside" the cartographic mapping.

```
#include    <stdio.h>
#include    <signal.h>
#include    "/usr/gips/include/gips.h"
        /* Flag fields filled by gipsflags() */
static char  Dflag;             /* debug output */
static char  Sflag;         /* print statistics */
static char  vflag;         /* be verbose */
static char  *mfile         = "-"; /* merge file */
static char  *ofile = "-"; /* output file */
        /* Parse table */
static flagentry    flagtable[] = {
        "D",  "1",          (char *) &Dflag,
        "S",  "1",          (char *) &Sflag,
        "v",  "1",          (char *) &vflag,
        "m",  "-> file",    (char *) &mfile,
        "o",  "-> file",    (char *) &ofile,
        NULL,     "[file]",     NULL
};
static char  *ifile = "-"; /* input image file */
extern       int    exit();      /* C library */
```

```
main(argc, argv)
        int     argc;           /* number of arguments */
        char    **argv;             /* array of argument pointers */
{
        FILE *fp;           /* input image stream */
        FILE *fm;           /* merged image stream */
        FILE *fq;           /* output image stream */
        char    *inbuf;             /* input buffer */
        int     xnum, ynum;         /* image size */
        char    dtype;              /* pixel type */
        int     dlen;           /* pixel length */
        double      lonlat[2];  /* pixel coordinates */
        double      dnull;          /* null pixel value */
        double      null;           /* pixel value of dnull */
        int     n;              /* number of pixels fixed */
        int     x, y;           /* pixel address */

        /* Parse callers argument list */
        saveargs(argv);
        argc = gipsflags(argc, argv, flagtable);
        if (argc > 2)
                error("too many args:", argv[2]);
        if (argc == 2)
                ifile = argv[1];
        if (Sflag)
                telltale(0);

        /* Open the input image */
        if ((fp = ihread(ifile, GIPSPROFILE)) == NULL)
                gipserr2(GIPSBADOPEN, ifile);
        if (fp == stdin)
                ifile = "'stdin'";
        putargs();

        /* Merge any additional information */
        if (*mfile != '-')
                if ((fm = ihmerg(mfile)) == NULL)
                        gipserr();
                else
                        ihbcl(fm);

        /* Get image characteristics */
        if (g_setup())
                gipserr();
        if (ihget("xnum", &xnum) | ihget("ynum", &ynum))
                gipserr1(GIPSBADVAL);
        if (ihget("dtype", &dtype, 1))
                gipserr1(GIPSBADVAL);
        if ((dlen = ihdtype(dtype)) == 0)
                gipserr1(GIPSBADTYPE);

        /* Get null pixel value */
        if (ihget("dnull", &dnull, 1))
                dnull = 0;
        gconv((char *)&null, (char *)&dnull, 1, ihdcvt(dtype), CVT_D);

        /* Allocate input buffer */
        if ((inbuf = alloc(dlen, xnum)) == NULL)
                gipserr1(GIPSNOCORE);
```

```
        /* Open the output image */
        if ((fq = ihwrit(ofile, 1)) == NULL)
                gipserr2(GIPSBADOPEN, ofile);
        if (fq == stdout) {
                signal(SIGPIPE, exit);
                ofile = "'stdout'";
        }
        /* Copy the image; set pixels outsize mapping to 'null' */
        for (n = y = 0; y < ynum; y++) {
                if (fread(inbuf, dlen, xnum, fp) != xnum)
                        gipserr2(GIPSBADREAD, ifile);
                for (x = 0; x < xnum; x++)
                        if (g_unmap(x, y, lonlat))
                                n++, bcopy(null, inbuf+x*dlen, dlen);
                if (fwrite(inbuf, dlen, xnum, fq) != xnum)
                        gipserr2(GIPSBADWRITE, ofile);
        }
        /* Cleanup */
        ihbcl(fp);
        ihbcl(fq);
        if (vflag)
                fprintf(stderr, "%s: %d pixels nulled0, ifile, n);
        if (Sflag)
                telltale(1);
        exit(0);
}
```

At the beginning of the program, a call to *saveargs* preserves the input argument list so that a subsequent call to *putargs* can insert a fresh *header* line into the output image header. Then the arguments are parsed by a call to *gipsflags*, which scans them against a *flagtable* structure in which the first column defines the argument character, the second is a code—"1" denotes a flag, "–>" a flag followed by a character string, etc. The third column tells *gipsflags* where to store the flag or argument value. If *gipsflags* detects an invalid argument, or if the program is invoked with a single "=" argument, it will print a usage note, which in this case will be:

Usage: *command* **[–D] [–S] [–v] [–m file] [–o file] [file]**

and then exit. For *command* it uses the value of *argv*[0] saved by the call to *saveargs.* Finally, if the **–S** flag has been specified, and therefore *Sflag* has a non-zero value, a call is made to *telltale* with a zero argument to initialize a resource-usage table that will be used by a second call to *telltale* at the very end of the program.

The return from *gipsflags* removes flags and options from the *argv*[] array, so what remains must be a "positional" argument, i.e. in this case, the name of the input image, which the program now opens with a call to *ihread*. If no image name had been specified in *argv*, the default *ifile* value would be "–", which *ihread* interprets to refer to the standard input stream, *stdin*. If *ihread* discovers an error reading the image header, it returns a NULL value, and the *gipserr* macro will print a suitable error message to *stderr*, and exit cleanly. Otherwise, the program tries to read any string or image header specified in the **–m** option, and, via a call to *ihmerg*, merge its arguments with those already saved in static memory.

After the call to *ihread* (possibly followed by one to *ihmerg*), all necessary header variables have been initialized, so the program now calls a series of subroutines to check that the image can be processed. The first, and most crucial, is *g_setup*, which first verifies that the *mapping* variable exists, and then creates a series of tables for subsequent use by the calls to *g_unmap*. The program then inspects the valus of *xnum*, *ynum*, *dtype*, and *dnull*. The first three are necessary, and if they are not found, the *gipserr1* macros prints an error message and halts execution. On the other hand, *dnull* is assumed to

be zero if not specified, but in any case, the value must then be translated into the binary pixel format specified by *dtype*, and this is performed by a call to *gconv*, where the variable *dnull* is converted from CVT_D format (i.e. *double*) to *null* in *ihdcvt(dtype)* format. The GIPS CVT_* flags are defined in "*gips.h*".

The next step is to allocate a buffer to contain the image rasters—the call to *alloc* returns a NULL value if there was insufficient memory available. Finally, the call to *ihwrit* opens an output stream, *fq*, and writes the header to it. Note the call to *signal* to ignore the SIGPIPE signal. While not strictly necessary, this prevents the C–shell from writing "Broken Pipe" messages when a pipe of GIPS filters is terminated abnormally, which would otherwise threaten to hide the "real" error message—the one that caused the pipe to terminate.

All is now ready to begin copying the pixels. The input rasters, from a file or a pipe, are read by calls to *fread* and written by calls to *fwrite*.[9] Any failing I/O is detected and *gipserr2* macros used to format error messages and terminate the program. Meanwhile, each pair of *x,y* pixel addresses is checked by the *g_unmap* routine, which translates them to longitude (*lonlat*[0]) and latitude (*lonlat*[1]), and returns a non-zero value if the pixel lies outside the mapping region, i.e. if *x,y* doesn't correspond to any physical latitude or longitude. If this is the case, the *dlen*–byte value *null* is copied in place of the existing pixel value.

When all rasters have been copied, the image streams are closed by *ihbcl*. This is a most important step since, if the input comes from an inter-process socket, the program must have a chance to wait until the socket is drained before exiting. Finally, if the program was invoked with the **–v** flag, a message is written to *stderr* and, if **–S** was specified, a second call is made to *telltale*, with non-zero argument, which will result in a resource-usage message being written to *stderr*.

BIBLIOGRAPHY

[1] Peter G. Ford, "*A GIPS Tutorial*", MIT Center for Space Research (1992).

[2] "*FITS—A Flexible Image Transport System*", a paper presented at the "International Workshop on Image Processing in Astronomy", held in June 1979 at the International Center for Theoretical Physics in Miramare, Trieste, Italy, and obtainable from *Dr. Eric W. Greisen, National Radio Astronomy Observatory, Edgemont Road, Charlottesville, VA 22901,*

[3] Peter G. Ford, "*The Comtal Image Processor*", MIT Department of Earth and Planetary Sciences (1984).

[4] Richardus, P. and Adler, R., "*Map Projections*", North Holland/American Elsevier, 1972.

[5] Cogley, J.G., "*Map Projections with Freely Variable Aspect*", EOS, **65** 34, p.481 (1984).

ACKNOWLEDGEMENTS

This work was supported under NASA contract NAS2-9473. The author wishes to thank Nafi Toksöz, and the staff of the Earth Resources Laboratory, particularly Steve Gildea and Al Taylor, for their assistance while Version 1 was being written. A special thank-you to Larry Soderblom and Eric Eliason of the U.S. Geological Survey, Flagstaff, and to Arthur Olson of the National Cancer Institute, for many useful discussions since that time.

[9] The special GIPS library routines *ihbin* and *ihbout* were used to read and write rasters in the example of Table 8b, but, provided the raster length ($xnum \times dlen$) is known, C programs can call *fread* and *fwrite* instead.

Gipstool† and Xgips† at a Glance

gipstool [**–W** *SunView options*] [**–KNc**] [**–C** *map*]
and [**–n** *colors*] [**–Iamo** *file*] [**–e** *n n n n n*]
xgips [**–xy** *size,org,off*]
[**–XY** *size,left,right*] [**–zZ** *zoom*] [*name*]

Options

All options beginning **–W** are passed directly to the window manager. See *sunview*(1).

–*depth*
forces the image to be stored in a 1-bit (**–1**), or 8-bit (**–8**) frame buffer.

–C *name*
override the "*colormap name*" of the image window (see **–a**).

–I *file*
name of a sub-command script file to be executed at startup time. (default is "~/*.gipstoolrc*").

–K detach the program from its parent process once the image has been read.

–N use a *dynamic* (private) colormap.

–a *file*
name of a file containing 256 red, green, and blue *colormap* intensities to use when displaying image pixels.

–c when the input is from a disk file, read only the *xsize*×*ysize* sub-image specified by the **–x** and **–y** options.

–e *n n n n n*
error-diffusion coefficients used with the **–Z0** option.

–m *file*
name of a General Image file whose header will be merged with that of the input image while reading the latter. Alternatively, *file* can be a list of header variables in the format "*name=value*", and those values will be merged with the input image header.

–n *colors*
number of colors (a power of 2) to use to display the image (default is 128 for *gipstool*, 64 for *xgips).*

–o *file*
all keystroke and mouse commands will be copied to *file*.

–r *buffers*
number of buffers to use when reading an image that was filtered through *ihgfmt*(1).

–x *xsize,xorg,xoff*
–y *xsize,xorg,xoff*
determine how much of the input image is to be saved in virtual memory, i.e. *xsize* pixels starting at *xorg*, and *ysize* pixels starting at *yorg* will be stored in virtual memory. Within this array, the display will begin at offset *xorg* and *yorg*, but the remainder of the array can be displayed via the pan commands.

–X *Xsize,Xleft,Xright*
–Y *Xsize,Xtop,Xbottom*
image area will occupy *Xsize* × *Ysize* screen pixels, with borders of *Xleft*, *Xright*, *Ytop*, and *Ybottom* screen pixels.

–z *xzoom*
–Z *yzoom*
x and *y* zoom factors. **–Z0** (monochrome images only) uses an *error diffusion* algorithm.

Pixel Values

Each pixel is first transformed according to the value of the *dscale* header variable:

s $pix \rightarrow \text{float}(pix)$

n $pix \rightarrow dfact \times pix + dzero$

l $pix \rightarrow \log_{10}(pix)$

p $pix \rightarrow dparm_0 + dparm_1 \times pix + dparm_2 \times pix^2 +$
$+ \ldots$ (*npoly*+1 terms)

f $pix \rightarrow n$, $f[n] \le pix < f[n+1]$, for f in the file *dfile*.

Then, unless *dscale=f*, the pixel is further transformed by

$pix \rightarrow 256 \times (pix - dmin)/(dmax - dmin)$,

where *dmax* and *dmin* default to 0 and 256, respectively, if not defined in the input image header. If the resulting pixel value is less than 0, it is set to zero; if more than 255, it is set to 255.

Sub-Command Keystrokes

^A pan to the left edge of the image.

^B pan to the left (see **P**).

^C terminate the program. You will be asked to confirm this action.

^D draw line from the image mark to the cursor location, and reset the image mark to this pixel.

^E pan to the right edge of the image.

^F pan to the right (see **P**).

^G run *gipsmongo*(1) in a separate window.

^H prompt for a keystroke and display information about the corresponding sub-command.

^I display image information.

^K remove all graphics created under the current graphic index.

^L re-display the image and all graphics that are currently turned "*on*".

^M remove all pop-up windows and error messages.

^N pan downward (see **P**).

^O reset the image mark to the cursor location; enter "*rubber band*" sub-image selection mode.

^P pan upwards (see **P**).

^Q toggle cursor cross-hairs (*gipstool* only).

^R scroll the list of *tagged* image pixels (see **{**, **T**, and **R**).

^S reset the image marker to the cursor location; enter "*rubber band*" line drawing mode.

^T pan to the top of the image.

^U pan to the bottom of the image.

^V display the coordinates and real-world location of the cursor (same as the MIDDLE mouse button).

^W display a color wedge insert.

^X make the cursor location the top left corner of the image display area.

^Y make the cursor location the bottom right corner of the image display area.

! invoke "/*bin*/*sh*" to execute a shell command.

. prompt for a pair of pixel addresses, then position the cursor on that pixel. Address formats are

n absolute (unzoomed) image pixel
±*n* image pixel relative to cursor
:*n* absolute screen pixel address
<*n* screen pixel relative to cursor
>*n* screen pixel relative to cursor
f realworld coordinate

/ prompt for the name, and optionally an **"="** sign and replacement value, for a GIPS header variable.

ignore all keyboard input until the next carriage return.

< execute a file containing sub-commands, one per line.

> begin or end echoing sub-commands to a disk file or pipe.

1 assign a sub-command to the LEFT mouse button.

A reset the text drawing angle, in degrees counter-clockwise from vertical.

B reset the background color: a triplet of RGB integers in the range 0–255.

C reset the graphic color value: an integer in the range 0–255 specifying the colormap value for subsequent graphics.

D hide the current graphic plane.

F reset the drawing font for all subsequent text.

G reset the current graphic index: an integer in the range 0–255.

H write the colormap and pixel histogram to the specified file or socket, in *General Image* format. The histogram is taken over the sub-image defined by the image mark and the current cursor location. It contains 256 rows and 7 columns:

- pixel index (0 through 255)
- realworld pixel value
- red colormap value
- green colormap value
- blue colormap value
- # pixels with this value
- # pixels <= this value

I select the rectangular sub-image defined by the image mark and the cursor location; write it to a disk file or socket.

J write the line of pixel values stretching from the image mark to the cursor location to the specified disk file or socket as an *n*×7 General Image whose rows represent the *n* individual pixels in the line and whose columns represent

- the pixel count,
- integer x-value
- real x-value
- integer y-value
- real y-value,
- integer pixel value
- real data value

K write the current colormap to a file or pipe.

L direct output from the **l** command to a disk file or pipe. Omitting the file name suspends this function.

M read a new colormap file.

N reset the numeric precision of all displays of floating-point numbers.

O reset the text origin code: an integer 0–8, specifying the relationship between the cursor location and the origin of graphic text strings:

1 2 3
4 0 5
6 7 8

P set the pan increment: the number of screen pixels that the display moves for **^B**, **^F**, **^N**, and **^P**.

R point cursor at tagged pixel location (see **T**).

S apply contrast-stretch (0–100%) to the colormap.

T prompt for a *tag* name by which to remember the cursor location (see **R**).

W select width of paint brush for **p** command.

Z set the zoom insert size: supply the following parameters for the MIDDLE mouse button action:

- number of screen pixels
- additional zoom factor
- colormap index of cross-hairs

a prompt for changes to *xaxis* and *yaxis* parameters, and draw both axes.

f set or clear the window frame title.

h convert an in-core GIPS header variable value to ASCII and write it into the current graphic.

l write the current pixel's coordinate and data values to the logging file (see **L**).

m draw grid lines on an image that has non-separable coordinates.

p paint pixels in overlay plane.

t write a text string into the current graphic.

u undo the most recent graphic operation.

v convert the current pixel coordinate (**x**, **y**, and/or **d**) to ASCII and write it into the current graphic.

w write overlay plane to disk file as GIPS image.

x update *xaxis* parameters and draw the *x*-axis.

y update *yaxis* parameters and draw the *y*-axis.

{ read a file of tag names and display it in a pop-up window.

} save the current tag name list in to a file.

Axis Drawing Parameters

The following variables are used in *xaxis* and *yaxis* header strings and when prompted by the **a**, **x**, and **y** commands.

mark= maximum number of scale labels.

tic= number of tick marks per scale label.

font= font for scale annotation.

place= where to write the axis; one or more characters:

l left of center, or
c centered, or
r right of center.

i inside image area, or
o outside image area.

t top of image, and/or
m within the image, and/or
b bottom of image

loc= scale value at which to write **m**iddle titles.

from=
to=
by= where the scale values are to be drawn; integer values imply pixel coordinates, floating-point notation implies realworld coordinates.

line= non-zero to connect scale marks.

form= the *printf*(3S) format with which to draw the scale annotation.

ticlen= length in screen pixels of the scale and tick marks.

scale= special scale-label formatting:

s simple fixed or float numbers
l ten to the power of ...
d degrees, minutes, and seconds
h hours, minutes, and seconds

NAME

gin – generate a dummy General Image

SYNOPSIS

gin [**–m** *file*] [**–s** *dscale*] [**–t** *dtype*] [**–xy** *num*]

DESCRIPTION

The *gin* command writes (to the standard output stream *stdout*) a dummy General Image file whose pixels are assigned positive integer values 0, 1, 2, ... This command is designed to be used to create a dummy image that, after passing through other GIPS filters, will be displayed at the user's console via the *git*(1) command, e.g.

gin | ihrot | git

The default header is as follows:

header = "Image Header"
version = *xxx*
dtype = b
dscale = s
xnum = 8
ynum = 8
FINIS =

The default image is therefore an 8×8 array of 1–byte pixels with decimal values from 0 through 63. This may be altered via *gin* options, as follows:

–m *name* specifies that the header from file *name* is to be merged with the default header; the resulting header defines the output General Image.

–s *char* alters the value of output header variable *dscale* to *char*.

–t *char* alters the value of output header variable *dtype*; e.g. "*gin -t f*" generates an image containing 4-byte floating-point pixels (*dtype=f*).

–x *num*
–y *num* alter the *x*– or *y*-dimension of the output image; e.g. "*gin –x 12 –y 16*" generates a General Image containing 16 rasters of 12 pixels each. The pixel values will increase from 0 through 191.

FILES

/usr/gips/tables/cprofile image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

git(1), ihc(1), ihread(3)

DIAGNOSTICS

When an error is detected, *gin* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- bad argument: *arg*

NAME

gipsmongo – use Mongo to display a General Image file

SYNOPSIS

gipsmongo [*Suntool options*] [**–S**] [**–I** *file*] [**–o** *device*] [name ...]

DESCRIPTION

Gipsmongo reads one or more General Image files from *name...*, and reformats them for display by the *mongo*(1) graphic utility. If any *name* is specified as "**–**", *gipsmongo* reads the General Image from the standard input stdin, instead. If the list of *names* is omitted entirely, *gipsmongo* waits for an input stream to be written to the *gipsmongo* inter-process socket.

OPTIONS

–S writes a list of UNIX usage statistics to the standard error stream *stderr* when *gipsmongo* terminates.

–I *file* specifies the name of a file containing *mongo* commands that will be executed immediately after the first General Image has been read.

–o *device*

specifies the type of output device. Currently allowed devices are as follows:

r640	Retrographics 640
vt125	DEC VT125
4014	Tektronix 4014
tek	Tektronix 4014
g270	Grinnell 270
hp2648	Hewlett Packard HP2648A
sun	SMI Graphic Tool Environment
versv	Versatec (vertical)
vers	Versatec (horizontal)
printv	Printronix (vertical)
print	Printronix (horizontal)
laserv	Laser Printer (vertical)
laser	Laser Printer (horizontal)

If *device* is not specified, *gipsmongo* uses the $TERM environment variable. Within *gipsmongo*, this may be changed by the **device**, **printer**, and **terminal** sub-commands.

GIPSMONGO SUBCOMMANDS

contour *#levels level1 level2 ... levelN*

generates a contour plot of the General Image, using *#levels* contouring levels (default 8). This may be followed by the realworld pixel values corresponding to these levels. If omitted, the levels will linearly subdivide the interval *dmin* to *dmax*.

header

writes the General Image header to the standard output stream *stdout*.

inform

writes a brief list of *gipstool* sub-commands to the standard output stream *stdout*.

mongo

invokes the interactive *mongo* command. The command-line prompt changes from ">" to "∗". Type **END** to return to *gipsmongo* command level.

next

terminates processing of the current General Image file. If *gipsmongo* was invoked with a list of file *names*, the next file is read. Otherwise, *gipsmongo* waits for another file to arrive at the inter-process socket.

plot3d *zenith azimuth scale origin*

generate a three-dimensional plot of the current General Image. The viewing geometry is specified

by *zenith* and *azimuth* (in degrees). *scale* specifies a Z-direction scaling factor. *Gipsmongo* always scales automatically, so *scale* is an additional multiplicative factor. *Origin* specifies an additional Z-direction origin offset which may be used to center the plot in the viewing area.

quit
closes the current General Image file and terminates *gipsmongo* immediately.

show
displays a list of current *gipsmongo* and *mongo* parameters.

! *command*
causes the string *command* to be executed by the default UNIX command shell.

In addition, *gipsmongo* intercepts the *mongo* subcommands **xlabel** and **ylabel**. If either is entered without an argument, *gipsmongo* supplies the values of the corresponding *xtitle* or *ytitle* header field of the current General Image, and passes the subcommand to *mongo*. using -50 for the logarithm of 0 or negative numbers.

MONGO SUBCOMMANDS

3dplot *alt az zmax*
3dplot makes a 3–d plot (grid of z heights over x,y) viewed from altitude *alt* and azimuth *az*, scaled so that z value *zmax* fills the vertical scale. *3dplot* must use data passed to *mongo* when it is called as a subroutine.

angle *d [x1 y1 x2 y2]*
angle will cause text from *label* to come out D degrees counter- clockwise from horizontal. It also causes points to be rotated counter-clockwise by D degrees. If there are four arguments present, they are taken to be user coordinate of two points and *angle* is set to the slope of the line.

axis *a1 a2 asmall abig ax1 ay1 ax2 ay2 ilabel iclock*
Makes an axis labelled from *a1* to *a2* stretching between device locations (*ax1*, *ay1*) and (*ax2*, *ay2*). If *abig* > 0 use that for spacing of large ticks. If *asmall* < 0 make a logarithmic axis, if *asmall* = 0, do the default. If *asmall* > 0 try to use that for the spacing of small ticks. *ilabel* parallel/perp/no labels, 00/10/20 for roman/plain/tiny font, 0/100/200/ *300* for default/0.0/dms/hms format. *iclock* = 1 for clockwise ticks on the axis, 0 for counter.

box *[m n]*
box puts axes around the plot region, labelling the lower and left. If arguments m and n are included (default 1 and 2) they are used as *ilabel* arguments for the lower and left axes.

buffer
Typing *buffer* will execute the contents of the command buffer, and *buffer* can be used as an argument for *list*, *write*, and *playback*.

connect
connect draws line segments connecting the coordinates read by *xcolumn* and *ycolumn*.

contour *n level1 ... leveln*
contour makes a contour plot with contours drawn a z values *level1* to *leveln*. *contour* must use data passed to *mongo* when it is called as a subroutine.

cursor *[x y]*
With no arguments *cursor* will turn on a terminal's cursor, wait for it to be positioned and then read the coordinates. The user coordinates are stored in the symbolic variables *cx* and *cy*. When *cursor* has two arguments it positions the cursor at that user location, but does not turn on the cursor. A cursor command is stored in the command buffer as "cursor" followed by the resulting cursor coordinates, so that the command buffer can be played back for any device, regardless of whether it has a cursor. *cursor* sets the current graphics pointer to the cursor position.

data *fspec*
data opens a data file for reading data. The file is assumed to have numerical data in columns separated by spaces, tabs or commas. Any column can be read as x or y coordinates by *x* and

ycolumn. In the case that *mongo* has been called as a subroutine, *data* with no arguments specifies that data be read from that array passed to *mongo*.

define *xxx*

define xxx creates an entry for a macro xxx, and subsequent commands until *end* are the body of the macro. The prompt changes to D* to indicate the mode. A macro is invoked by its name, and its name can be used as an argument to *list*, *write* and *playback*. The tokens &1, &2, ..., &9 can be used in the body of the macro, and when a macro is invoked its arguments will be substituted for these tokens.

delete *[n1 n2]*

delete removes commands from the command buffer (numbered according to *list*). If no arguments, the last command is deleted, if one, that command is removed, and if two arguments all commands from one argument to the other are deleted.

device *n*

N	Device	N	Device
1	Retrographics 640	–1	Versatec (vertical)
2	DEC VT125	–2	Versatec (horizontal)
3	Tektronix 4010	–3	Printronix (vertical)
4	Grinnell 270	–4	Printronix (horizontal)
5	HP2648A	–5	Laser (vertical)
6	SUN windows	–6	Laser (horizontal)

dot *dot* draws a point at the current location (set by *relocate*, *draw*, etc) in the style determined by *ptype*. The point's size and rotation are governed by *expand* and *angle*.

draw *x y*

draw draws a line to (x,y) from the current location (set by *relocate*, *draw*, etc), and makes (x,y) the current location.

ecolumn *n*

ecolumn reads column N from the data file as the magnitudes of the error bar displacements from the corresponding (x,y) coordinates. Note that *x* and *ylogarithm* do not adjust the error values to logarithmic coordinates.

end In define mode *end* finishes the macro definition and returns to execute mode. In insert mode *end* returns to execute mode. In execute mode, *end* exits from the program altogether (as does ^Z).

erase

erase erases the graphics screen if in terminal mode, or it reinitializes a printer plot (sets the vector count to 0).

errorbar *k*

errorbar is analogous to *points*; it draws error bars on all (x,y) points (read by *x* and *ycolumn*) of length read by *ecolumn*. The argument is 1 to put the bar along the +x direction, 2 for +y, 3 for –x, and 4 for –y. Use *expand* to govern the size of the caps.

expand *e*

expand expands all characters and points, its default is 1.0. Note: with default expansion, characters are sent to the graphics terminal without font translation; in order to get characters of size 1 with font translation, use *expand* 1.0001.

format *[...]*

With no argument, *format* uses a list directed read from data files. An argument is used as a *fortran* format statement (parentheses must be included) for reading from data files. Note that all data format entries must be floating point and that "Gn.0" will read any floating data item in a field of length n.

grid *[n]*

Without an argument, *grid* makes a grid of dotted lines at all major tick locations within a *box*.

With an argument of 0 or 1, *grid* makes a grid at major or all ticks of the current line type.

halftone *[z1 z2 [contrast]]*

halftone makes a grayscale picture of data, with data values z1 to z2 drawn at gray levels ranging between the background and "set" levels of the output device. If the argument "contrast" is not present, *halftone* uses a linear interpolation between z1 and z2. If contrast is present and equal to 0, *halftone* makes an enhanced mapping of data values to gray scale levels so that there are an equal number of pixels at each gray level. If contrast is positive (negative), this mapping is altered to enhance the number of background ("set") pixels; values of -5 to +5 are useful. *halftone* will make the picture fill the current graphics area, and requires data passed to *mongo* via the subroutine.

hardcopy

After a *printer* or *erase* command all graphical output is stored. *hardcopy* causes the stored vectors to be plotted on the printer and reinitializes the vector count.

help *xxx*

Without an argument, *help* prints a one line description of all builtin commands. With an the name of a command as an argument, *help* prints more information on that command.

histogram *[n [f]]*

histogram connects the coordinates read by *xcolumn* and *ycolumn* as a histogram. If the argument n is present, the area between the histogram and device y coordinate n is hatched. If n < 0 the hatching is at –45 degrees and y coordinate –n is used. If an argument f is present the area between histograms in *y* and Ecolumns is hatched.

id *id* puts the name of the current input file, data file, date and time outside the upper right hand corner of the plot region.

input *fspec*

input reads plot commands from a file and treats them exactly as if they were typed: commands are executed, or can be inserted or used in macro definitions.

insert *[n]*

insert enters insert mode, indicated by a I* prompt, and subsequent commands are inserted without execution into the command buffer until an *end* command is encountered. Without an argument, *insert* inserts at the end of the buffer; with an argument, *insert* starts inserting just before that location.

label ...

label ... writes the string ... (which starts one space after *label* and continues to the last non-space character) at the current location (set by *relocate*, etc). The string's size and angle are determined by *expand* and *angle*. The character "\" is an escape character and causes the following action.

```
**********************************
* \\x – set mode x              *
* \x  – set mode x for next char  *
* \r – roman font               *
* \p – plain font               *
* \g – greek font               *
* \s – script font              *
* \t – tiny font                *
* \i – toggle italics           *
* \o – old english font                *
* \u – superscript              *
* \d – subscript                *
* \b – backup character width          *
* \e – end string               *
* \0 – \9 – user variables \0 – \9  *
```

****

limits *[x1 x2 y1 y2]*

limits sets the coordinates of the plot region. All coordinates from *x* and *ycolumn*, *relocate*, *draw*, etc, are referred to these limits. If *limits* has no arguments, the data from the most recent *x* and *ycolumn* are used to set the limits.

lines *l1 l2*

lines sets the range of lines read from the data file by *x*, *y*, and *ecolumn*. It is useful to avoid non-data lines.

list *[xxx]*

list lists the commands of the named macro. The macro name "all" or no argument lists the command buffer. The numbers assigned to commands by *list* are those used by *delete* and *insert*.

location *gx1 gx2 gy1 gy2*

The plot region is a rectangle within the coordinates allowed by the plotting device. Vectors and points are truncated at the bounds of the plot region. *location* specifies (in device coordinates) where the plot region is located. *location* can be used to make an arbitrary size and shape plot.

ltype *n*

All lines except for points and characters are drawn with line type n, where n refers to lines of the following type:

0 = solid
1 = dot
2 = short dash
3 = long dash
4 = dot – short dash
5 = dot – long dash
6 = short dash – long dash

lweight *n*

All vectors plotted on a hardcopy printer are made with a weight of n.

page *n*

page assumes *printer* output onto fanfold paper. It sets the physical limits and location so that vectors are offset onto page n after the first one.

pcolumn *n*

pcolumn is analogous to *ptype*, except that when *points* is executed each (x,y) point is plotted with the corresponding entry in the point column, instead of the same point type for all points. The format is the same as *ptype*, except that the numbers are contracted together. If the entry has a fractional part, it is treated as an expansion factor (fractional part less than .01 gives default expansion). For example, an entry of 103.5 in a point column is the same as *ptype* 10 3, *expand* .5.

physical *lx1 lx2 ly1 ly2*

The plot region is a rectangle within the coordinates allowed by the plotting device. *physical* specifies (in device coordinates) the absolute limits of where graphics output can go. Anything outside of these limits is truncated.

playback *[xxx]*

playback xxx executes the macro xxx without storing the commands in the command buffer. With an argument of "all" or no argument, *playback* replays all commands in the command buffer.

points

points makes points of the current style (*ptype*), size (*expand*), and rotation (*angle*) at the coordinates read by *xcolumn* and *ycolumn*.

printer *[n [device]]*

printer directs graphical output to a graphics printer and sets default values for the plot region location and device physical limits. An optional argument is used to specify a particular device.

n	Device
1	Versatec (portrait)
2	Versatec (landscape)
3	Printronix (portrait)
4	Printronix (landscape)
5	Laser printer (portrait)
6	Laser printer (landscape)

A second optional argument specifies the device name.

ptype *n s*

ptype n s causes points to be drawn as n sided polygons of a style s, where s refers to:

0 = open
1 = skeletal (center connected to vertices)
2 = starred
3 = solid

ptype will also accept one argument which is a composite = 10∗n + s.

putlabel *l ...*

putlabel writes a label at the current location with rotation and size specified by *angle* and *expand* (exactly like *label*). The label is oriented with respect to the current location according to the argument L which can be 1 – 9 for:

	right	center	left justified
label above	7	8	9
centered	4	5	6
below	1	2	3

relocate *x y*

relocate sets the current location to (x,y) without drawing a line.

reset

reset does something which I have not yet decided.

set *v x [op y]*

set sets the symbolic variable V to a value *x* if only two arguments are present. If four arguments exist V is set to *x* (op) *y* where op is either + – ∗ / % \ > or <. V can be a user variable \0 – \9 or a symbolic variable such as gx1. These variables can then be used as arguments to any other command.

show

show lists the values of some the variables, including current location and plot region limits in user and device coordinates, the value of the expansion and angle variables, the line type and weight, and the physical limits.

terminal *[n [device]]*

terminal directs graphical output to the graphics terminal and sets default values for the plot region location and device physical limits. An optional argument is used to specify a particular device.

n	Device
1	Retrographics 640
2	DEC VT125
3	Tektronix 4010
4	Grinnell 270
5	HP 2648A
6	Sun Windows

A second optional argument specifies the device name, eg, *tta0*:.

ticksize *smallx bigx smally bigy*

ticksize determines tick intervals for *box*. *smallx* refers to the interval between small tick marks on the x axis, *bigx* refers to the interval between large ticks, etc. If *big* is 0, the axis routine will

supply its own intervals according to the label limits. If *small* < 0, the axis will have logarithmic tick spacing with large ticks at each decade and small ones at each integer. (Despite the axis labels, the limits are still logarithms, e.g. –2 and 2 refers to limits of .01 and 100.)

vfield *[i [unity]]- draws a vector field as a sequence of arrows*

vfield draws a vector field as a number of arrows. The tails of the arrows lie at points (x,y) read by *x* and *ycolumn*, the lengths of the arrows r are read by *pcolumn* and the direction of the arrow (ccw from right in degrees) is read by *ecolumn*. *expand* is used to vary the length of the arrows. Note that the use of *pcolumn* in this context requires *ptype* to be re-executed before plotting points. *vfield* 1 uses cartesian coordinates for vectors; (dx,dy) = (p,ecolumn). *vfield* i *unity* scales the length of the vectors so that a vector of magnitude *unity* is drawn with a length of 1 on the x-axis.

window *nx ny k*

window makes the current plot location window k, where there are nx windows across and ny windows up and down, and the windows are counted across from the lower left. k = 1 specifies the first window, and *window* 1 1 1 resets the plot location to the entire plot area; be sure to do this before changing plot location or device!

write *xxx fspec*

write the commands making up a macro to a file. If the macro name given is "all" all macros currently defined and the entire command buffer are written to the file.

xcolumn *n*

xcolumn reads column N from the data file as the values of the x coordinates of the data points.

xlabel ...

xlabel writes a label centered under the x axis made by *box*.

xlogarithm *[n]*

xlogarithm takes the logarithm of x values from *xcolumn*, using -50 for the logarithm of 0 or negative numbers. Use of an argument N takes the logarithm of *x* / *y* /P/Ecolumn for N=1/2/3/4.

ycolumn *n*

ycolumn reads column N from the data file as the values of the y coordinates of the data points.

ylabel ...

ylabel writes a label centered left of the y axis made by *box*.

ylogarithm *n*

ylogarithm takes the logarithm of y values from *ycolumn*, using -50 for the logarithm of 0 or negative numbers.

FILES

/usr/local/tables/cprofile	image header definition table
/tmp/gips_socket	interprocess communications socket

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

mongo(1), suntools(1), mongo(3)

For a complete description of the *mongo* interactive computer graphics language, see "*Mongo*" by John Tonry, MIT Center for Space Research, 1985.

ACKNOWLEDGEMENTS

Much thanks to John Tonry for his assistance in merging his powerful graphics package with GIPS and SunTools.

DIAGNOSTICS

When an error is detected in accessing a General Image file, *gipsmongo* writes a message to the standard error stream *stderr*, and halts with system code 1. Messages generated by GIPS library routines

are described in *ihread*(3i). Other possible messages are

- unsupported output device: *name*

BUGS

Many. This is only a test version. In particular, *gipsmongo* reading from a socket written by *gipstool* can hang so badly that a reboot is necessary!

In the ‘‘things to do’’ category, the *contour* command should be completely rewritten to incorporate a contour smoothing algorithm with automatic level-indicator insertion.

NAME

gipstool – display a GIPS image in a Sun window

SYNOPSIS

gipstool [*Suntool options*] [*– depth*] [**–KNc**] [**–C** *map*] [**–Iamo** *file*] [**–n** *colors*] [**–r** *buffers*] [**–xy** *size,org,off*] [**–XY** *size,left,right*] [**–zZ** *zoom*] [name]

DESCRIPTION

Gipstool reads a general image file from *name*, and displays the image in a Sun window. If *name* is omitted, it reads the standard input stdin, instead. *Gipstool* then attempts to execute commands from the startup file "~/*.gipstoolrc*", and, unless an error occurs, becomes an interactive image editor, accepting input from the Sun keyboard and mouse.

There are three principal ways of using *gipstool* to display an image. By default, the entire image is read into memory. If the image is too large to fit within the window area, the arrow keys may be used to roam around. Alternatively, if *gipstool* is invoked with the **–c** option, enough of the image is read to display within the window area. If the image is larger than this, the arrow keys may be used to roam, but this will take longer since fresh parts of the image must be read from disk. Finally, a "cellular" image, one that has been filtered through the *ihgfmt* command, will be treated as if the **–c** option had been used, but, because it is easy to locate sub-units of cellular images, the image can easily be roamed by the arrow keys.

Once the image has been loaded and the startup file executed, the cursor changes from an hour-glass to an arrow, signifying the start of interactive mode–each *gipstool* command is executed by entering a single character or clicking a mouse button. Depressing the RIGHT-hand mouse button will display a menu of all possible commands. Help about any command key is available by pressing and releasing **^H** (CTRL–H or backspace), followed by the key in question. The **^C** command (the CTRL and C keys held down simultaneously) causes *gipstool* to terminate, while **^Z** suspends the program for later resumption.

The remaining keystroke commands are grouped into three broad categories:

Control Keys:

(**^A** through **^Z**) are reserved for simple image or graphic operations. Control keys are executed immediately–they never result in prompting for further input.

Uppercase Keys:

are used to supply values to internal *gipstool* variables. They cause a message box and reply area to appear in the top left corner of the display. Subsequent input characters, terminated by a carriage return, will be assigned to the particular internal variable. During text entry, the **DEL** key erases the last character typed, and the **^H** (backspace) key displays information about the command.

Lowercase Keys:

execute ‘‘higher level’’ *gipstool* functions, some of which may also result in further prompts for text entry.

The startup file "~/*.gipstoolrc*" contains a series of *gipstool* commands, one per line. The command character itself must begin in column 1. If the command would prompt the user to enter additional text, the latter should follow the command character on the same line. For convenience, commands that consist of control characters may be represented as two characters, a carat followed by an upper or lower case letter. (This convention is also adopted by the > command-logging command.)

OPTIONS

All options of *gipstool* that begin with **–W** are passed directly to the SunView manager. See the manual entry under *suntools*(1) for details.

– *depth* where *depth* can have the value **1** or **8**, selects the depth of frame buffer to use on systems with a choice, e.g. Sun/3–60C or 3/110C. In particular, you *must* use the **–8** flag to display color images on these models.

–C *name*

only has meaning for 8-bit (color) displays. It specifies the symbolic "*colormap name*" to be given to the *gipstool* window. This is **not** the same entity as the colormap *file* name supplied by the **–a** option–if two copies of *gipstool* are running simultaneously on the same Sun display and they share the same *colormap* name, any change made to one window's color lookup table will **automatically** be applied to the other. If you don't supply a **–C** option, the symbolic colormap name will be taken from the name of the colormap file supplied by the **–a** option. The default colormap name is therefore *bw*.

–I specifies the name of a command file to be read when *gipstool* starts up, instead of the default file named "~/*.gipstoolrc*".

–K Executes a *fork*(2) system call immediately after startup. *Gipstool* will execute in the background. If the image is to be read from the standard input, control will not return to the command or shell that invoked *gipstool* until the image has been displayed.

–N only has meaning for 8-bit (color) displays. It tells *gipstool* to use a full 256-entry color table to display the image. It is equivalent to specifying **–n 256**.

–a *file* specifies the name of a file containing the red, green, and blue output intensities to use when displaying image pixels. *gipstool* searches for this file in the current directory, and then in "*/usr/local/lib/cmap/*∗*.cmap*". The default is "*bw.cmap*" – a linear grey-scale mapping.

–c may be specified when the input image is a disk file. *gipstool* will keep sufficient image pixel values in core to repair window damage, but subsequent image re-sizing or panning will cause the relevant portions of the input file to be re-read. For images that are smaller than the amount of real memory available to *gipstool*, this mode of operation is less efficient than keeping the entire image in storage. It is, of course, necessary to use this flag when working with images that are too large to fit in virtual memory, and it is set automatically when the input image header variable *dtype* has the value "*G*", indicating that the image is stored in the special *cellular* format created by passing an image through the *ihgfmt* filter.

–m *file* specifies the name of a General Image file whose header will be merged with that of the input image before any window sizing or pixel conversion is performed. Alternatively, can be a list of header variables of the form "*name=value*", and those values will be replaced in the input image header.

–n *colors*

only has meaning for 8-bit (color) displays. It tells *gipstool* to use *n* entries from the colormap table specified by the **–a** option. *n* must be a power of 2 in the range from 2 to 256. When 256 colors are requested, all table entries are used, but these colors will only be displayed when the Sun cursor is within the current *gipstool* window. If 128 colors are requested, they are taken from lines 2, 4, 6, etc. of the colormap file specified by the **–a** option; for 64 colors, from lines 4, 8, 12, etc.

–o *file* specifies that all interactive *gipstool* keystroke and mouse commands will be copied to *file*. This may be changed within *gipstool* via the > function, described below.

–r *buffers*

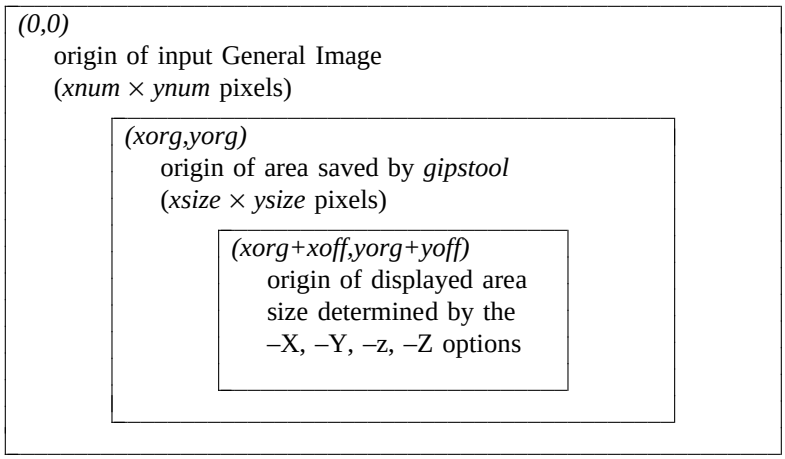
tells *gipstool* to allocate a number of buffers (default 32) when reading a *cellular* image of the type generated by *ihgfmt* . See the discussion of the *–c* flag, above.

–x *xsize,xorg,xoff*
–y *xsize,xorg,xoff*

determine how much of the input image is to be saved in virtual memory, i.e. *xsize* pixels starting at *x*–pixel address *xoff* will be reformatted, and similarly in the *y*–direction. *xsize* and *ysize* default to the entire image area (the input header variables *xnum* and *ynum*). The top left corner of the initial display will be offset by an additional *xoff* and *yoff* pixels, although the entire saved *xsize* × *ysize* image can be displayed via *gipstool*'s pan commands. If *xorg* or *yorg* are omitted, they default to the values of the header variables *xorg* and *yorg*,

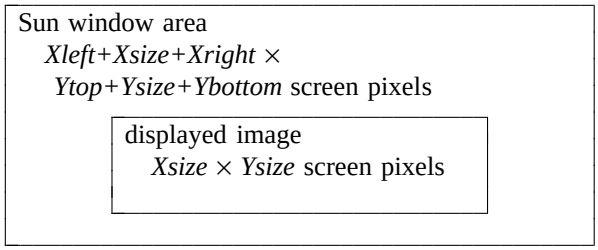
respectively, or to 0 if these too are undefined. *xoff* and *yoff* also default to 0. (If the **–c** flag is specified, *xsize* and *ysize* are disregarded – the full *xnum* × *ynum* image may be displayed via the pan commands, but the initial display is still offset by *xorg+xoff* pixels in the *x*–direction and *yorg+yoff* pixels in the *y*–direction, as measured from the top-left origin).

The following figure shows the three image areas – the area of the input general image, the (possibly smaller) area of image stored within *gipstool*, and the (maybe even smaller) area that can be displayed at any moment within the Sun window. All *(x,y)* values are in input image pixel coordinates:



–X *Xsize,Xleft,Xright*
–Y *Xsize,Xtop,Xbottom*

determine the size of Sunwindow to be used to display the image. The image area will occupy *Xsize* × *Ysize* screen pixels, with additional borders of *Xleft*, *Xright*, *Ytop*, and *Ybottom* screen pixels. By default, *gipstool* uses no borders and sizes the Sunwindow to fit the image exactly. If the input image is too large, the window will exactly fill the screen, leaving the borders as specified. The window and image area are related as follows:



–z *xzoom*
–Z *yzoom*

specify that each image pixel will occupy *xzoom* × *yzoom* screen pixels. **–z** sets a common zoom factor in both *x* and *y* directions (the default is, of course, 1), and **–Z** specifies a separate *y*–direction zoom factor. Because of this zoom feature, you must note carefully the difference between *image* pixels (described by the **–x** and **–y** options), and *screen* pixels (described by **–X** and **–Y**), which may possibly be zoomed.

On 8-bit displays, zooming is achieved by pixel replication, and the zoom factors may be any positive integers. On 1-bit (black-and-white) displays, zooming permits differing pixel values to be displayed as contrasting patterns. Such patterns are currently restricted to zoom factors of 1, 2, 3, or 4, and must be the same in both *x* and *y* directions.

On 1-bit displays only, the special value **–Z0** used in conjunction with the default **–z1** tells *gipstool* to display a one-bit-per-pixel image using an "*error diffusion*" algorithm. This technique is particularly effective in displaying quite subtle variations in areas of low contrast, but it prevents interactive *gipstool* commands from reporting accurate pixel values.

PIXEL VALUES

Gipstool performs a variety of transformations on the input General Image pixels before displaying them on the screen.

If the image has been passed through *ihfft*(1i) so that *dtype=c*, each complex pixel will be replaced by its absolute value before further conversion.

Each pixel is then transformed according to the value of the *dscale* header variable:

s $pix \rightarrow \text{float}(pix)$

n $pix \rightarrow dfact \times pix + dzero$

l $pix \rightarrow \log_{10}(pix)$

p $pix \rightarrow dparm_0 + dparm_1 \times pix + dparm_2 \times pix^2 + \ldots$ (*npoly*+1 terms)

f $pix \rightarrow n$, $f[n] \le pix < f[n+1]$, for *f* in the file *dfile*.

Then, unless *dscale=f*, the pixel is further transformed by

$$pix \rightarrow 256 \times (pix - dmin)/(dmax - dmin),$$

where *dmax* and *dmin* default to 0 and 256, respectively, if not defined in the input image header.

In many cases, the above steps are purely symbolic – *gipstool* optimizes the conversions whenever possible, and constructs lookup tables. The result, in any case, is that each pixel is transformed to an integer in the range 0–255. Values below 0 are set to zero, those above 255 are set to 255. From here on, 8-bit displays and 1-bit displays take different paths.

COLOR DISPLAYS

The mapping between pixel value and displayed color is determined by the contents of the colormap file specified by the **–a** option. A colormap file should contain exactly 256 lines, excluding comment lines beginning with the ''#'' character. Each line must contain a triplet of integer red, green, and blue intensity values in the range 0 through 255, separated by tabs or spaces. Additional comments may follow the blue value. The 256 RGB values determine the output color and intensity to be given to pixels with byte values from 0 through 255, respectively.

The actual number of colors to be displayed is determined by the value supplied to the **–n** option of *gipstool*. The default value is 128, permitting GIPS images to share the Sun display with other colored graphics and windows without forcing the Suntools manager to switch colormap tables as the different windows are selected. The 128 values are taken from lines 0, 2, 4, etc. of the colormap file, (ignoring comment lines). Therefore, pairs of pixel values 0 and 1, 2 and 3, and so forth will share a common output RGB value.

MONOCHROME DISPLAYS

Any **–a** option will be ignored. Each pixel is reduced to a number in the range 0 through 255, as outlined above. The pixel is then displayed according to the value of the **–z** option, which may be given the value 1 (the default), 2, 3, or 4. This value determines the size of display area (in screen pixels) to correspond to each input image pixel. For **–z1**, a single output pixel is used, and will be set "*on*" (i.e. black) if the input pixel byte value exceeds 127, and will be "*off*" (white) otherwise. For **–z2**, a 2×2 output pixel is used, For **–z3**, a 3×3 output pixel is used, and so on. The input pixel byte value determines how many of the *n*×*n* output pixels will be set "*on*", as follows:

–z	Input Value	"on" Bits	Input Value	"on" Bits	Input Value	"on" Bits
1	0–127	0	128–255	1		
2	0–63	0	64–127	1	128–191	2
	192–254	3	255	4		
3	0–31	0	32–63	1	64–95	2
	96–127	3	128–159	4	160–191	5
	192–223	6	224–254	7	255	8
4	0–15	0	16–31	1	32–47	2
	48–63	3	64–79	4	80–95	5
	96–111	6	112–127	7	128–143	8
	144–159	9	160–175	10	176–191	11
	192–207	12	208–223	13	224–239	14
	240–254	15	255	16		

Alternatively, if the **–Z0** option is chosen, *gipstool* uses an "*error diffusion*" algorithm to assign on–off values to the one-bit-per-pixel image. The simplest version of this algorithm may be described as follows:

- Reduce the image to an array of elements A_{xy}, each in the range from 0 to 1.
- Inspect the A_{xy} values sequentially, raster by raster. If $A_{xy} \geq 0.5$, set the output pixel *on* (black), otherwise set it *off* (white).
- Compute the error e incurred by this assignment, i.e. $e=A_{xy}$ when $A_{xy} < 0.5$, and $e=1–A_{xy}$ when $A_{xy} \geq 0.5$.
- Distribute this error evenly to those surrounding pixels that have not yet been displayed, i.e. add $0.25e$ to elements $A_{x+1,y}$, $A_{x–1,y+1}$, $A_{x,y+1}$, and $A_{x+1,y+1}$.

THE MOUSE

While *gipstool* is loading a general image and executing the commands in the "*gipstoolrc*" file, the Sun cursor is displayed as an hour glass. If the image is too large to display in the window, the proportion of the total image that is already stored will be indicated by a flashing line across the image. Once the line reaches the bottom of the window, input will be complete, and the cursor will change from an hour glass to an arrow. During interactive operation, *gipstool* uses the mouse and its arrow cursor for three distinct purposes:

To issue a command:

The LEFT mouse button may be assigned to a particular *gipstool* command by means of the **1** (one) command. From then on, clicking the left button is equivalent to typing the key.

To determine the coordinates of a pixel:

Clicking or dragging the MIDDLE mouse button within the image area causes *gipstool* to display the coordinates and data value of the pixel that the cursor is pointing to. It also causes an insert to appear in which the area surrounding the cursor is displayed in zoomed (magnified) form.

To select a line or area:

Several commands, e.g. **^O**, **^S**, put *gipstool* into "*rubber band*" mode–the cursor changes into a small cross, waiting for the user to hold the LEFT mouse button down and drag the cursor to the required point.

To select commands from a menu:

When the RIGHT mouse button is depressed, a multi-level menu is displayed. All *gipstool* commands may be issued in this manner. The descriptions in the menus are laconic, so for more

details, hit the **^H** (backspace) key, followed by the key that you want to find out about. An informative message will appear on the screen. This feature will also work whenever *gipstool* is prompting for keyboard text. Hitting **^H** at this moment will generate a message describing the input that *gipstool* expects.

INTERACTIVE COMMAND KEYS

^A **Pan to left edge:** The **^A** key causes the display to pan to the extreme left edge of the image. If this is already visible, the key has no effect.

^B **Pan image left:** The **^B** (back) key causes the current image to move to the left in its display window by an amount that may be set by the **P** command. The default is to pan by half the current window size. If the left edge of the image is already visible, the key has no effect. This command may also be executed by the left-arrow key on the Sun keyboard.

^C **Cancel Gipstool:** The **^C** key terminates *gipstool* . You will be asked to confirm this action.

^D **Draw straight line:** The **^D** key draws a line from the image mark to the current cursor location. The image mark is reset to the current cursor location.

^E **Pan to right edge:** The **^E** (end) key causes the display to pan to the extreme right edge of the image. If this is already visible, the key has no effect.

^F **Pan image right:** The **^F** (forward) key causes the current image to pan to the right in its display window by an amount that may be set by the **P** command. The default is to pan by half the current window size. If the right edge of the image is already visible, the key has no effect. This command may also be executed by the right-arrow key on the Sun keyboard.

^G **Run mongo in a separate window:** The **^G** key starts the Mongo graphics program in a new window. Mongo will wait for you to send it a histogram or image via the **H**, **J**, or **W** subcommands.

^H **Get subcommand help:** The **^H** key generates help messages. When typed during a prompt for text entry, you may cancel the current command by pressing the LEFT mouse button, or continue to enter text as requested. When typed from *gipstool* command level, **^H** prompts you to enter a key-stroke, after which it displays a description of that command.

^I **Display image info:** The **^I** key displays a pop-up panel of information about the state of *gipstool*. Most of the items can be changed by putting the cursor in the oval to the left of the item and clicking the LEFT mouse button.

Frame title	the text that appears in the top strip of the frame.
Program version	the current version of *gipstool*.
Original image	the name of the input image file.
Original size	the size recorded in the header of the input image.
In-core size	the size of the pixel array that was actually read from the input image.
In-core origin	the location of the top left in-core pixel relative to the top left of the input image.
Display size	the size of the rectangle that is currently being displayed on the screen.
Display origin	the location of the top left displayed pixel relative to the top left of the in-core image array.
Display margins	the width in pixels of the right hand, and bottom margins, respectively.
Image mark	The pixel coordinates within the in-core image array, of the pixel marker, which may be reset by the **^S** command.
Colormap name	the logical colormap name specified by the **–C** option when *gipstool* was invoked.
Colormap file	the name of the current colormap definition file.

Color stretch	the percentage of adaptive stretch applied to the current colormap by the `S´ subcommand.
Linear stretch	the limits of realworld pixel values that are to be mapped to the current colormap range by the `s´ subcommand.
Graphic index	the logical index number under which subsequent graphics will be stored.
Graphic color	the color value assigned to the current graphic index.
Graphic angle	the angle, clockwise from vertical, at which subsequent text will be drawn.
Indices 'on'	the indices of the graphics that are currently displayed.
Indices 'off'	the indices of the graphics that are not currently being displayed.
Roman font	the name of the default text drawing font.
Italic font	the name of the font that will be used to draw *italic* text.
Bold font	the name of the font that will be used to draw **boldface** text.
Symbol font	the name of the font that will be used to draw special symbols.
Text orientation	the current code, set by the **O** command, that determines the relationship between text to be entered into the image and the current cursor location.
Command log file	the name of the file into which is written a log of all *gipstool* commands executed during the current session.
Pixel log file	the name of the file into which the **l** command will write its results.
Tag file	the name of the file that contains tag information about features in the current image.
Left button	the name of the command currently assigned to the LEFT mouse button.
Middle button	the name of the command currently assigned to the MIDDLE mouse button.
Right button	the name of the command currently assigned to the RIGHT mouse button.

^K **Clear current graphic:** The **^K** key removes all graphics that were created under the current graphic index, including the paint overlay used by the **p** command. The image will be re-drawn without those graphics, which will be discarded. If you merely want to hide those graphics, with the possibility of restoring them later, use the **D** command. Note that the paint overlay cannot be hidden.

^L **Repaint image:** The **^L** key re-displays the image and all graphics that are currently turned "*on*". This is often necessary after temporary prompts and inserts have clobbered portions of your graphics.

^M **Clear all screen inserts:** The **^M** (return) key removes all pop-ups and error messages from the image display area. It is also used to terminate all requests for data entry.

^N **Pan image down:** The **^N** (next) key causes the current image to pan downward in its display window by an amount that may be set by the **P** command. The default is to pan by half the current window size. If the bottom of the image is already visible, the key has no effect. This command may also be executed by the down-arrow key on the Sun keyboard.

^O **Set image marker and draw box:** the image marker is reset to the pixel pointed to by the cursor. *Gipstool* enters "*rubber band*" mode, drawing a temporary box with the image marker at one corner and the current cursor position at the other. Pressing any other key removes the box, but the image marker location is preserved.

^P **Pan image up:** The **^P** (previous) key causes the current image to pan up in its display window by an amount that may be set by the **P** command. The default is to pan by half the current window size. If the top of the image is already visible, the key has no effect. This command may also be executed by the up-arrow key on the Sun keyboard.

^Q **Toggle cross-hairs:** Hitting the **^Q** key alternately displays and removes a pair of lines that more clearly mark the vertical and horizontal location of the cursor.

^R **Scroll image tags:** the **^R** key displays the list of currently *tagged* image pixels, i.e. those cursor locations that have been marked by the **T** command or read in from an external tag file by the { command. Use the **R** command to restore the cursor to one of these addresses.

^S **Set image marker and draw line:** the image marker is reset to the pixel pointed to by the cursor. *Gipstool* enters "*rubber band*" mode, drawing a temporary line from the image marker to the current cursor position. Pressing any other key removes the line, but the image marker location is preserved.

^T **Pan to top edge:** The **^T** (top) key causes the display to pan to the top of the image. If already visible, the key has no effect.

^U **Pan to bottom edge:** The **^U** (under) key causes the display to pan to the bottom of the image. If already visible, the key has no effect.

^V **Show pixel inserts:** The **^V** The **^V** key displays a zoomed image of the pixels surrounding the current cursor location. The size and zoom factor may be reset by the **Z** command. The display will appear at the bottom of the screen and will be continuously updated as the cursor is moved. While holding the SHIFT button down, the arrow keys will move the cursor by one image pixel; if the insert is too large to fit below the image area, it will overlay the image itself.

^W **Display color wedge:** A color wedge is displayed. Put the cursor within the wedge and press the right mouse button to display a menu allowing you to select the direction of increasing grey-scale values. Choose "*done*" from the "*Frame =>*" menu to remove the wedge.

^X **Set top left of image:** Hitting the **^X** key causes *gipstool* to resize the image display area: the current cursor location will become the top left corner of the new display. The bottom right corner may similarly be redefined by the **^Y** key.

^Y **Set bottom right of image:** Hitting the **^Y** key causes *gipstool* to resize the image display area: the current cursor location will become the bottom right corner of the new display. The top left corner may similarly be redefined by the **^X** key.

! **Execute a UNIX command:** Enter a single command line to be executed by the UNIX */bin/sh* shell.

. **Locate image pixel:** The "**.**" key prompts you to specify a pair of pixel addresses, (*x* followed by *y*), and then positions the cursor on that pixel, possibly the image if necessary to bring that pixel within the display area. The addresses may be specified in several formats:

- a floating-point number denotes a realworld value
- an unsigned integer denotes an image pixel address
- a signed integer denotes an image pixel address relative to the current cursor location
- an integer preceded by "**:**" denotes a (possibly zoomed) display pixel address
- an integer preceded by "<" or ">" denotes a display pixel address relative to the current cursor location.

/ **Display or set GIPS variable:** The "/" key causes *gipstool* to prompt you to enter the name, and, optionally, a replacement value, for a GIPS header variable. If you only enter the variable name, *gipstool* displays the variable type, description, and current value. If you reply *name=*, the variable is deleted. If you reply *name=value*, the variable is set to *value.*

**Begin a comment:** The text that you enter, up to and including the first newline character, will be treated as a comment. If command logging is in effect (the > command), your comment will be copied to the log file.

< **Read command file:** Enter the name of an existing file containing *gipstool* commands, e.g. one that was created by the > command. If the name begins with a "|" character, interpret the remainder of the text as a shell command to be executed and its output to be a series of *gipstool*

commands to be executed. If an error is detected during any command, input will cease and *gipstool* will return to interactive input.

> **Begin command logging:** Enter the name of a file to receive a copy of your subsequent *gipstool* commands. If you enter a null line, any current logging will be terminated. If the name begins with a "|" character, interpret the remainder of the text as a shell command to be executed and its input to be the subsequent *gipstool* commands. To append the logging information to an existing file, enter a second > before the file name. In this latter case, if you omit the file name, *gipstool* will use the most recently selected log file.

1 **Reassign left button:** The **1** key causes *gipstool* to prompt you to redefine the command to be assigned to the LEFT mouse button. Your first keystroke will become the new command. Subsequent keystrokes, up to a carriage return, specify the default text to be entered when that command is invoked from the mouse button. The. LEFT button is initially unassigned.

A **Set text angle:** the angle (in degrees counter-clockwise from horizontal) at which to draw subsequent graphic text. The default is 0, i.e. upright and horizontal.

B **Set background color:** a triplet of integers in the range 0–255, to become, respectively, the new red, green, and blue intensities for background pixels. The initial background intensities are determined by the zero entry in the current colormap.

C **Set graphic color value:** an integer in the range 0–255 to specify the colormap lookup value for subsequently created graphics. Once a graphic entity has been created, it is no longer possible to change its lookup value, although you may, of course, change its displayed color by altering the current colormap.

D **Delete current graphic:** the image will be redrawn without any graphic items that were created under the specified graphic index. These items can subsequently be restored by executing the **G** command.

F **Set drawing font:** the name of the font file that defines the shape and size of characters used to create subsequent graphic text. If a relative pathname is specified, *gipstool* looks first in the current directory, then in "*/usr/lib/vfont*", and finally in "*/usr/lib/fonts/fixedwidthfonts*". Optionally, the the font name may be preceded with a style indicator, a single letter **r**, **i**, **b**,**or s**, or a corresponding number in the range 0–3, indicating that the font name that follows is to be used respectively for *roman*, *italic*, *bold*,*or symbol* text only (as selected by **f** codes in the text string).

G **Set current graphic index:** the specified number, an integer in the range 0–255, becomes the current graphic index. Subsequent graphic text will be saved under this number, and can be either permanently deleted or temporarily removed from the display by the **^K** and **D** commands, respectively.

H **Write image histogram:** the current colormap and a pixel histogram are written to the specified file, as a 256 row by 7 column *General Image*.

Row	*Contents*
1	pixel number (0 through 255)
2	corresponding realworld pixel value
3–5	corresponding rgb colormap value
6	number of pixels with this value
7	number of pixels with this value or less

The histogram is taken over the sub-image defined by the image mark and the current cursor location. Pixels with the same value as the *dnull* image header keyword are not counted. If you enter a null line, the data is sent to the socket, from which it can be read and displayed by *mongo* commands.

I **Write a sub-image:** the rectangular sub-image whose corners are defined by the current pixel mark (set by the **^S** or **^O** commands) and the current cursor location, is written to the specified disk file. If the name is omitted, the sub-image is written to the socket.

J **Write line trace:** the line of pixel values stretching from the current pixel mark to the current cursor location is written to the specified disk file as an $n \times 7$ general image whose rows represent the n individual pixels in the line, and whose 7 columns represent the following:

Row	*Contents*
1	pixel number
2	pixel x–coordinate
3	real-world x–coordinate
4	pixel y–coordinate
5	real-world y–coordinate
6	integer pixel value
7	real-world pixel value

If the file name is omitted, this image is written to the *gipsmongo* socket.

K **Write image colormap:** the current colormap is written to the specified output file. The output file will reflect the current *stretch* status set by a prior **S** command.

L **Begin pixel logging:** enter the name of the output file to which output from the **l** command will be appended. If logging is in progress, enter a null line to close the file, or enter another file name to close the original file and open a new one.

M **Read new colormap file:** the file specifies the translation between pixel values and displayed colors. *Gipstool* first looks in the current directory, and then in "*/usr/local/lib/cmap*". The new colormap table is applied to the current window **and** to all other windows that share the same colormap name (see the description of the *gipstool* **–C** option, above).

N **Set numeric precision:** a digit in the range 1 through 14 specifying the number of decimal digits of precision desired for all floating-point numbers displayed by *gipstool*.

O **Set text origin code:** an integer in the range 0–8, specifying the relationship between the mouse cursor location and the origin of text strings created by the **t** command, as follows:

1	2	3
4	5	6
7	8	9

Thus, if the origin is **1**, text will be written to the right and below the cursor location, whereas origin **0**, will cause the text to be centered about the cursor, and so forth.

P **Set pan increment:** the increment, a non-negative integer, defines the number of screen pixels by which the display will be moved by one of the incremental panning commands: **^B**, **^F**, **^N**, and **^P**. The initial value is **0**, a special value that *gipstool* interprets as denoting half the current width (or height, if panning vertically) of the displayed image.

R **Recall a tagged location:** prompts for a tag name. This should be one of the names supplied by previous **T** commands to remember image locations. The result will be to reset the image cursor to point to that pixel location, possibly panning the image to do so. If no tag name is supplied, the last accessed image tag will be restored.

S **Stretch colormap:** applies a contrast-enhancement to the colormap, based on the distribution of pixel values in the stored image. This distribution is computed whenever the stored image is altered: usually, this occurs only during the initial load phase, but if you have specified the **–c** flag, the histogram will be recomputed whenever you resize or pan the displayed image.

No new colormap intensities are created; rather, the transformation selectively reassigns color/intensity values to increase the contrast between most-likely pixel values. At the same time, contrast will be lost between less-likely values. *Gipstool* prompts for an integer percentage in the range 0–100, setting the ''strength'' of the transformation. 100% denotes a ''maximal'' stretch, whereas 0% restores the original colormap values. Use of the **s** command will reset the stretch to 0%.

T **Set image tag:** prompts for a tag name. The current cursor location is then remembered under this

tag name, and may be recalled via the **R** command. The list of current tag names may be scrolled through by means of the **^R** command.

V **Set foreground color:** a triplet of integers in the range 0–255, to become, respectively, the new red, green, and blue intensities for foreground pixels. The initial foreground intensities are determined by the zero entry in the current colormap.

W **Set paintbrush width:** sets the *x* and *y* dimension of the area that will be painted by the next **p** command.

Z **Set zoom pixel size:** supplies three parameters for the **^W** (zoomed window) command: the first determines the number of screen pixels that will be displayed in the zoomed window, the second specifies the additional zoom factor to be applied. The third specifies the color of the box and cross-hairs. If the screen image is already zoomed (i.e. *gipstool* was invoked with the **–z** or **–Z** options), the result will be a doubly zoomed sub-image.

a **Draw both axes:** prompt for changes to axis-drawing parameters (see below), and draw the axes, using the current graphic settings i.e. color, font, index, and numeric precision, specified by **C**, **F**, **G**, and **N** commands, respectively.

f **Set frame label:** prompts for a text string to replace the label in the stripe at the top of the image display. If the string is null, the stripe will be removed.

h **Display header item:** the header item of this name will be converted to ASCII and inserted in the display as a graphic item according to the current graphic settings, i.e. angle, color, font, index, and origin, specified by the **A**, **C**, **F**, **G**, and **O** commands, respectively.

l **Log pixel value:** the coordinate and data values of the current pixel are appended to the current pixel-logging file (opened via the **L** command). The format is specified by the user:

format	*action*
%x	replace with the x–pixel coordinate
%rx	replace with the realworld x–coordinate value
%tx	replace with the x–axis title

and similarly with **y** or **d** in place of **x .**

m **Draw non-local grid lines:** The **m** command is only valid for images in non-local projections, i.e. those created by the *ihgrid* and *ihgeom* programs, and therefore contain a header variable called *mapping* that defines the relationship between their *x* and *y* pixel addresses and their real-world coordinates. The spacing of the grid lines is given by the *xaxis* and *yaxis* header variables.

p **Paint pixel:** sets the pixel that is pointed to by the cursor to the current graphics drawing color. When the **p** command is assigned to the left mouse button, the painting will take place wherever the cursor is dragged with the left button held down, provided a **^S** or **^O** command isn't in progress. Alternatively, if the **SHIFT** key is held down at the same time, the pixels will be cleared instead. The size of the area that is drawn or cleared may be changed by the **W** command.

s **Enter colormap stretch points:** applies a linear contrast-enhancement to the colormap, specified by the minimum and maximum *realworld* values supplied. No new colormap intensities are created; rather, the transformation reassigns the pixel values between the limits to the available colors. Use of the **S** command will reset the linear stretch limits to the values of header variables *dmin* and *dmax*, respectively.

t **Enter text string:** the string will be converted to a graphic item according to the current graphic settings, i.e. angle, color, font, index, and origin, specified by the **A**, **C**, **F**, **G**, and **O** commands, respectively. The size and style of the text string may be changed by means of *troff*-style escape sequences embedded in the string. See *comlabel*(3i) for more details.

u **Undo last graphic:** the graphic most recently defined under the current graphic index is removed from the display. In the case of commands that generate multiple graphic items, e.g. axes, the **u** command removes one item at a time.

v **Enter cursor location** the *x* , *y* , and/or *data* values of the pixel pointed to by the cursor are converted to a graphic item according to the current graphic settings, i.e. angle, color, font, index, and origin, specified by the **A , C , F , G ,** and **O** commands, respectively. The number and order of the values are specified by the user: enter **x , y ,** and/or **d** in any order, and each optionally preceded by **r** signifying that the realworld value is to be used instead of the image value.

w **Write overlay:** write the 8-bit overlay plane to the specified disk file in GIPS format. This image will contain pixels set by execution of the **p** command. The output file will be overwritten by the overlay.

x **Draw x–axis:** prompt for changes to *x*–axis drawing parameters (see below), and then draw the *x*–axis.

y **Draw y–axis:** prompt for changes to *y*–axis drawing parameters (see below), and then draw the *y*–axis.

{ **Read tag file:** read tag definitions from an external file. The tags describe locations within the image. Null lines and lines that begin with # are discarded. Each remaining line must begin:

tag_name xreal yreal comment_string

where *tag_name* must contain no whitespace, and *xloc* and *yloc* are the realworld tag coordinates. The tag definitions are appended to the current tag list. They may be saved again by the } command.

} **Save tag file:** writes the current tag list to the specified file. A tag file can be read again by the { command.

AXIS DRAWING PARAMETERS

The *gipstool* axis-drawing commands **a**, **x**, and **y** can draw axis lines, scale marks, tick marks, scale annotation, axis labels, and grids. In addition, the **m** command draws grid lines on images with non-separable coordinates. The resulting graphic items will be displayed in the current graphic color and index, as displayed by the *gipstool* **^I** command.

The axis labels are drawn using the current definitions of the header variables *xtitle*, *xfont*, *ytitle*, and *yfont* of the input image. Both title and font must be defined or the title will be suppressed. The orientation is implied by the font file names: names ending in "*r*" denote ‘‘rotated’’ (i.e. vertical) titles, otherwise they will be horizontal. No intermediate angles are possible.

The remaining axis graphics are constructed from the *xaxis* and *yaxis* header variables of the input general image. Each variable should contain a list of sub-fields of the form "*item=value*", separated by blanks or tabs, for example:

xaxis="mark=10 tic=1 font=I.6 by=15.0 place=tbci"

When the **a**, **x**, or **y** command prompts you to "*Enter updated axis parameters:*", you may specify new sub-field values for *xaxis* and *yaxis*; i.e. replies to **x** command will update *xaxis*, replies to **y** will update *yaxis*, and replies to **a** will update both *xaxis* and *yaxis*. For instance, if you execute the **x** function, specifying "*font=B.6*", *gipstool* will generate an *x*–axis using the "*B.6*" font (6–point Hershey bold) for scale annotation. In addition, *gipstool* will alter its stored value of the *xaxis* header variable to include the new definition of the "*font=*" sub-field.

Here are the definitions of the possible *xaxis* and *yaxis* sub-fields:

mark= the maximum number of scale labels that may be drawn along the axis. If not specified or set to 0, the default is 1000.

tic= the number of tic marks to be inserted between each scale interval.

font= the font in which to write the scale annotation. *Gipstool* searches for the font file in the current directory, then in "*/usr/lib/vfont*", and finally in "*/usr/lib/fonts/fixedwidthfonts*".

place= one or more characters in the string "**lcriotmb**" determining to whether the titles and scale notation should be **L**eft, **C**enter, or **R**ight justified, inserted **I**nside or **O**utside the image

area, and placed at the **T**op, **M**iddle, and/or **B**ottom of the image.

loc= defines the meaning of ‘‘middle’’ titles – i.e. within *xaxis*, *loc* specifies the *y*–value at which the *x*–axis is to be drawn.

from=
to=
by= together define the values at which the scale annotations are to be drawn. If specified in floating-point form, realworld values are assumed. Otherwise, they are assumed to represent input image pixel values. The same notation is used for specifying the **loc=** sub-field.

line= an 8-bit mask used to construct lines connecting the top and bottom of the image at each scale mark. If "*line=0*" (the default), no connecting lines will be drawn.

form= a non-default format in which to draw the scale annotation. The format is that of the *printf*(3S) function, and should contain no embedded white-space. The default is "*%.6g*" unless *scale=d* or *scale=h*, in which case the defaults are "*%d\013%d’%.12g’’*" and "*%dh%dm%.12gs*" respectively. Therefore, there must be at least three percent fields in these cases, and one in all others.

ticlen= the length in screen pixels of the scale and tic marks.

scale= a code letter which causes axis scale numbers to be formatted as:

- **s** simple fixed or float numbers
- **l** ten to the power of ...
- **d** degrees, minutes, and seconds
- **h** hours, minutes, and seconds

FILES

/usr/local/lib/cmap/*.cmap	colormap files
/usr/local/tables/cprofile	image header definition table
/usr/local/tables/gipstool.menu	command definition script

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

"*Guide to GIPS*" and "*A GIPS Tutorial*", by Peter Ford, MIT Center for Space Research, 1984.
gipsmongo(1), ihs(1), iged(1), suntools(1), comlabel(3)

ACKNOWLEDGEMENTS

The technique of "*error diffusion*" was first brought to my attention by John Tonry, the author of the *mongo* program.

DIAGNOSTICS

When an error is detected before the SunView frame is created, *gipstool* writes a message to the standard error stream *stderr*, and halts with system code 1. Messages generated by GIPS library routines are described in *ihread*(3). Other possible messages are

- bad –z value: *value*
- bad x–pixel origin, size, or offset: *value*
- bad x–view size, or left/right border: *value*
- bad y–pixel origin, size, or offset: *value*
- bad y–view size, or left/right border: *value*
- can’t create image sub-window
- can’t create tool
- can’t open image sub-window
- cannot create saved pixrect
- multiple input file(s): *name*
- x–limits outside image

- xzoom != yzoom not implemented for depth=1 display
- y–limits outside image

BUGS

The axis drawing commands **a**, **x**, and **y** have some bizarre problems. Although it is intended that they should generate "reasonable" scale numbers, they frequently need a careful choice of *from*, *to*, *by* parameters to work properly.

The text-entry pop-up is sometimes not fully repainted, particularly on color screens. The user can force a repaint by selecting the *Redisplay* action from the frame menu.

NAME

git – list the header or pixel values of a General Image

SYNOPSIS

git [**–achns**] [**–p** *precision*] [*name*]

DESCRIPTION

The *git* command converts the pixel values of a General Image to ASCII numbers, writing them to the standard output *stdout*. If *name* is specified, the image is read from the file of that name; otherwise, it is read from the standard input stream *stdin*. If the variable *dnull* is defined in the image header, repeated null pixels are represented by the notation "**[***n***]**", where *n* denotes the repetition count. The following flags and options apply:

–a applies to Fourier-transformed images with complex pixels (*dtype=c*). The pixel values are converted to real by taking their norms. Otherwise, they will be displayed as "**(***cos***,***sin***)**", representing the Fourier *cosine* and *sine* components.

–c compress repeated output pixel values, i.e. display them as "*count***value*".

–h also copies the General Image header to *stdout*, one header item per output line. The image values follow this header.

–n turns on the **–h** flag, writing the General Image header to the standard output stream *stdout*, but suppressing the output of pixel values.

–p *prec* specifies the number of decimal digits of precision to be kept when converting floating-point pixels. *prec* must be an integer in the range 0 through 12, the default being 6. Non-zero pixel values that would be rounded to zero under the chosen precision are output as "**~ 0**".

–s inserts two newline characters after each output raster, and displays the raster number. Normally a single space separates each pixel value and a single newline signals the end of each raster.

FILES

/usr/gips/tables/cprofile image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

gin(1), ihs(1), ihread(3)

DIAGNOSTICS

When an error is detected, *git* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- dtype=*code* conflicts with –a
- illegal –p value: *num*

NAME

iged – interactive graphic-plane editor

SYNOPSIS

iged [**–***command ...*]

DESCRIPTION

When invoked with arguments, *iged* is said to be in *pipeline* mode: it expects to read a Generalized Image header file from the standard input stream *stdin*, and its arguments are then interpreted as a series of subcommands, each of which consists of a series of words, the first of which must begin with a dash followed by a command character, e.g.

iged –g 1 –c red –v xfont=R.10 –a < image.one

If *iged* is invoked without arguments, it is said to be in *interactive* mode, and reads commands from the standard input stream *stdin*. The command format is identical to that in *pipeline* mode, except that each command must end in a "newline" character, and the initial dash is optional (except in subcommand macros – see the ´"´ command, below).

Iged first tries to read the file "*.igedrc*" in your home directory. If this file exists, its contents are interpreted as *iged* subcommands and are executed as if they had been issued at an interactive terminal. Some subcommands are not accepted – e.g. those that redirect input and output streams and those that generate listings.

Iged can maintain temporary and permanent updates to several graphic and image planes simultaneously, but for ease of command syntax, only one of each is *current* at any one moment. The **g** and **i** subcommands define, respectively, the current graphic and image. They also cause that plane to be displayed.

If you get stuck within interactive*iged*, on-line help is available by typing "**?**" – this displays a list of subcommands – or type "**??**" – which tells you about *iged* command syntax. In addition, a number of subcommands have "help" modes, e.g. the "**l**" subcommand (entered without arguments) displays a number of useful internal variables, "**f**" tells you which character-drawing fonts are currently loaded, and "**f***" lists all possible fonts.

Many *iged* sub-commands take arguments that refer to a particular pixel location within an image or graphic. Both *x* and *y* coordinates must be specified, in that order; there are seven ways of specifying either *x* or *y*, the last two of which are only accepted in interactive mode:

- An unsigned integer, representing a pixel address in the currently displayed image. Unless *iged* has read a General Image header (e.g. via the "**h**" subcommand, or from *stdin* in pipeline mode), this address is identical to the absolute x/y display address described under the next item below. Otherwise, it is an address relative to the top left corner of the image as defined by the respective *xorg* or *yorg* header variable.
- A colon followed by an unsigned integer represents an absolute pixel address in the current display. *x*–addresses increase from left to right, *y*–addresses from top to bottom; the top left pixel is "**:0:0**".
- A floating-point number (i.e. a numeric string containing a decimal point or the letter "e" or "d"), representing a "realworld" *x* or *y* coordinate value.
- The character @ representing the current *x* and *y* location of the display target. Unlike the **%** specification below, the user has no opportunity to re-position the target.
- The character < or > followed by an unsigned integer, representing that many x (y) pixels respectively to the left of (above) or to the right of (below) the current target location. *iged* gives the user no opportunity to reposition the target.
- The character **%** represents the *x* and *y* location of the display pointer. *Iged* will prompt you with a # character and you must then re-position the display pointer and hit a carriage return. Typing any other key before the carriage return will cause the *iged* sub-command to terminate immediately.
- A signed integer represents an offset from the current position of the display pointer (remember that

the x-axis coordinates increase left to right, and the y-axis coordinates increase *top to bottom*). *Iged* will prompt with a # in the manner described above.

Aside from the metacharacters @ and **%**, which must appear by themselves, the other formats may be freely mixed, i.e. " **-10** >**21** ", " **101.2 :4** ", etc., but remember that "**%**" and signed integers are only permitted in interactive mode.

Iged maintains a knowledge of real-world coordinates by reading General Image headers. Only one header may be *current* at any one time, although you can define special compound commands called *macros* which will change the *iged* environment with only a couple of keystrokes. These are described under the ´"´ subcommand, below, and may be used to switch between image headers.

In the following table, the subcommand character is shown in **boldface** and its skeleton argument list is *italicized*. All meta-characters in filenames will be expanded by *csh*(1) before execution. (The choice of shell may be changed via the `**!***´ command.)

A "*%*" sign in the following command definitions indicates the use of any of the 7 acceptable kinds of target designation, including a single percent sign itself. Square parentheses around an argument indicate that its presence is optional. Ellipsis, i.e. "...", indicates that one or more repetitions of the preceding argument are also accepted.

a *[parms]* **Generate x- and y-axis scales and titles**, using header variables *xtitle*, *ytitle*, *xfont*, *yfont*, *xaxis*, and *yaxis*. Sub-fields in *xaxis* and *yaxis* may be overridden by *parms* (see under *Axis Specifications, below*). Any changes that you specify will update subfields in the incore *xaxis* and *yaxis* variables. If you wish to update *xaxis* and *yaxis* separately, use the **x** and **y** subcommands instead.

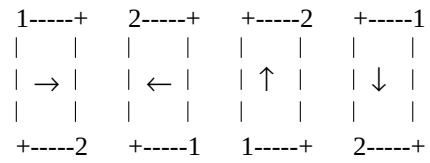
b **Enter block-mode.** When executing in *block* mode, *iged* refreshes the graphic plane only when terminating, when changing planes, or when an explicit **w** command is issued to the plane. Block mode is the only mode permitted when *iged* takes its input from the standard input stream. (cf. single-command mode, the **s** subcommand).

c *[color]* **Change the color of the currently displayed graphic.** If *color* is omitted, *iged* displays the current color setting. The following 16 'colorcode' mnemonics are recognized (they may be entered in upper or lower case):

BLA	black	LPI	light pink
BLU	blue	MAG	magenta
CYN	cyan	ORN	orange
DGR	dark green	PNK	pink
DTU	dark turquoise	RED	red
FGR	forest green	TUR	turquoise
GRN	green	LOR	light orange
YEL	yellow	WHT	white

d *%...%* **Draw an open polygon**, i.e. a line is drawn from one screen location to the next. If more than two locations are specified, the line is continued to the third point, and so on. The type of line is specified by the "k" command.

e *%%* **Insert a data wedge into the current image.** A box is inserted in the current graphic, and the corresponding region of the current image is filled with pixels in ascending value order. The orientation of the values within the box is determined by the order in which the extremities of the box are defined:



The *1* and *2* denote the locations of the first and second pixel addresses specified, and the arrows denote the direction of increasing wedge value. If you make a mistake, you can remove the wedge with the **U** command, and undo the graphic border with **u**.

f *[name]* **Change the current title-drawing font** – *name* specifies a binary font file in *vfont*(5) format. If a relative pathname is used, *iged* first searches for a file of this name in the current directory, then in */usr/lib/vfont*. Names ending in "*r*" are assumed to be rotated (vertical) styles. If the font is of "Roman" type, *iged* will also attempt load the italic, bold, and/or symbol fonts of that type and size. The alternate fonts may then be selected by troff-style in-line escape codes as described in *comlabel*(3). Alternate fonts may also be specified individually by the **F** subcommand. When *name* is omitted, *iged* displays the name of the currently used font. A listing of all available fonts in */usr/lib/vfont* may be obtained by entering "*f* ∗", and any characters following the "∗" are interpreted as shell metacharacters used to select a subset of these names, e.g. "*f* ∗ *nonie*∗" displays all of the "*nonie*" fonts.

g *[#]* **Change the "current" graphic plane.** # specifies which graphic plane is to receive output. The contents of the current plane are saved and the new plane is restored (if previously saved in this *iged* session). If # is omitted, *iged* displays the numbers of the current plane (marked with a "∗") and all saved planes. To delete a saved plane, enter the negative of its number.

h *[name]* **Read a General Image header.** *Name* is expanded by *csh*(1) and, assuming that a file of that name is found, its GIPS header is read and stored. Subsequent knowledge of "realworld" coordinate and pixel values will be determined by this header. If *name* is omitted, *iged* displays the file name of the currently stored GIPS header.

i *[#]* **Change the "current" image plane.** Image plane number # will supply data values for subsequent **e**, **n**, **H**, **N**, and **U** commands. Initially, image plane 1 is current. If *name* is omitted, the current image plane number is displayed.

j *% item* **Insert header item into current graphic.** The value of incore header variable *item* is (if necessary) converted to a character string and that string is written into the current graphic at the specified pixel address.

k *[linetype]* **Change line drawing pattern.** *Linetype* specifies an 8-bit pattern to be repeated by all line-drawing subcommands, e.g. "**d**". *linetype* may be an unsigned decimal integer, an octal integer (with leading '0'), or a string of 8 characters. When numeric, the line type is taken from the last 8 bits of the binary number; when alphanumeric, the bit pattern is specified by alphanumeric (bit=1) and non-alphanumeric (bit=0) characters in the string. e.g. "*k 23*", "*k 027*" and "*k ---x-xxx*" are equivalent. If *linetype* is omitted, iged displays the currently defined line pattern.

l *[item ...]* **Display incore header variables or GIPS statistics.** *item* represents the name of a GIPS header variable, whose value is displayed. If *name* is omitted, *iged* displays the current image and graphic numbers, and the most recent GIPS headers that have been read or merged. If "∗" is used for *item*, the entire header is displayed. If the "∗" is followed by a valid file name, that name is shell-

expanded by *csh*(1), and the header is written to that file.

m *[name]* **Merge additional GIPS header fields.** *Name* is expanded by *csh* and, assuming that a file of that name exists, its GIPS header is used to augment the header that was last read via the "**h**" command. If *name* is omitted, *iged* displays the name of the last GIPS header file thar was merged.

n *%* **Display an image pixel value.** The 8-bit value of the specified pixel in the current image plane is transformed according to the information in the GIPS header and is displayed (along with its *x* and *y* coordinates), in its screen and realworld units, as follows:

x: *xpixel xmin xmax* **y:** *ypixel ymin ymax* **d:** *dpixel dmin dmax*

where *xpixel* represents the pixel coordinate, *xmin* and *xmax* the realworld limits spanned by the pixel, etc. Realworld fields outside the display limits are reported as **MAX** or **MIN**, or, if no header has been read, as **NUL**.

o *[#]* **Change relative origin of graphic text** written into the current graphic by subsequent **t** commands. Values 0 thru 8 determine the text origin according to the following table

1	2	3
4	0	5
6	7	8

e.g. "*o 0*" (the default value) causes text to be centered about the chosen location, "*o 1*" causes it to be written wholly to the bottom right of it, etc. For rotated fonts (with names ending in 'r'), left, right, top, etc. refer to the rotated characters themselves, i.e. *top* indicates the direction to the *right* of the screen. If # is omitted, *iged* reports the current origin code.

p *[#]* **Change floating-point precision** (the maximum number of significant digits displayed) of all subsequent output of floating point quantities. If # is omitted, the current precision is displayed.

q **Terminate the *iged* session.** When *iged* is used interactively (i.e. reading commands from the standard input stream), the command must be repeated to become effective.

r *[name]* **Read a graphic plane.** File *name* overwrites the current *iged* graphic. If *name* is numeric, it is assumed to refer to the graphic plane of that number; if omitted, the current display graphic is read back into *iged*; otherwise, *name* is expanded by *csh* and the first matching file is read.

s **enter single command mode.** Updates to the current graphic will be displayed as they are made. (cf. the **b** subcommand).

t *% [text]* **Position target or write graphics text** – if *text* is omitted, position the display target at the specified pixel. If this is not currently in view, the image will be roamed to do so.

If *text* is supplied, that text string is written into the current graphic plane using the current font, in a location whose relation to the specified pixel is set by the **o** subcommand.

u **Undo the last change** applied to the current graphic plane. Another **u** immediately following will undo the undo.

v *item [= value] ...*

Update incore header variables – variable *item* is assigned a new value. If *value* contains blanks, enclose it in double quotes. If *value* is omitted, the variable is deleted. If "= *value*" is omitted entirely, the variable type and description

of *item* are displayed. Multiple items may be processed on the same line, but only one at a time may be deleted. Also, it is only possible to delete the most recent *history* item from an incore header, but any number may be added.

w *[name]* **Write the current graphic** – to file *name* (expanded by *csh*). If *name* already exists, use "*w!name*" instead. If *name* is numeric, the current graphic is written to the graphic plane of that number. If *name* is omitted, the current graphic is updated.

x *[parms]* **Draw x-axis** lines, scale marks and numbers, and title, using header variables *xtitle*, *xfont*, and *xaxis*, with the latter updated (for the remainder of this *iged* session) by any *parms* that you specify (see *Axis Specification*, below).

y *[parms]* **Draw y-axis** lines, scale marks and numbers, and title, using header variables *ytitle*, *yfont*, and *yaxis*, with the latter updated (for the remainder of this *iged* session) by any *parms* that you specify (see *Axis Specification*, below).

z *name %...%* **Report matching General Image pixels.** *Name* is C-shell-expanded and the General Image file of that name is scanned for pixels whose *realworld* x- and y-coordinates overlap the coordinates of the pixel(s) specified in the **z** subcommand.

C **Clear the current graphic** plane to binary zeroes.

D *%...%* **Draw a filled polygon.** The area enclosed by the specified series of pixel locations will be filled within the current graphic. If the starting and ending locations are not identical, the figure will be closed automatically.

F *[type [name]]* **Define alternate fonts.** *Name* specifies an alternate font definition file (cf. the **f** subcommand). If a relative pathname is used, *iged* first searches for a file of this name in the current directory, then in "*/usr/lib/vfont*". Names ending in "r" are rotated (vertical) styles. *Type* must be one of the following words or abbreviations:

TYPE	ABBREVIATION	ESCAPE CODE	DESCRIPTION
roman	R or r	\fR or \f1	Major (default) font
italic	I or i	\fI or \f2	Italic font
bold	B or b	\fB or \f3	Bold font
symbol	S or s	\fS or \f4	Symbol font

The alternate fonts may then be selected by the troff-style in-line escape codes as described in *comlabel*(3). When *name* is omitted, *iged* displays the name of the current alternate font of that type.

G *# oper [~]* **Merge graphic planes.** The specified saved graphic plane number # is merged with the current graphic according to the operator *oper*, where *oper* must be one of the following:

|	logical ORing (inclusive OR)
&	logical ANDing
^	logical XORing (exclusive OR)
=	bitwise copying

In addition, the "~" operator may follow one of the above, indicating that the source should be the logical NOR of the saved graphic. The saved graphic plane is never altered.

H *[code]* **Start or stop automatic histogramming.** *Code* specifies one or more automatic histograming and pixel display functions to be executed by all subsequent line-drawing and area-filling subcommands, until canceled by a subsequent **H** command without any *code*. If *code* is "**s**" or "**t**", pixel values will be accumulated and written as a histogram table when histograming mode is canceled. The

modes differ in that "**t**" histograms are displayed down the screen, whereas "**s**" histograms are displayed across the screen. If *code* is "**v**" or "**r**", pixel and coordinate values will be displayed whenever the corresponding graphic bits are set in subsequent *iged* commands. "**r**" differs from "**v**" in that *truncated* realworld units are displayed, rather than upper and lower data limits. If *code* contains an "**n**", null pixels will not be listed. A subsequent "**Hz**" subcommand will resume listing null pixels. "**s**" and "**t**" and "**v**" (or "**r**") mode may be active simultaneously.

L *% name [scale]* **Insert a graphic logo.** *Name* is expanded by *csh* and the file of that name is opened. The contents are translated into a bit pattern to be written into the current graphic at the specified location. Each newline-terminated line of *name* contributes to a single graphic raster. Within the line, blanks turn corresponding graphic bits *off* and non-blanks turn them *on*. Tabs will be expanded into spaces in the manner of the *expand*(1) command. Specifying *scale* as a positive floating-point value will cause the resulting graphic to enlarge or shrink accordingly. The primary use of this sub-command is to insert a logo or other often-drawn figure into a graphic.

N *%* **Report a truncated pixel value.** The 8-bit value of the specified pixel in the current image plane is transformed according to the information in the GIPS header and is reported in pixel and realworld units, as follows:

x: *xpixel xreal* **y:** *ypixel yreal* **d:** *dpixel dreal*

where *xpixel* represents the pixel coordinate, *xreal* the realworld value, etc. Unlike the **n** subcommand, which displays the upper and lower realworld limits, **N** reports a single average "truncated" value for the realworld x, y and data values.

O *% %* **Draw a circle.** The first location determines the center, the second a point on the circumference of a circle to be inserted into the current graphic using the *linetype* specified by the **k** subcommand.

P *% n* **Change an image pixel value.** The value of the specified pixel in the current image is changed to *n*, an integer in the range 0 through 255.

R *[count] command*

Repeat another subcommand. The subcommand "*command*" is executed repetitively "*count*" times. If *count* is omitted, *command* is executed 100 times, or until it encounters an error. When repeating shell commands, i.e. "**R ! ...**", typing **CTRL-C** will terminate the loop; similarly, when repeating a subcommand that prompts with "#" for you to reset the display target, i.e. "**R n %**", hitting any other key before carriage-return will also exit from the repeat-loop. Some commands may not be executed repetitively, e.g. **R** itself, and this also applies to macros containing those commands.

S *[code] [%...%]* **Draw a smooth curve** passing through the specified pixels. *Code* represents one of more of the following options:

m*n*	select interpolation method *n*: **0**: spline (default), **1**: quartic, **2**: circular.
p*n*	select point-smoothing density *n*: **0**: coarse, **1**: medium (default), **2**: smooth.
q	don't update the display after each segment.
r	prompt repetitively for next cursor location.
w	join last point to first.

and, unless option "**r**" is specified, these must be followed by at least 3 cursor location fields.

T *% code...* **Insert pixel value into graphic** – the value of the chosen pixel is converted to *truncated* realworld units. *Code* must be one or more characters *x*, *y*, or *d*, signifying that the realworld values of either the x-coordinate, y-coordinate, pixel data value, or some combination of them, is to be written into the current graphic plane, using the current value of the origin offset to locate the text (see the **o** subcommand above).

U **Undo most recent image update** – the effect of the most recent **e** or **P** subcommand is undone. Unlike the **u** subcommand, a second execution of **U** will *not* cancel the effect of the first.

< **[</|]** *name* **Redirect the input command-stream.** File *name* is opened and its contents interpreted as *iged* subcommands, one per line. Unless a second "<" is included, the subcommands are echoed as they are executed. Alternatively, if a leading "|" is specified, *name* is interpreted as a command or pipe whose standard output is to be read as a series of *iged* commands. If *name* is omitted, *iged* displays the name of the last file executed.

> **[>/|]** *name* **Redirect output listings.** Subsequent *iged* listings (as distinct from prompts or error messages) will be directed to file *name*. If a second ">" is used, the output will be appended to the file; otherwise any existing file will be overwritten. Alternatively, if a leading "|" is specified, *name* is interpreted as a command or pipe whose standard input is to receive subsequent *iged* listing output. If *name* is omitted, listings will be sent to the standard output stream; normally, this will be the terminal.

! *text* **Execute a shell command.** *text* is passed to *csh* and executed immediately. It is possible to execute other commands that reference the display, including a fresh copy of *iged* itself. The choice of shell program may be changed by issuing the command `**!***shell´*, where *shell* is the absolute pathname of the shell command, e.g. `*/bin/sh´*. This shell will be used instead of the default, `*/bin/csh´* for all subsequent filename expansions performed by *iged*.

’’ *char [value]* **Define or invoke an *iged* macro** – the macro is named *char* (a lower-case letter a–z). and represents a sequence of *iged* subcommands that will be executed together whenever the macro is invoked. *value* specifies the string of subcommands, in which each subcommand character must be preceded by a minus sign. If *value* is omitted, the macro *char* is executed. If *char* is omitted, all currently defined macros are listed.

[*% %* **Draw an open rectangle** – a box is drawn with the specified locations at its corners. The sides of the box will be oriented parallel to the x- and y-axes.

] *% %* **Draw a closed rectangle** – a filled box is drawn with the specified locations at its corners. The sides of the box will be oriented parallel to the x- and y-axes.

? *[command]* **Return information about an *iged* subcommand.** If *command* is omitted, a list of all *iged* subcommands and features is displayed; otherwise, detailed information is displayed about that particular subcommand or feature. *Iged* actually reads its UNIX manual file and pipes the relevant sections through *nroff*(1).

. *[text]* **Pass a command to the image display.** Assuming that the display device can accept ASCII command strings, *text* following the period is converted to upper case and is executed as if it were entered directly at the display keyboard. If *text* is omitted, the display device is "reset".

= *[text]* **Re-execute an *iged* subcommand** - where *text* begins with a subcommand character. Any additional text will be appended to the most recent instance of that subcommand. When *text* is omitted, the "**=**" subcommand lists the most recent instances of each *iged* subcommand. "**==**" executes the most recent non-macro

subcommand.

AXIS SPECIFICATION

The axis titles and the fonts in which they are written are specified by GIPS header variables **xtitle**, **ytitle**, **xfont**, and **yfont**, but they are located with respect to the axes themselves by information in the **xaxis** and **yaxis** string items, each of which contains a number of sub-items of the form "**item=value**". A sample pipeline-mode x-axis drawing command might be specified as

iged –x mark=10 tic=1 font=I.6 by=15.0 place=tbci

In interactive mode, the leading dash may be omitted and the entire sequence must end with a carriage return. The permitted sub-fields are as follows:

mark specifies the maximum number of scale labels that these commands attempt to insert along the axis line.

tic specifies the number of tic marks to be inserted between each scale interval.

font specifies the Versatec/Varian font to be used to write the scale numbers.

place specifies one or more characters in the string "**lcriotmb**" according to whether the titles and scale notation should be **l**eft, **c**enter, or **r**ight justified, inserted **i**nside or **o**utside the image area, and placed at the **t**op, **m**iddle, and/or **b**ottom of the image.

loc defines the meaning of "middle" titles – i.e. within *xaxis*, *loc* specifies that y-value at which the x-axis is to be drawn.

from,to,by

specify the steps to be used in drawing the axis scale notations. If specified in floating-point form, i.e. with an explicit decimal point or exponent, realworld values are assumed. Otherwise, they are assumed to represent pixel values. The same notation is used for specifying the **loc** option.

line specifies an 8-bit mask used to construct grid lines connecting each scale mark. Refer to the **k** subcommand for details. If *line=0* (the default), no connecting lines will be drawn.

form specifies a non-default format in which to draw the axis scale numbers. The format is specified as for the UNIX *printf*(3S) function, and should contain no embedded whitespace. The default is "*%.6g*" unless *scale=d* or *scale=h*, in which case the defaults are "*%d\(de%d\'%.12g\'\'*" and "*%dh%dm%.12gs*" respectively. Therefore, there must be at least three percent fields in these cases, and one in all others.

ticlen specifies the length in display pixels of the axis scale and tic marks.

scale specifies a code letter which causes axis scale numbers to be formatted as: **s** (simple fixed or float numbers), **l** (ten to the power of ...), **d** (degrees, minutes, and seconds), or **h** (hours, minutes, and seconds).

COMMAND EDITING

In interactive mode, *iged* maintains a list of the most recently issued instance of each subcommand. That command can then be reissued by prefixing an equals sign to the command character. Any text following the subcommand character will be appended to the command. E.g. if the last instance of the "**t**" command was "**t <20 >30 Tag**", then typing "**=t 2**" results in the issuing of "**t <20 >30 Tag 2**".

Similarly, the "^" character invokes string replacement in the most recent instance of a subcommand. Continuing the example begun in the previous paragraph, typing "**t^30^40^**" would result in the subcommand: "**t <20 >40 Tag 2**". The third carat may be omitted if the replacement string contains no blank characters, and the second carat can be omitted if it is your intention to remove the search string entirely. The change is applied once only to the subcommand string, unless a "*g*" character follows the third carat, when the change is applied *globally* to all occurrences of the search string.

FILES

/usr/gips/tables/cprofile GIPS header definition

/usr/gips/lib/man.iged	nroff macros for help messages
/usr/lib/vfont/*	graphics font definition files

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihs(1), comsubs(3), comlabel(3)

BUGS

So far, *iged* has only been implemented on the Comtal Vision One/20 display system. See *comsubs*(3) for details on the display interface.

NAME

iha – perform pixel arithmetic on one or more General Images

SYNOPSIS

iha [**–BSp**] [**–d** *dir*] [**–amo** *file*] [**–l** *libs*] [*expr*...] [*name*...]

DESCRIPTION

The *iha* command reads one or more input General Images and generates an output General Image that represents the result of a given arithmetical operation applied pixel-by-pixel. All input and output images must contain the same number of *x* and *y* pixels, (defined by header variables *xnum* and *ynum*), although the pixels may be represented by different *dtype* formats or *dscale* mappings.

The operation, which we shall refer to as a **script**, must be written in C source code, and must either be stored in the file specified by the **–a** option or must appear as "*expr*" in the *iha* command line. Unless any of the input or output images contain complex pixels (*dtype=c*), the form of the *expr* should be

p = *funct*(p$_1$,p$_2$,...p$_n$,x,y)

specifying that the output pixel *p* should be created by an operation performed on the corresponding pixels of input images 1 through *n*. *x* and *y* may be used to reference, respectively, the column and row index of the image, counting the first pixel (i.e. the top left of a displayed image) as *x=0, y=0.* For example, the following script

iha "**p = (x>=20 && y>=20) ? p1+p2 : 0**" **gi.one gi.two** > **gi.three**

will create "*gi.three*" from "*gi.one*" and "*gi.two*" by summing corresponding pixels, but setting the first 20 rows and columns to zero. The script syntax is C language. In fact, *iha* uses the C compiler to create a temporary program, so the script can refer to other subroutines provided these are made known to the UNIX loader (see the **–l** option). Note that the *expr* script is enclosed within quotation marks to "hide" any meta-characters from the shell, and to instruct *iha* to treat it as a single argument.

The following header variables quantities may also be used in the script, provided they are defined in the header of the input image, or have been added via the **–m** option:

dmax, dmin, dnull
xmax, xmin, xlfft, xnum
ymax, ymin, ylfft, ynum

The following example shows how you might use *iha* to apply a user-written function *fun1()*, a C function returning an *int* value, that you have compiled into the object file *fun1.o*, to the image file *gi.one*:

iha "**extern fun1(); p=fun1(p1)**" **–l fun1.o gi.one** > **gi.two**

The script syntax for complex pixels is similar, but they are referred to as *r* and *i* instead of *p*, e.g.

iha "**r=r1+i2 ; i=r2–i1**" **gi.one gi.two** > **gi.three**

Note the semi-colon separating the two assignment operators, just as there would be between two C statements.

–B instructs *iha* to use "brute force" conversion even when the program determines that it would be quicker to generate a conversion table, e,g, to transform two byte-pixel images into a third, a 64Kbyte conversion table would otherwise be constructed.

–S writes processing statistics to *stderr*.

–p writes the temporary C source file to the standard output stream *stdout*. Only image headers are processed. Use this flag for debugging purposes only.

–a *file* specifies the name of a file that contains the C-language expression to be applied to the input images. If **–a** is omitted, *iha* assumes that the expression is its first non-flag-related argument.

–d *dir* overrides "*/tmp*" as the directory in which to allocate temporary files.

–l *libs* specifies one or more additional object files or libraries to be loaded along with the

temporary C program. *iha* automatically includes the GIPS routine library and the C math library. You must supply any others that are used by your *expr* script.

–m *file* specifies a file whose header is to be merged with that of the first input image to form the header of the output image.

–o *file* directs the output General Image to the named file, the default being the standard output stream *stdout*.

Remaining arguments in the *iha* command line are the script itself (only if no **–a** file was specified) followed by the input file name(s). If file names are omitted entirely, the standard input stream *stdin* is read. To read *stdin* along with other files, use "**–**" as a file name.

FILES

/tmp/iha∗.c	temporary script source file
/tmp/iha∗	temporary compiled script command
/usr/gips/tables/cprofile	image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihm(1)

DIAGNOSTICS

When an error is detected, *iha* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- bad dscale=p for *file*
- can't compile
- can't execute
- cannot create pipe
- compiler error: *text*
- different xnum: *file*
- different ynum: *file*
- dscale=f unsupported: *file*
- expr missing*file*
- *file*: bad dtype
- not an assignment: *file* or *expr*
- open error on pipe
- write error to pipe

BUGS

The output image cannot be defined as *dscale=p*, and no image may be defined as *dscale=f*.

The syntax is not close enough to *awk*.

Bizarre error messages occur when running *iha* on Sun–3 systems if the value of the FLOAT_OPTION environment variable differs from that specified when the GIPS library was compiled.

NAME

ihbidr – access a Magellan BIDR as a GIPS cellular image

SYNOPSIS

ihbidr [**–D**] *file*

DESCRIPTION

The *ihbidr* constructs a GIPS header to describe Magellan BIDR image *file* so that the *gcell*(3) routines can access it as a GIPS cellular image. *file* must be located on a disk—it cannot be specified as a pipe. The header may be read by any GIPS command that accepts cellular images, e.g. *gipstool*(1), *ihgrid*(1). or *ihgeom*(1).

If a file named *file*.aux is found in the same directory as *file*, it is assumed to contain an index file created by the *bidrindx* program, and is used to initialize tables that assist the *gcell* routines in locating particular BIDR data blocks. Otherwise, *gcell* must read through the BIDR sequentially until the requested block is located.

–D instructs the *gcell* routines to write the following information information to the standard error stream, *stderr*, whenever a new BIDR block is accessed:

- latitude of first pixel in block
- longitude of first pixel in block
- x-offset of first pixel relative to image origin
- y-offset of first pixel relative to image origin
- file offset of first pixel in block

FILES

/usr/gips/tables/cprofile image header definition file

SEE ALSO

gcell(3)

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

- cannot read *file*
- missing file name
- multiple file names: *args*
- unknown option: *text*

NAME

ihbox – apply a boxcar convolution filter to a General Image

SYNOPSIS

ihbox [**–nu**] [**–mo** *file*] [**–0** *dzero*] [**–1** *dfact*] *array* [*name*]

DESCRIPTION

The *ihbox* command reads a General Image from file *name*, and applies a spatial filter defined by the contents of the file specified by the (required) *array* argument. If *name* is omitted, *ihbox* reads the standard input stream *stdin*.

Array specifies a file containing an $x \times y$ array of fixed or floating point numbers in ASCII coded format, beginning with a pair of integers specifying the column and row dimensions of the array (both of which must be odd and greater than 1) and continuing with the numeric array elements themselves in row-major order (all of row 1, followed by all of row 2, etc). Repeated elements of the same value may be denoted by an integer repeat count, followed by an asterisk, followed by the value to be assigned that many times, e.g. "*32*20.5*".

Ihbox first looks for *array* in the current directory, then in "/*usr*/*gips*/*lib*/*boxcar*".

Alternatively, if *array* contains any whitespace characters (blanks, tabs, or newlines), *ihbox* interprets it as a string containing the array dimensions and element values themselves. This string format is particularly useful when *ihbox* is invoked within a script.

If the input General Image contains "integer" pixels (i.e. *dtype = b, u, s, h, i,* or *l*), the array elements will, if necessary, be converted to signed integers. *ihbox* then transforms the image by replacing each pixel with the convolution of the user-defined array with the pixel (and its neighbors), using signed integer or floating-point arithmetic (depending on *dtype*). If header variable *dnull* is defined, pixels with that value will not be included in the sum, although they may be replaced by the filtering. If *xscale=w*, the convolution wraps around the left- and right-hand map extremities. This feature may be used when filtering images in which the x-axis represents an azimuthal variable, e.g. longitude.

When 8-bit or unsigned integer pixels are filtered, it is quite likely that the resulting values cannot be represented correctly (e.g. a low-pass filter applied to a "*dtype=u*" image may generate negative values, which cannot be represented by that pixel format). To avoid this problem, if the input image is in "*dscale=n*" format, new values of *dzero* and *dfact* may be specified for the output image by means of the **–0** and **–1** options, respectively. The pixels will then be transformed to floating point values before filtering, using the original *dzero* and *dfact*, and back to *dtype* before they are written out, using the new *dzero* and *dfact*.

–o *file* directs the output General Image stream to *file*. If omitted, the output will be written to the standard output stream *stdout*.

–m *file* specifies a second General Image file whose header is to be merged with that of the input image before filtering.

–n directs *ihbox* to filter *only* those pixels with value equal to the header variable *dnull*. Otherwise, all pixels are filtered.

–u no normalization is performed on the convolution-sum output. Otherwise, each output pixel is normalized by the sum of the absolute values of the contributing *array* elements.

–0 *val*
–1 *val* specify new values of *dzero* and *dfact* to apply to output pixels of *dscale=n* images (see above).

FILES

/usr/gips/tables/cprofile	image header definition table
/usr/gips/lib/boxcar/	directory containing specimen arrays

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihc(1), ihread(3)

DIAGNOSTICS

When an error is detected, *ihbox* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image handling routines are described in *ihread*(3). Other possible messages are:

- multiple input files: *file*
- missing array file
- bad array limits: *file*

BUGS

Output from *ihfft*(1) (signified by header variable *dtype=c*) and all images in polynomial format (*dscale=p*) cannot be handled by *ihbox*.

NAME

ihc – create a General Image

SYNOPSIS

ihc [**–** *size]* [**–bfirw**] [**–d** *dir*] [**–hmo** *file*] [**–s** *bytes*] [*name*]

DESCRIPTION

The *ihc* command combines an ASCII header and a binary image array or pixel stream to produce a General Image that can be processed by subsequent GIPS routines. If *name* is specified, the binary array or pixel stream is read from a file of that name; otherwise, it is read from the standard input stream *stdin*.

–*size* specifies the (maximum) physical block size of the binary input file. This should always be specified when input comes directly from magnetic tape. The default is 1024 bytes (which would be specified as "*–1024*").

–h *file* specifies the name of the input header file that will describe the output General Image. If a relative pathname is used, *ihc* will search for it in the current directory, then in "**/usr/gips/tables/**". The format of image headers is described in *ihread*(3).

–m *file* specifies a secondary header file whose contents are to be merged with the header specified by the **–h** option. The same search rules apply.

–o *file* specifies the destination of the output General Image. If omitted, it will be written to the standard output stream *stdout*.

–i inverts the input array, i.e. from top to bottom.

–r reverses the input array, i.e. from left to right.

–b byte-swaps the input array (interchanging consecutive pairs of bytes).

–s *n* skips (ignores) *n* bytes at the beginning of the input stream.

–w specifies that the input stream contains pixel values in random order. The input consists of sets of 3 binary fields: a 4-byte integer representing the *x*-coordinate, a 4-byte integer specifying the *y*-coordinate, and a 4-byte floating-point field specifying the pixel value itself.

–f has the same effect as **–w**, but the pixels that are read from *stdin* are in ASCII format: pairs of (optionally signed) integers, each followed by a floating-point pixel value.

–d *dir* overrides the default directory "*/tmp*" in which any necessary scratch files will be written. For example, it is necessary to create a temporary file if the input is from *stdin* and the array is to be inverted, or if from tape and the specified block size is not a multiple of the raster length, or if the **–w** or **–f** flags are used.

FILES

/tmp/ihc*	default spool file
/usr/gips/tables/cprofile	image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

gin(1), ihs(1), ihm(1), ihread(3)

DIAGNOSTICS

When an error is detected, *ihc* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- multiple input files
- odd raster length - can't byte swap
- unexpected EOF: *file*

NAME

ihfft – Fourier transform each raster of a General Image

SYNOPSIS

ihfft [**–i**] [**–mo** *file*] [*name*]

DESCRIPTION

The *ihfft* command reads a General Image from file *name* (or, if omitted, from the standard input stream *stdin*), and applies a one-dimensional "fast" Fourier transform to each raster – i.e. each line of pixels parallel to the x-axis.

The output image, written to the file specified by the **–o** *file* option (default output to the standard output stream *stdout*), contains pairs of 4-byte floating-point pixels, representing respectively, the cosine and sine components of the transform. If the number of pixels in the raster is not an exact power of 2, the raster is extended: if "*xscale=w*", the raster is "wrapped" from end to end before the transform is taken; otherwise it is extended with zeroes. The number of original input pixels per raster is saved in header variable *xlfft*.

Use *ihrot*(1) to apply *ihfft* to columns of the transformed image, or *ihm*(1) or *iha*(1) to "edit" images in Fourier space. When a complex image is passed through *ihs*(1), each sine-cosine pixel pair is replaced by its modulus – sqrt(sin**2 + cos**2).

–m *file* specifies a second General Image file whose header is to be merged with that of the input image before performing the transform.

–o *file* overrides the destination of the output General Image, which, by default, goes the the standard output stream *stdout*.

–i directs *ihfft* to take the inverse transform. The number of (complex) output pixels will be determined by the value of header variable *xlfft*. When performing a double Fourier transform, use "*ihrot*" to rotate the image for the second forward transform, and "*ihrot –ri*" to rotate it back again before the second reverse transform.

FILES

/usr/gips/tables/cprofile image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihrot(1), ihs(1), ihread(3)

DIAGNOSTICS

When an error is detected, *ihfft* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- bad arg: *name*

BUGS

Images in polynomial format (*dscale=p*) cannot be transformed.

The Fourier transform code is optimized for the VAX, not for the MC68881.

NAME

ihfromfits – convert a FITS image into GIPS format

SYNOPSIS

ihfromfits [**–bh**] [*name*]

DESCRIPTION

Ihfromfits is a filter that converts an input image file in the Flexible Image Transport System (FITS) format into the GIPS General Image Processing System format, suitable for subsequent pipeline processing and display on a suitable image frame buffer.

–b causes the binary input image to be "*byte-swapped*", i.e. adjacent pairs of bytes are interchanged in the binary image array.

–h causes all **HISTORY** entries in the input header to be retained in the output GIPS header. If omitted, only a single history entry will be generated, composed of a concatenation of the **OBJECT, DATE, DATE-OBS, ORIGIN, INSTRUME,** and **OBSERVER** fields of the input FITS header.

The *ihfromfits* command reads from the named file, or, if omitted, from the standard input *stdin*, and writes its output to the standard output *stdout*. It generates an output header by using the contents of file **FITSinit** as a skeleton. If this file is not found in the current directory, it is read from **/usr/gips/tables/**. This header is then updated and augmented with information from the header of the FITS file. The following FITS header variables are recognized, with the restrictions as noted. Variables not contained in this list are ignored.

END	must be present to denote the end of the FITS header.
SIMPLE	must be present and have the value **T**.
BITPIX	must be present and have the value 8, 16, or 32, corresponding to unsigned bytes, signed 16-byte integers and signed 32-byte integers, respectively. There is no provision for handling either unsigned integers or floating-point pixel fields.
NAXIS	must be present and have the value 2.
NAXIS*n*	Both **NAXIS1** and **NAXIS2** must be present. The output image header will specify pixel limits as *xnum*=**NAXIS1**, *ynum*=**NAXIS2**.
CRVAL*n*	Both **CRVAL1** and **CRVAL2** must be present. They are used, along with **CRPIX***n* and **CDELT***n* to generate header variables *x*/*ymin* and *x*/*ymax*.
CRPIX*n*	Both **CRPIX1** and **CRPIX2** must be present.
CDELT*n*	Both **CDELT1** and **CDELT2** must be present.
CROTA*n*	Both **CROTA1** and **CROTA2** must be present.
CTYPE*n*	The 8-byte FITS mnemonics are, where possible, translated into longer, more descriptive titles, in the manner described below, and are written to header variables *x*/*ytitle*.
DATAMAX	This field is copied to the output variable *dmax*.
DATAMIN	This field is copied to the output variable *dmin*.
BSCALE	If both **BSCALE** and **BZERO** are present, they are copied to header variables *dfact* and *dzero*, and *dscale* is given the value **n**. Otherwise, *dscale* is set to **s**.
BZERO	See **BSCALE** above.
BUNIT	The 8-byte FITS mnemonic is, where possible, translated into a longer, more descriptive title, in the manner described below, and is passed to header variable *dtitle*.
BLANK	This field is passed to header variable *dnull*.
OBJECT	This is copied to the header variable *title*.
DATE	This is copied to the header variable *date*.
DATE-OBS	The character-string value is copied to a *history* entry.
ORIGIN	The character-string value is copied to a *history* entry.
INSTRUME	The character-string value is copied to a *history* entry.
OBSERVER	The character-string value is copied to a *history* entry.
HISTORY	Provided the **–h** flag is set, all HISTORY fields in the FITS header are copied to separate *history* entries in the output header.

In addition, the FITS variables that specify unit-types are translated into more descriptive titles, according to the following scheme:

K	Kelvins
JY/BEAM	Jy per beam area
JY/PIX	Jy per pixel
MAG/PIX	Magnitudes per pixel
M/SEC	Meters per second
DEGREES	Angle in degrees
RA	Right Ascension (deg)
DEC	Declination (deg)
LL	Tangent plane E-W (deg)
MM	Tangent plane N-S (deg)
GLON	Galactic longitude (deg)
GLAT	Galactic latitude (deg)
ELON	Ecliptic longitude (deg)
ELAT	Ecliptic latitude (deg)
TIME	Time (sec)
FREQ	Frequency (Hz)
LAMBDA	Wavelength (meters)
VELO	Velocity (m/sec)
VELO-LSR	Velocity wrt local rest
VELO-HEL	Velocity wrt Sun
VELO-OBS	Velocity wrt observer
PIXEL	Pixel
STOKES	Stokes polarization
COMPLEX	Complex

FILES

/usr/gips/tables/FITSinit	skeleton image header
/usr/gips/tables/cprofile	header definition table

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihtofits(1), ihs(1), ihread(3)

FITS is a product of the data processing sections of Kitt Peak National Observatory and the National Radio Astronomy Observatory, and is described in a paper presented at the "International Workshop on Image Processing in Astronomy", held in June 1979 at the International Center for Theoretical Physics in Miramare, Trieste, Italy. More information about the format can be obtained from

Dr. Eric W. Greisen
National Radio Astronomy Observatory
Edgemont Road
Charlottesville, VA 22901

DIAGNOSTICS

When an error is detected, *ihfromfits* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- multiple input files: *file*
- EOF while reading FITS header: *file*
- variables missing from FITS header: *item*
- error while reformatting *file*
- This is not a SIMPLE image

- NAXES not .GE. 2: *item*
- Unsupported BITPIX: *item*
- EOF before end of image: *file*

NAME

ihfromgif – translate a GIF image file to GIPS format

SYNOPSIS

ihfromgif [**–Sv**] [**–amo** *file*] [*name*]

DESCRIPTION

The *ihfromgif* command reads an image in Compuserve's GIF format and writes it out to the standard output, or to the file specified by the **–o** flag, in GIPS image format. This operation is essentially the reverse of that performed by the *ihtogif*(1) command.

–S writes processing statistics to *stderr*.

–a *file* specifies where to write any colormap information present in the input image. See the manual entry for *gipstool*(1) for the format of a *colormap* file.

–o *file* overrides the default destination for the output GIPS image (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the output image.

–v invokes verbose mode, writing statistics about the input image to the standard error stream, *stdout*.

FILES

/usr/gips/tables/cprofile image header definition file

SEE ALSO

ihtogif(1)

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

When an error is detected, *ihfromgif* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- GIF buffer overflow
- bad GIF header
- corrupt GIF file - bad end-of-header
- insufficient memory for output buffer
- missing GIF image separator
- multiple input files
- unexpected EOF in file header
- unexpected EOF in image header
- unexpected end of image

NAME

ihfrompix – translate a Sun rasterfile to GIPS image format

SYNOPSIS

ihfrompix [–S**]** [**–amo** *file*] [*name*]

DESCRIPTION

The *ihfrompix* command reads a Sun *rasterfile* image and writes it out to the standard output, or to the file specified by the **–o** flag, in GIPS image format. This operation is essentially the reverse of that performed by the *ihtopix*(1) command.

–S writes processing statistics to *stderr*.

–a *file* specifies where to write any colormap information present in the input image. See the manual entry for *gipstool*(1) for the format of a *colormap* file.

–o *file* overrides the default destination for the output GIPS image (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the output image.

FILES

/usr/gips/tables/cprofile image header definition file

SEE ALSO

gipstool(1), ihfrompix(1), rasfilter8to1(1), rasterfile(5), rastrepl(1)

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

When an error is detected, *ihfrompix* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- multiple input files

NAME

ihgeom – transform random-access General Image via a control grid

SYNOPSIS

ihgeom [*–buffers*] [**–DSfnx**] [**–mo** *file*] *file* [*grid*]

DESCRIPTION

The input control grid file *grid* (or, if omitted, the standard input stream *stdin*), is used to apply a transformation to a "random-access" General Image file, and the resulting General Image is written to the file specified by the **–o** option (or, if omitted, to the standard output stream *stdout*).

The *file* argument is required. It is the name of the input random-access General Image, which must be a disk file that has been created from a normal General Image by the *ihgfmt*(1) command.

–buffer specifies the number of image cells to be allocated. Access to the input image is optimized by storing several such cells in an in-core ring buffer; by default, 8 are stored, but this number may be altered, viz: "*ihgeom –32*" will use 32 buffers (and, therefore, more data memory).

–f specifies that the input *grid* file is written in formatted ASCII code, rather than in binary integers and floats.

–n suppresses the output image. It may be used in conjunction with the **–D** flag to verify or display an input control grid stream.

–x each output pixel is generated by averaging the four nearest-neighbors of the required location in the input image; otherwise, only the nearest pixel value is used.

–m *file* specifies the name of a file containing a General Image header whose contents are to be merged with the input random-access General Image before the transformation is attempted.

–D instructs *ihgeom* to write a one-line description of the input and output coordinates of each control grid cell to the standard error stream *stderr*.

–S causes *ihgeom* to write UNIX usage statistics to *stderr* when it has completed its task.

CONTROL GRID FORMAT

The control grid file specifies the transformation by breaking up the output image into *blocks* of rasters, and further sub-dividing each block into a series of rectangular *cells*. In general, the shape of a region of input image that corresponds to a rectangular output cell will not itself be rectangular. *ihgeom* assumes that the cells have been chosen small enough that, to find the input pixel coordinate that corresponds to an output pixel within a given cell, it is sufficient to perform a bi-linear interpolation using the coordinates of the *corners* of the input and output cells.

The control grid may be supplied either in binary or ASCII format, the latter being indicated by the **–f** flag. In binary format, each input field is either a *long* integer (i.e. INTEGER∗4), or a *float* (REAL∗4). *Ihgrid*(1) always generates a binary control grid.

The control grid begins with two lines of ASCII characters, each ending in a linefeed character. The first line specifies the number of columns and rows (rasters) to be assigned to the output General Image, and the number of rasters to be allocated to the *ihgeom* output buffer. The number of columns and rows are also assigned to variables *xnum* and *ynum* respectively, in the output image header. This line may be decoded by *sscanf()* using the format string GIPS_GRID defined in "/*usr*/*gips*/*include*/*gips.h*". The second line defines the parameters of the transformation; its *sscanf()* format is defined by GIPS_MAPPING in the same file. The second line will become the value of the *mapping* variable in the header of the output image created by *ihgeom*.

The remainder of the control grid file contains sets of 12 numbers: 4 integers followed by 8 floats. The integers define the top-left (*x*,*y*) and bottom-right (*x*,*y*) pixel coordinates of a rectangular cell in the output image. The 8 floats specify the (*x*,*y*) pixel coordinates of the corners of the corresponding cell in the input image, in the order: *top-left*, *top-right*, *bottom-left*, and *bottom-right*.

The output image will be generated as a set of blocks, each containing a fixed number of *y*-coordinate rasters (the number specified by the third header field), except the last block, which may, of course, be shorter. The control grid cells must refer to the blocks according to increasing *y* value, but, within each block, they may fill at random. A cell may not overlap output blocks.

Here is a simple example of a single output block and a single control grid cell. If the **–f** flag is specified, this would represent a perfectly legal *ihgrid* input stream of ASCII digits and (ignored) blanks, tabs, and newline characters. Decimal points on the floats are optional if the field has no fractional part:

```
Gips control grid: xnum=512 ynum=512 rasters=512
name=Tiepoint type=0 origin=0,0
    0 0
        512 512
            300.    100.        800.    250.
            200.    390.        900.    450.
```

This control grid will generate a 512×512 image from a segment of an input image whose (*x*,*y*) coordinates are (300,100), (800,250), (200,390), and (900,450). In this case, the control grid defines a single rectangular cell. Note that the bottom-right output coordinates (512 and 512 in this example) are one more than the pixel address of the corresponding bottom-right output pixel, viz. (511,511). A more realistic example might be the following:

```
Gips control grid: xnum=2000 ynum=1500 rasters=750
name=Tiepoint type=0 origin=0,0
    0 0
        1000 750
            6.      0.          1081.   21.
            42.     673.        983.    814.
    1000 0
        2000    750
            1081.   21.         1932.   17.
            983.    814.        1811.   791.
    0 750
        1000 1500
            42.     673.        983.    814.
            80.     1405.       1527.   695.
    1000 750
        2000 1500
            983.    814.        1811.   791.
            1527.   695.        1928.   1527.
```

This control grid is composed of four cells and two output blocks: the first pair of cells refers to block 1. Cell 1 specifies that the output rectangle with coordinates (0,0), (1000,0), (0,750), (1000,750) is to be linearly interpolated from input image coordinates (6,0), (1081,21), (42,673), (983,814), respectively. The output pixels will fill the box from *x*=0 through *x*=999 and *y*=0 through *y*=749. The second cell maps the output rectangle (1000,0), (2000,0), (1000,750), (2000,750) from input coordinates (1081,21), (1932,17), (983,814), (1811,791). These first two cells define the mapping into the first 750-raster-long output buffer. The remaining pair of cells defines the mapping into the second buffer.

FILES

/usr/gips/tables/cprofile image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihgfmt(1), ihgrid(1), ihread(3)

DIAGNOSTICS

When an error is detected, *ihgeom* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- bad arg: *arg*
- bad geometry: *list*
- bad *x*/*y*-coordinate in cell: *n*
- cell spans output buffers: *n*
- input not generated by ihgfmt: *file*
- no input image file specified

BUGS

The Control Grid file format is not "compact" – it is, in fact, about twice as long as it needs to be for most transformations.

NAME

ihgfmt – create a random-access General Image

SYNOPSIS

ihgfmt [*–size*] [**–NS**] [**–d** *dir*] [**–mo** *file*] [*name*]

DESCRIPTION

The input General Image file *name* (or, if omitted, the standard input stream *stdin*), is reformatted and written to the file named by the **–o** option (or, if omitted, to the standard output stream *stdout*). The output image is reformatted into cells whose size in bytes will be a power of 2.

The reformatted General Image files, which contain special headers, are suitable for subsequent processing by the *ihgeom* command. They are distinguished from "normal" General Images by the special value "*G*" assigned to header variable *dscale*, which prevents these images from being processed by most other GIPS commands.

–*size* specifies the cell size of the output image. *size* must be a power of 2, i.e. "*ihgfmt –4096*" will write 4096–byte cells. The default cell size is 1024 bytes, which is chosen to be the same as the physical blocksize of many UNIX disk files.

–m *file* specifies the name of a file containing a General Image header whose contents are to be merged with the input General Image before reformatting is attempted.

–N begins writing the output image cells immediately after the output header. Normally, *ihgfmt* adds null bytes between the header and the first output cell so that each cell begins at a byte-offset from the start of the file that is a multiple of the cell size itself. Note that most GIPS commands cannot read an image that has been generated with the **–N** flag of *ihgfmt*.

–S causes *ihgfmt* to write UNIX usage statistics to *stderr* when it has completed its task.

–d *dir* overrides the name of the directory (by default, "/*tmp*") into which *ihgfmt* will write a temporary scratch file when the input General Image is read from the standard input stream *stdin*.

FILES

/tmp/ihg∗	default scratch file
/usr/gips/tables/cprofile	image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihgeom(1), ihgrid(1), gcell(3), ihread(3)

DIAGNOSTICS

When an error is detected, *ihgfmt* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- bad arg: *arg*
- bad *–size* specification

NAME

ihgrid – generate control grid for General Image transformation

SYNOPSIS

ihgrid [*–nproj*] [**–DSf**] [**–C** *long,lat*] [**–l** *long,lat,rho*] [**–Pp** *ang*] [**–XY** *org*] [**–r** *rad*] [**–R** *fact*] [**–bnxy** *num*] [**–mo** *file*] [*name*]

DESCRIPTION

A control grid is generated for transforming a random-access General Image into a given cartographic coordinate system. Consult *ihgeom*(1) for a discussion of control grids.

–nproj specifies the desired transformation:

0	**Tie-Point**	Polynomial	Bi-variant
1	**Gnomonic**	Azimuthal	r=tan(d)
2	**Stereographic**	Azimuthal	r=2tan(d/2)
3	**Orthographic**	Azimuthal	r=sin(d)
4	**Equidistant**	Azimuthal	r=d
5	**Equivalent**	Azimuthal	r=2sin(d/2)
6	**Lambert**	Conformal	One parallel
7	**Lambert**	Conformal	Two parallels
8	**Mercator**	Conformal	Equatorial
9	**Polar**	Conformal	Stereographic
10	**Mollweide**	Equivalent	Non-conic
11	**Albers**	Equivalent	One parallel
12	**Albers**	Equivalent	Two parallels
13	**Lambert**	Equivalent	Cylindrical
14	**Lambert**	Equivalent	Polar azimuthal
15	**Bonne**	Equivalent	Pseudo-conical
16	**Sinusoidal**	Equivalent	
17	**Werner**	Equivalent	
18	User defined projection via "**user()**"		
19	**Aitoff**	Equivalent	Non-conical
20	**Hammer**	Equivalent	Non-conical
21	**Briesemeister**	Equivalent	Non-conical
22	**Cylindrical**	Equivalent	Non-conical

When a cartographic transformation is specified (types 1 through 21), *ihgrid* expects that the realworld *x*- and *y*- coordinates of the input image are defined in *degrees* of, respectively, longitude and latitude. Also, if the input longitude extends through the full 360 degrees, the input image header should specify that "*dtype=w*", i.e. that the coordinate system *wraps*.

–D generates "debug" output on *stderr*, detailing set-up parameters.

–P *ang* specifies the second standard parallel (in degrees) for Lambert Conformal (–7) and Albers Equivalent (–12) transformations.

–R *fact* multiplies the default transform scaling factor by the floating-point quantity *fact*, thereby "fine tuning" the size of the output image. Alternatively, the user may supply a scaling factor via the **–r** option (qv.) This option is ignored for tiepoint transformations (–0).

–S writes a line of UNIX "statistics" to *stderr* upon command completion.

–X *org*
–Y *org* specify, respectively, the *x* and *y* origins of the transformed image in the output pixel space. This is, typically, the point into which the input origin defined by the **–l** option is mapped, although this is not true for all transformations. The origin may also be specified by the **–C** option. These options are ignored for tiepoint transformations (–0).

–b *num* specifies the number of buffer rasters that *ihgeom* will require to transform an image via the

control grid that is to be created. This information must be supplied to *ihgrid* and is passed to *ihgeom* in the output stream header. The default value is 32. The larger the value, the more efficient the transformation and the smaller the control grid file, but *ihgeom* will use more memory. The size of the actual output buffer allocated by *ihgeom* will be this number times the length of each raster.

–f generates formatted ASCII output. Use the **–f** flag of *ihgeom*(1) to read it.

–C *long,lat* specifies the longitude and latitude origin of the transformed image in the output pixel space. This is, typically, the point into which the input origin defined by the **–l** option is mapped, although this is not true for all transformations. The origin may also be specified by the **–X** and **–Y** options. This option is ignored for tiepoint transformations (–0).

–l *long,lat,rho*
defines the Euler angles (in degrees) of a rotation to be applied to the (longitude,latitude) coordinate system before the Cartographic transformation is applied. The origin of longitude will become *long*, the origin of latitude will become *lat*, and the resulting figure will be rotated clockwise by *rho* . If unspecified, these angles will be set to zero. This option will be ignored for tiepoint transformations.

–m *file* specifies the name of another General Image file whose header is to be merged with that read from *name* or *stdin*.

–n *order* is used only with tie-point transformations, and specifies the order of bi-linear polynomials to be used.

–o *file* the control grid will be written to *file*, rather than to the standard output stream *stdout*.

–p *ang* specifies the central latitude (or first standard parallel) of the transformed image (in degrees). If this option is omitted, *ihgrid* uses the arithmetic mean of the realworld *y*-axis limits of the input General Image. It is ignored for tiepoint transformations (–0).

–r *rad* specifies the transform scaling factor – whose meaning depends on the particular non-tiepoint transformation selected. If not specified, *ihgrid* will compute a value that attempts to maximize the area of input image mapped into the output image. Some clipping and/or bordering is almost guaranteed! Use the **–R** option for fine tuning.

–x *num*
–y *num* specify, respectively, the x- and y-pixel extents of the output image to be constructed via the control grid.

name specifies the name of the General Image file to be transformed. If omitted, *ihgrid* expects to read the image (in fact, its header only) from the standard input stream "*stdin*".

TIE POINTS

When *ihgrid* performs a transformation via tie-points (–0 flag), the points are read from the standard input stream *stdin* in groups of 4 ASCII floating-point numbers. If *name* is omitted, *ihgrid* will also expect to read a General Image header from *stdin* before reading the tie points. Non-ASCII bytes in the input stream will cause *ihgrid* to quit. The numbers, separated by any number of spaces, tabs, or newlines, must be in the following order:

- Pixel x-coordinate of point in input image,
- Pixel y-coordinate of point in input image,
- Pixel x-coordinate of corresponding point in output image, and
- Pixel y-coordinate of corresponding point in output image.

Sufficient tie points must be specified so that the polynomial mapping developed by *ihgrid* is not singular. This is generally assured provided the points are not co-linear, and there must be at least (*num*+1)×(*num*+1) of them, where *num* represents the order of bi-linear mapping polynomial specified via the **–n** option.

USER-DEFINED PROJECTION

The user may generate a "customized" version of *ihgrid* by writing a C routine, named *user()* in "user.c", and compiling it via the command:

```
cc /usr/gips/lib/ihgrid.o user.c –o ./ihgrid –lGI –lm
```

Here is an example of a *user()* routine:

```
user(sw,x,y,xy)
        double  x, y, xy[2];
{
        switch (sw) {
        case 0:
                /* Initialization call. Precalculate what you want,  */
                /* using x = –p value, y = –P value, and xy[0] = –r value. */
                /* If –r flag not specified, user() will be called twice with sw=0*/
                /*
                if /* initialization ok */
                        return(0);
                else
                        return(1);
        case 1:
                /* Transform realworld xy[0] and xy[1]
                   to pixel coordinates (x,y) */
                xy[0] = ...;
                xy[1] = ...;
                if (/* transformation non-singular */)
                        return(0);
                else
                        return(1);
        case 2:
                /* Transform pixel coordinates (x,y)
                   to realworld xy[0] and xy[1] */
                xy[0] = ...;
                xy[1] = ...;
                if (/* xy[0] and xy[1] are within input image */)
                        return(0);
                else
                        return(1);
        default:
                break;
        }
        /* illegal sw value (or any other abort situation) */
        return(–1);
}
```

FILES

/usr/gips/lib/ihgrid.o	partially linked *ihgrid* for user-defined projection
/usr/gips/tables/cprofile	image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihgeom(1), ihgfmt(1), gtrans(3)

Ford, P., *Guide to GIPS*, MIT Center for Space Research, 1984.

Richardus, P. and Adler, R., *Map Projections*, North Holland/American Elsevier, 1972.

Cogley, J.G., *Map Projections with Freely Variable Aspect*, EOS, **65** 34, p.481 (1984).

DIAGNOSTICS

When an error is detected, *ihgrid* halts with system code 1, writing a message to the standard error stream *stderr*. Possible error messages are

- bad –n value: *value*
- bad arg: *value*
- forward mapping for *projection* not implemented
- illegal numeric in input stream: *code*
- inverse mapping for *projection* not implemented
- mapping is singular for this origin choice
- no –P value specified
- setup for *projection* not implemented
- singular tie-point transformation
- unable to compute –r value

BUGS

The Sun–3 version of *user()* must be compiled and linked with the same value of the FLOAT_OPTION environment variable as was in effect when *ihgrid* and the GIPS library were created.

NAME

ihm – merge and/or resample one or more General Images

SYNOPSIS

ihm [**–q**] [**–d** *dir*] [**–hmo** *file*] [**–l** *oper*] [*name*...]

DESCRIPTION

The *ihm* command reads one or more input General Images and generates an output General Image that represents the merging or resampling of the input according to specifications in the output header.

The merge is performed by a weighted average over each input pixel that overlaps a given output pixel. A faster, less accurate merge may be achieved by specifying the **–q** flag, in which case each output pixel is given the value of the first non-null input pixel that overlaps it. Alternatively, the **–l** option lets you combine images according to a set of Boolean operators.

–d *dir* overrides "/*tmp*" as the directory in which to allocate temporary files.

–h *file* specifies an existing General Image file whose header is to form the basis for the output image description. Otherwise, the header of the first input file is used.

–l *oper* indicates that the image pixels are to be treated as unsigned logical quantities and to be combined by Boolean operations. *oper* is a list of operators and subfields, one per input General Image, specifying how each input pixel from that particular image is to be handled. This option overrides **–q** and is invalid for floating-point input or output pixels. Valid subfields in *oper* begin with one of the following symbols

= assign pixel value to output field.
| inclusively **OR** pixel to output field.
& logically **AND** pixel to output field.
^ exclusively **OR** pixel to output field.

The symbol may be followed by the "**~**" character, indicating that the input pixel is to be complemented before the operation. Four optional numeric fields may follow, separated by commas: the offset and length (in bits) of the input field, and the offset and length (in bits) of the output field. Offsets are counted from the most significant bit. For example, the command

ihm –l "=3,5,0,5 =~5,3,5,3" gi.1 gi.2

produces a General Image in which each pixel contains the last 5 bits of a pixel from "gi.1" concatenated to the complement of the last 3 bits of each corresponding pixel in "gi.2". In general, *oper* should contain as many sub-fields as there are input General Image files, but if fewer sub-fields are specified, the remaining files are processed according to the last sub-field specified.

Bit-field and arithmetic operations of arbitrary complexity may also be performed by the *iha*(1) command, which is limited, however, to input and output images containing the same number of *x* and *y* pixels.

–m *file* specifies a second file whose header is to be merged with that specified by the **–h** option or with that supplied by the first input file.

–o *file* directs the output General Image to the named file, the default being the standard output stream *stdout*.

The remaining arguments in the *ihm* command line are the input file names themselves. If omitted entirely, the standard input stream *stdin* is read. To merge *stdin* with other files, use "**–**" as a file name.

ihm attempts to process all input files synchronously, and always writes the output in increasing y-raster order. If this requires that an input file be read in a non-sequential manner, and if it is not being read directly from disk (e.g. it is a pipe or a tape file), *ihm* will copy that input stream to disk before proceeding. This will increase execution time and generate a warning message on the standard error stream *stderr*.

FILES

/tmp/ihm*n	default temporary spooled images
/usr/gips/tables/cprofile	image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

iha(1), ihc(1), ihread(3)

DIAGNOSTICS

When an error is detected, *ihm* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- –q flag illegal for dtype=p input: *file*
- bad –l field offset: *n*
- bad –l value: *oper*
- bad dtype for –l option: *file*
- illegal change of dtype: *file*
- inconsistent wrap limits: *file*
- spool sequence error: *file*
- too many –l fields: *oper*

BUGS

The **–q** option is sometimes disregarded, e.g. when *dscale=f* or when *dscale=n* and the input and output values of *dzero* and *dfact* are not identical.

The format of the **–l** option used to specify Boolean bit-field operations is a poor hack.

NAME

ihmed – apply a median filter to a General Image

SYNOPSIS

ihmed [**–S**] [**–mo** *file*] [**–xy** *dim*] [*name*]

DESCRIPTION

The *ihmed* command reads a General Image from file *name*, and applies a median filter defined by the **–x**, and **–y** options. If *name* is omitted, *ihmed* reads the standard input stream *stdin*. Each pixel is replaced with the median value of the pixels in a rectangular box surrounding it, with dimensions specified by the **–x** and **–y** options. Null pixels, i.e. those with the same value as the *dnull* header variable, are ignored when computing the median.

If the input General Image contains "integer" pixels (i.e. *dtype = b, u, s, h, i,* or *l*), the calculation will be performed in fixed-point arithmetic. Otherwise, floating-point arithmetic will be used. If *xscale=w*, the convolution wraps around the left- and right-hand map extremities. This feature may be used when filtering images in which the x-axis represents an azimuthal variable, e.g. longitude.

–S write processing statistics to *stderr*.

–m *file* specifies a second General Image file whose header is to be merged with that of the input image before filtering.

–o *file* directs the output General Image stream to *file*. If omitted, the output will be written to the standard output stream *stdout*.

–x *num* the number of x-pixels to be averaged. The default is 3.

–y *num* the number of y-pixels to be averaged. The default is 3.

FILES

/usr/gips/tables/cprofile image header definition table

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihbox(1), ihread(3)

DIAGNOSTICS

When an error is detected, *ihbox* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image handling routines are described in *ihread*(3). Other possible messages are:

- multiple input files: *file*

NAME

ihnull – copy a General Image, possibly updating header fields

SYNOPSIS

ihnull [**–S** *file*] [**–amo** *file*] [**–n** *factor*] [*name*]

DESCRIPTION

The *ihnull* command reads a General Image from file *name*, updates the header from an file specified by the **–m** option, and writes it out to the standard output, or to the file specified by the **–o** flag, without altering the pixels in any way. If *name* is in cellular format, i.e. it has been created by the *ihgfmt*(1) program, *ihnull* will turn it into a "flat" image that can be piped into other GIPS commands.

–S *file* writes processing statistics to *stderr*.

–a *file* writes an additional copy of the output file to *file*.

–n *fact* shrinks the output image by a positive integer factor, *fact*. If the input image header contains a *mapping* variable, *ihnull* will also reduce the *origin* and *radius* fields by an amount equal to *fact*, thereby preserving the mapping information for most projections.

–o *file* overrides the default output destination (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the input image.

FILES

/usr/gips/tables/cprofile image header definition file

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

When an error is detected, *ihnull* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- multiple input files
- bad –n value
- unexpected EOF
- bad cell format

NAME

ihpds – translate a Planetary Data System image to GIPS format

SYNOPSIS

ihpds [**–hlvS**] [**–mo** *file*] [*name*]

DESCRIPTION

The *ihpds* command reads a PDS image label file, locates the image, and writes it out to the standard output, or to the file specified by the **–o** flag, in GIPS image format.

–h write only the image header to the standard output, removing carriage returns and adding newlines where necessary.

–l instead of converting the image itself, write a list of image parameters to the standard output.

–v while converting the image, write the name and dimension of the image file to the standard error stream, *stderr*.

–S write processing statistics to *stderr*.

–o *file* overrides the default destination for the output GIPS image (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the output image.

FILES

/usr/gips/tables/cprofile image header definition file

SEE ALSO

gipstool(1)

Standards for the Preparation and Interchange of Data Sets, T. Z. Martin, *et al.*, NASA Planetary Data System, Version 1.1, Jet Propulsion Laboratory, 1989.

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

When an error is detected, *ihpds* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- *file*: unknown label format
- bad bits/pix: *count*
- bad histogram format: STREAM
- bad image format: *code*
- bad image format: STREAM
- image location not specified in PDS label
- *keyword*: bad value: *value*
- multiple input file(s): *name*
- *text*: bad image structure
- unsupported encoding type: *code*

NAME

ihpho – create and update a cellular GIPS image file

SYNOPSIS

ihpho [**–DHSZ**] [**–c** *color*] [**–m** *file*] [**–x** *num*] [**–y** *num*] *image* [*script*]

DESCRIPTION

The *ihpho* program creates and/or updates a GIPS cellular *image* file with a mixture of sub-images, graphics, and text, specified through a sequence of sub-commands read from a *script* file. If *script* is omitted, the sub-commands will be read from the standard input stream, *stdin*.

If the *image* file doesn't exist, it will be created with the number of pixels specified by the **–x** and **–y** options, and each pixel will be initialized to the value of the **–c** option. In this situation, **–x** and **–y** must be specified, and if **–c** is omitted, the pixels will be initialized to zero. When invoked with the **–H** flag, *ihpho* creates an image containing two-byte pixels. Otherwise, the image will contain one-byte pixels. If *image* already exists, **–x** and **–y** will be ignored. In either case, the **–c** value becomes the default drawing color, unless subsequently changed by the *color* sub-command.

The image can be written as a VICAR2 image or Sun Rasterfile by the *vicarfile* and *rasterfile* subcommands. These output images may be either monochrome (one-byte pixels) or full-color (three-byte pixels). There is no connection between the number of bytes-per-pixel in the photoproduct image file (1 or 2) and the number in the VICAR2 or rasterfile (1 or 3). The conversion is performed by table lookup from an external *colormap* file.

OPTIONS

–D generates debugging output, listing each command before executing it.

–H any newly created image file will contain 2 bytes per pixel. An existing image file must have been created with 2 bytes per pixel.

–S writes command statistics to the standard error stream, *stderr*, after successful program execution.

–Z override input pixel length checking when loading an image via the *image* command. Unless this flag is specified, *ihpho* will only load images whose pixel size matches the output image, i.e. one byte per pixel unless the **–H** flag is used, in which case 2 bytes per pixel.

–c *color* specifies the default color value for subsequent sub-commands. New images will have all pixels set to this value. It may subsequently be changed by the *color* subcommand.

–m *file* specifies the name of a GIPS file whose header variables are to be merged with a new cellular *image* file. It is illegal to use this option if the *image* file already exists.

–x *num* specifies the number of *x*–pixels in a new *image* file. If *image* already exists, the **–x** value will be ignored.

–y *num* specifies the number of *y*–pixels in a new *image* file. If *image* already exists, the **–y** value will be ignored.

SUBCOMMANDS

The *ihpho* sub-commands should be entered on separate lines. White space surrounding sub-command text will be ignored. Blank lines and lines whose first character is '#' will be ignored. Sub-command lines can be continued by ending them with a back-slash character.

arc *x y radius angle1 angle2 width*

draw an arc of given radius and width, centered at *x,y*. subtending angles between *angle1* and *angle2* from the vertical (y-axis). If the width is zero, extend the arc into a wedge.

bold *name*

set the font to be used to write boldface text. This will be selected by **\fB** or **\f3** within strings in **text** or **label** commands.

circle *x y radius width*

draw a circular ring of given radius and width, centered at *x,y*. If the width is zero, draw a solid circle.

color *color*
set the color used for subsequent text and graphic items, an integer between 0 and 255 (for single-byte pixels) or between 0 and 65535 (for two-byte pixels).

colormap *file*
read a file containing a series of lines specifying a triplet of *red*, *green*, and *blue* intensities in the range 0 through 255 to use when writing *vicar* or *rasterfile* image files. If *ihpho* is using one byte per pixel, the colormap file will contain 256 lines; if the **–H** argument was used, the colormap file must contain 65536 lines. Null lines and lines beginning with '#' are comments, and will be ignored.

font *file* reset the text font file. If a relative pathname is used, the current working directory will be searched first, then "/*usr*/*lib*/*vfont*", and finally "/*usr*/*lib*/*fonts*/*fixedwidthfonts*".

header *file*
read the header from the GIPS image *file*. Subsequent *latitude* and *longitude* sub-commands will then work correctly, without the overhead of reading the entire image file. Alternatively, if *file* begins with the vertical bar `|´, the remainder of the line will be executed as a shell command, and the output of this command interpreted as a GIPS image header that defines the action of subsequent *latitude* and *longitude* sub-commands.

image *x y file*
insert the GIPS image from *file*, with its top left corner at *x,y*. Alternatively, if *file* begins with the vertical bar `|´, the remainder of the line will be executed as a shell command, and the output of this command interpreted as a GIPS image to be inserted in the image file. Input image pixels are inserted into the output file without any translation. If the **–Z option is specified, input pixels** zero-filled on the left to match the output size. Otherwise, the input and output pixels must have the same length.

include *file*
temporarily suspend reading the input file and execute the commands in *file*. *Include* commands cannot be nested, i.e. an included file cannot contain *include* commands. During execution of an included file, the action of the **–D** debug flag is suppressed.

italic *name*
set the font to be used to write italic text. This will be selected by **\fI** or **\f2** within strings in **text** or **label** commands.

label *x y string*
insert the text string at *x,y*. For origin codes 0, 4, and 5, the text will be vertically aligned to its baseline.

latitude *lat lon1 lon2 incr width*
draw a parallel of latitude at *lat* degrees between longitudes *lon1* and *lon2* degrees. The curve will use straight-line segments of *incr* degrees of longitude, and *width* specifies the line width in pixels. The information required to map latitude and longitude into the image array is derived from the most recent GIPS header read by an *image* or *header* sub-command.

line *x1 y1 x2 y2 width*
draw a straight line of given width between *x1,y1* and *x2,y2*.

longitude *long lat1 lat2 incr width*
draw a meridian of longitude at *long* degrees between latitudes *lat1* and *lat2* degrees. The curve will use straight-line segments of *incr* degrees of latitude, and *width* specifies the line width in pixels. The information required to map latitude and longitude into the image array is derived from the most recent GIPS header read by an *image* or *header* sub-command.

origin *code*

reset the text origin code. See *comlabel*(3) for details.

rasterfile *shrink depth name*
copy the photoproduct file to the Sun raster file *name*, as a monochrome or RGB image, depending on whether *depth* is 1 or 3. The image is reduced (by nearest-pixel resampling) by a factor of *shrink*, which must be a positive integer. If *name* begins with `|´, the remainder of the name is interpreted as a UNIX command which is processed by the Bourne shell (*/bin/sh*), which is supplied the raster file as its standard input stream, *stdin*.

rect *x1 y1 x2 y2 width*
draw a rectangular frame of given width with outer corners at *x1,y1* and *x2,y2*. If the width is zero, draw a solid rectangle.

reset *x y* reset the drawing origin to *x,y*. Until the next *reset*, *x,y* addresses in subsequent commands will be interpreted relative to this origin.

symbol *name*
set the font to be used to write symbol characters. This will be selected by **\fS** or **\f4** within strings in **text** or **label** commands, or when special **\(cc** strings are interpolated.

text *x y string*
insert the text string at *x,y*. For origin codes 0, 4, and 5, the text will be vertically aligned to its mid-point. See *comlabel*(3) for information on how to select special characters and how to change font size and character position within *string*.

vicarfile *depth name volser file*
copy the photoproduct file to the VICAR2 file *name*, as a monochrome or RGB image, depending whether *depth* is 1 or 3. If *name* begins with "/*dev*/", *ihpho* assumes that the output is to tape, and creates ANSI standard labels (level 3), with volume serial number *volser*, and file name *file*.

vwedge *x1 y1 x2 y2 color1 color2 [m v]*
draw a rectangular *color wedge* between *x1,y1* and *x2,y2*, with colors running from *color1* on the top to *color2* on the bottom. If *m* and *v* are defined, the color value is then masked with *m* (logical AND), and then the value of *v* is logically OR'ed with the result.

wedge *x1 y1 x2 y2 color1 color2 [m v]*
draw a rectangular *color wedge* between *x1,y1* and *x2,y2*, with colors running from *color1* on the left to *color2* on the right. If *m* and *v* are defined, the color value is then masked with *m* (logical AND), and then the value of *v* is logically OR'ed with the result.

ximage *x y file*
is identical to the *image* subcommand except that only non-zero valued pixels of the input image will replace pixels in the photoproduct file. Non-zero input image pixels are inserted into the output file without any translation. If the **–Z option is specified, input pixels** zero-filled on the left to match the output size. Otherwise, the input and output pixels must have the same length.

AUTHOR

Peter G. Ford, MIT CSR.

SEE ALSO

comlabel(3), gcell(3), ihgfmt(1), rasterfile(5), vfont(5)

DIAGNOSTICS

- **bad arg:** *arg*
  too many *ihpho* arguments specified.
- **bad args:** *text*
  an unexpected error occurred when parsing the specified subcommand.
- **bad** #*n* **arg:** *text*

there is an error in subcommand argument number *n*.

- *file:* **bad command:** *text*
  the command word is illegal.
- *file:* **bad dtype:** *dtype*
  the secondary image file is in cellular format, or it has more than one byte per pixel.
- **bad frame number:** *frame*
  the *number* argument of the **frame** command must be 1, 2, or 3.
- **bad** *x*/*y* **origin:** *org*
  the coordinates of the *reset* command must be within the image.
- **illegal nesting:** *file*
  the named file cannot be **included** within an included file.
- **no photoproduct file opened**
  an updating command was specified before any image was created or initialized by a prior **open** command.

NAME

ihproj – apply a non-local projection to a General Image

SYNOPSIS

ihproj [*–nproj*] [**–DSf**] [**–R** *fact*] [**–XY** *org*] [**–d** *dir*] [**–mo** *file*] [**–l** *long,lat,rho*] [**–Pp** *ang*] [**–r** *rad*] [**–bnxy** *num*] [*name*]

DESCRIPTION

The input General Image in *name* is transformed into a given cartographic coordinate system. If *name* is omitted, the input is taken from the standard input stream, *stdin*. *Ihproj* first calls *ihgfmt*(1) to reformat the input image into a form more suitable for random access, and then calls *ihgrid*(1) to generate transformation coefficients and *ihgeom*(1) to apply those coefficients to the random access file. *Ihproj* writes its output to the file specified by the **–o** option, or, if omitted, to the standard output stream, *stdout*. The following options may be specified:

–nproj specifies the desired transformation. See the manual entry under *ihgrid*(1) for possible values.

–D generates "debug" output from *ihgrid* and *ihgeom* on *stderr*, detailing set-up parameters.

–P *ang* specifies the second standard parallel (in degrees) for Lambert Conformal (–7) and Albers Equivalent (–12) transformations.

–R *fact* multiplies the default transform scaling factor by the floating-point quantity *fact*, thereby "fine tuning" the size of the output image. Alternatively, the user may supply a scaling factor via the **–r** option (qv.) This option is ignored for tiepoint transformations (–0).

–S writes a line of UNIX "statistics" to *stderr* upon command completion.

–X *org*
–Y *org* specify, respectively, the *x* and *y* origins of the transformed image in the output pixel space. This is, typically, the point into which the input origin defined by the **–l** option is mapped, although this is not true for all transformations. These options are ignored for tiepoint transformations (–0).

–b *num* specifies the number of buffer rasters to be used. The default value is 32. The larger the value, the more efficient the transformation, but *ihproj* will use more memory. The size of the actual output buffer allocated by *ihproj* will be this number times the length of each raster.

–f *dir* specifies the directory that *ihproj* will use for its scratch files. These files will occupy approximately the same space as the original input image itself.

–l *long,lat,rho*
defines the Euler angles (in degrees) of a rotation to be applied to the (longitude,latitude) coordinate system before the Cartographic transformation is applied. The origin of longitude will become *long*, the origin of latitude will become *lat*, and the resulting figure will be rotated clockwise by *rho* . If unspecified, these angles will be set to zero. This option will be ignored for tiepoint transformations.

–m *file* specifies the name of another General Image file whose header is to be merged with that read from *name* or *stdin* before the mapping is begun.

–n *order* is used only with tie-point transformations, and specifies the order of bi-linear polynomials to be used.

–o *file* the transformed General Image will be written to *file*, rather than to the standard output stream *stdout*.

–p *ang* specifies the central latitude (or first standard parallel) of the transformed image (in degrees). If this option is omitted, *ihproj* uses the arithmetic mean of the realworld *y*-axis limits of the input General Image. It is ignored for tiepoint transformations (–0).

–r *rad* specifies the transform scaling factor – whose meaning depends on the particular non-tiepoint transformation selected. If not specified, *ihproj* will compute a value that attempts to maximize the area of input image mapped into the output image. Some clipping and/or bordering is almost guaranteed! Use the **–R** option for fine tuning.

–x *num*
–y *num* specify, respectively, the x– and y–pixel extents of the output image.

name specifies the name of the General Image file to be transformed. If omitted, *ihproj* expects to read the image (in fact, its header only) from the standard input stream "*stdin*".

FILES

/usr/gips/tables/cprofile image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihgeom(1), ihgfmt(1), ihgrid(1), gtrans(3)

Ford, P., *Guide to GIPS*, MIT Center for Space Research, 1984.

Richardus, P. and Adler, R., *Map Projections*, North Holland/American Elsevier, 1972.

Cogley, J.G., *Map Projections with Freely Variable Aspect*, EOS, **65** 34, p.481 (1984).

DIAGNOSTICS

When an error is detected, *ihgeom* halts with system code 1, writing a message to the standard error stream *stderr*. Possible error messages are

- bad –n value: *value*
- bad arg: *value*
- forward mapping for *projection* not implemented
- illegal numeric in input stream: *code*
- inverse mapping for *projection* not implemented
- mapping is singular for this origin choice
- no –P value specified
- setup for *projection* not implemented
- singular tie-point transformation
- unable to compute –r value

NAME

ihrot – rotate, reverse, or invert a General Image

SYNOPSIS

ihrot [*–size*] [**–inr**] [**–d** *dir*] [**–hmo** *file*] [*name*]

DESCRIPTION

The *ihrot* command reads a General Image from file *name* and rotates it by 90 degrees in the counter-clockwise sense. If *name* is omitted, input is taken from the standard input stream *stdin*.

–*size* specifies the maximum number of bytes to allocate to the in-core buffer used to rotate the image. The default is ¼ megabyte, i.e. "*–262144*". Larger images will be rotated by writing them into a series of temporary scratch files.

–o *file* overrides the default output destination (the standard output stream *stdout*), and directs it to *file* instead.

–r reverses each input raster from right to left before rotation.

–i inverts the input image from top to bottom before rotation.

–n instructs *ihrot* to perform no rotation; the **–i** and/or **–r** flags will then invert an image without rotating it.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the input image. If the image is to be rotated (**–n** flag omitted), any file specified by a **–h** option is read *before* the *x* and *y* header items are swapped. Conversely, any **–m** file is read *after* swapping the fields, but before commencing the rotation.

–d *dir* overrides the default directory "/*tmp*" into which *ihrot* will write scratch files if the rotation cannot be performed in core.

FILES

/tmp/ihr*n	default temporary spool file(s)
/usr/gips/tables/cprofile	image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihread(3)

DIAGNOSTICS

When an error is detected, *ihrot* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- multiple input files
- *item* < 1
- bad operation for this *item*
- bad dtype value: *value*
- cannot invert
- interrupt

NAME

ihs – strip header and convert image to 1-byte per pixel format

SYNOPSIS

ihs [**–cinqs**] [**–hm** *file*] [**–fo** *file/#*] [**–** *size*] [**–l** *length*] [**–xy** *org*] [**–XY** *lim*] [*name*]

DESCRIPTION

ihs reads an image file from *name*, or, if *name* is omitted, from the standard input stream *stdin*. It converts it to byte-format (one pixel per byte) using the information contained in the image header. If the image has been passed through *ihfft*(1) so that *dtype=c*, each complex pixel will be replaced by its absolute value before further conversion. What happens next to each pixel (*pix*) is determined by the value of header variable **dscale**:

s $pix \rightarrow$ float(pix)
n $pix \rightarrow dfact \times pix + dzero$.
l $pix \rightarrow \log_{10}(pix)$
p $pix \rightarrow dparm_0 + dparm_1 \times pix + dparm_2 \times pix \times pix + ...$ (*npoly*+1 terms)
f $pix \rightarrow n$, $f[n] \le pix < f[n+1]$, for *f* in the file *dfile*.

In all cases except *dscale=f*, *pix* is further transformed by

$$pix \rightarrow 256 \times (pix - dmin)/(dmax - dmin),$$

where **dmax** and **dmin** default to 0 and 256, respectively, if not defined in the header. The above description is purely symbolic – *ihs* optimizes the conversions whenever possible by constructing lookup tables, using bit-shifts when multiplying or dividing by powers of 2, etc. etc.

–*size* sets the output blocksize, in bytes, when no re-formatting is to be performed (the **–s** option), and no header generated (i.e. no **–h** option specified).

–c specifies that the input file (or pipe) is not read past that point when the output array has been written. If the input came from a pipe, this may result in a "broken pipe" error message.

–h *file* the header will be saved in the named file. To write the header to the standard output, use "**–h –**". If the **–o** flag specifies that the image is being written directly to the display device, the header will not be written to the standard output until *after* the image is complete; otherwise, the header is written *before* the image. This logic ensures that *ihs* and *iged* do not attempt to write to the display at the same time.

–q eliminates certain warning messages (see under "Diagnostics", below).

–s bypasses all image reformatting. The input image array is copied, without any alteration whatever, to the output designated by the **–o** option. This flag may therefore be used to remove a header from an image. The output blocksize may be specified via the *–size* option.

–f *file* writes to *file* a *stretch-table* consisting of 256 unsigned short integers. The table is generated from a histogram of output 8-bit pixel values, and may be used to maximize the usable contrast of the displayed image. If *file* is numeric, the table is written directly to the stretch table of that number in the display device itself.

–m *file* specifies the name of a General Image file whose header will be merged with that of the input image.

–n instructs *ihs* not to process the input image itself, but to write the output header to the output stream defined by the **–h** option. This is a useful way to extract a header from a General Image. It is equivalent to "*gin -n*".

–l *size* specifies a maximum line length (in bytes) for the output header. This is most useful in conjunction with the **–n** option; i.e. "*ihs –n –l1 name*" writes the header of General Image file *name* to the standard output with one item per line.

–o *file* specifies that the stripped image will be written to the file specified, or, if omitted, to the standard output *stdout*. If the file name is numeric, the image will be written directly to the display image plane of that number.

–i specifies that the image will be *inserted* into the image plane – surrounding pixels will not be cleared; otherwise, the entire image plane will be rewritten. This option is only valid when the output is written directly to an image plane.

–x *org*
–y *org* override any definition of image origin supplied by the header variables *xorg* and *yorg*.

–X *lim*
–Y *lim* override the image display width variables *xlim* and *ylim*.

FILES

/dev/image.*	image plane pseudofile (see *comsubs*(3)).
/dev/function.*	function table pseudofile (see *comsubs*(3)).
/usr/gips/tables/cprofile	image header definition table

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihc(1), iged(1)

DIAGNOSTICS

If the image display will be truncated by the particular choice of *org* or *lim* variables, the message

"**ihs: warning - image truncated on:** *left*/*right*/*top*/*bottom*"

is written to the standard error stream *stderr*, unless the **–q** (quiet) option is specified.

When any other error is detected, *ihs* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- multiple input files: *file*

BUGS

The *ihs* command is only implemented (fully) for the Comtal Vision ONE/20 display system, and (partially) for the AED 767 frame buffer.

NAME

ihstokes – translate a JPL AIRSAR image to GIPS image format

SYNOPSIS

ihstokes [**–DLSb**] [**–c** *te,ta,re,ra*] [**–mo** *file*] [*name*]

DESCRIPTION

The *ihstokes* command reads a JPL AIR-SAR image file, selects a particular radar polarization, and writes it out to the standard output, or to the file specified by the **–o** flag, in GIPS image format, with *dtype=f* (float) pixels.

–S selects verbose mode and writes the following statistics to the standard error stream, *stderr*:

- Transmitted phase and eccentricity
- Transmitted Stokes Vector
- Received phase and eccentricity
- Received Stokes Vector

–L write the output in deciBels, instead of the default which is linear in cross-section.

–S writes processing statistics to *stderr*.

–c *te,ta,re,ra*

specifies the polarization states desired in the output image:

te transmitted eccentricity
ta transmitted phase (degrees)
re transmitted eccentricity
ra transmitted phase (degrees)

The default value for all 4 values is 0.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the output image.

–o *file* overrides the default destination for the output GIPS image (the standard output stream *stdout*), and directs it to *file* instead.

FILES

/usr/gips/tables/cprofile image header definition file

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

When an error is detected, *ihstokes* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- bad –c value: *text*
- cannot find start of data
- image dimensions missing from header
- multiple input files
- not a Stokes image

NAME

ihtofits – translate a GIPS image into FITS format

SYNOPSIS

ihtofits [**–bS** *file*] [**–mo** *file*] [*name*]

DESCRIPTION

The *ihtofits* command reads a General Image from file *name*, updates the header from an file specified by the **–m** option, and writes it out to the standard output, or to the file specified by the **–o** flag, in "*fits*" format (Flexible Image Transport System). This operation is essentially the reverse of that performed by the *ihfromfits*(1) command.

–S *file* writes processing statistics to *stderr*.

–o *file* overrides the default destination for the output *fits* file (the standard output stream *stdout*), and directs it to *file* instead. When written to a tape or pipe, the blocksize will always be the *fits* standard, i.e. 2880 bytes.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the input image.

FILES

/usr/gips/tables/cprofile image header definition file

SEE ALSO

ihfromfits(1)

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

When an error is detected, *ihtofits* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- multiple input files

NAME

ihtogif – translate a GIPS image into GIF format

SYNOPSIS

ihtogif [**–MS**] [**–amo** *file*] [*name*]

DESCRIPTION

The *ihtogif* command reads a General Image from file *name*, updates the header from an file specified by the **–m** option, and writes it out to the standard output, or to the file specified by the **–o** flag, in Compuserve's GIF format. This operation is essentially the reverse of that performed by the *ihfromgif*(1) command.

–M transform the specified RGB colormap into a greyscale map, preserving the luminance of each pixel.

–S writes processing statistics to *stderr*.

–a *file* appends the *colormap* file *file* to the output *rasterfile*. See the manual entry for *gipstool*(1) for the format of a *colormap* file.

–o *file* overrides the default destination for the output *rasterfile* (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the input image.

FILES

/usr/gips/tables/cprofile	image header definition file
/usr/gips/lib/cmap	directory containing colormap files

SEE ALSO

ihfromgif(1)

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

When an error is detected, *ihtogif* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- bad colormap record: *text*
- multiple input file(s)
- unable to write GIF file

NAME

ihtopix – translate a GIPS image into Sun raster format

SYNOPSIS

ihtopix [**–S**] [**–amo** *file*] [*name*]

DESCRIPTION

The *ihtopix* command reads a General Image from file *name*, updates the header from an file specified by the **–m** option, and writes it out to the standard output, or to the file specified by the **–o** flag, in Sun "*rasterfile*" format. This operation is essentially the reverse of that performed by the *ihfrompix*(1) command.

–S writes processing statistics to *stderr*.

–a *file* appends the *colormap* file *file* to the output *rasterfile*. See the manual entry for *gipstool*(1) for the format of a *colormap* file.

–o *file* overrides the default destination for the output *rasterfile* (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the input image.

FILES

/usr/gips/tables/cprofile	image header definition file
/usr/gips/lib/cmap	directory containing colormap files

SEE ALSO

gipstool(1), ihfrompix(1), rasfilter8to1(1), rasterfile(5), rastrepl(1)

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

When an error is detected, *ihtopix* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- multiple input files

NAME

ihtrans – convert a General Image generated by an unlike CPU

SYNOPSIS

ihtrans [**–S** *file*] [**–mo** *file*] [*name*]

DESCRIPTION

The *ihtrans* command reads a General Image from file *name*, updates the header from an file specified by the **–m** option, and writes it out to the standard output, or to the file specified by the **–o** flag, translating the pixel fields to or from a *version* that is not supported on the local processor.

This command can be used to convert a General Image that is piped through the *rsh* command to or from a remote node of a network. To be consistent in an environment of unlike or unknown processors, *ihtrans* should always be the first command executed by *rsh* and the standard output of *rsh* should be immediately piped through *ihtrans*.

? Invoked with a single "?" argument, *ihtrans* writes a table of known *version* and *dtype* combinations to *stdout*.

–S *file* writes processing statistics to *stderr*.

–o *file* overrides the default output destination (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the input image.

FILES

/usr/gips/tables/cprofile	local header definition file
/usr/gips/tables/gipsversion	image header definition file

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

When an error is detected, *ihtrans* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- input conversion error: *file*
- output conversion error: *file*
- unsupported dtype: *char*
- unsupported version: *code*

NAME

ihvicar – translate a VICAR-2 file to GIPS image format

SYNOPSIS

ihvicar [**–bDS**] [**–mo** *file*] [*name*]

DESCRIPTION

The *ihvicar* command reads an image file in VICAR-2 format, writes it out to the standard output, or to the file specified by the **–o** flag, in GIPS image format.

–b byte-swap integer pixel fields. This operation may be necessary when reading VICAR-2 images that were written by a CPU that writes integers in a different format.

–D write the VICAR-2 header keywords and their values on the standard error stream, *stderr*.

–S write processing statistics to *stderr*.

–o *file* overrides the default destination for the output GIPS image (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the output image.

FILES

/usr/gips/tables/cprofile image header definition file

SEE ALSO

gipstool(1)

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

When an error is detected, *ihvicar* halts with system code 1, writing a message to the standard error stream *stderr*. Messages generated by image header routines are described in *ihread*(3). Other possible messages are

- VICAR label error in byte *n*
- bad *keyword* value
- *file*: unexpected EOF
- illegal FORMAT value: *format*
- keyword not in header: *keyword*
- multiple input files
- non-numeric *keyword* value
- non-string *keyword* value

NAME

xgips – display a GIPS image in an X window

SYNOPSIS

xgips [*Xview options*] [*gipstool arguments*]

DESCRIPTION

Xgips reads a general image file from *name*, and displays the image in an X window. It is a direct port of the *gipstool*(1) command to the XView application interface. All options and modes of that command are implemented, and this manual file only deals with those that differ in their effects.

The principal difference lies in the allocation of colors. The *xgips* default is to allocate 64 to the image (*gipstool* allocates 128), and their definitions are taken from every fourth entry in any specified colormap file. These colors **cannot** be changed during program execution, i.e. the **B**, **F**, **M**, **S**, and **s** subcommands are disabled. If the –N option is specified on the *xgips* command line, a "dynamic" colormap will be obtained and the colors can be changed, but at the penalty that the colors will only be displayed correctly when *xgips* has the "colormap focus". Some window managers, e.g. *olwm*(1), permit this focus to be retained intependent of the window focus itself.

Another difference lies in the way that *xgips* starts up. In order to be able to display the image as it is being read into the program, it is necessary to obtain the "input focus" directly at startup time. For this reason, it it is necessary that the cursor points within the image window when it first appears on the screen. Although *xgips* tries to place the cursor in the center of the window, some X window managers do not permit this to happen, and the user must physically move the cursor to point there.

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

"*Guide to GIPS*" *and* "*A GIPS Tutorial*"*, by Peter Ford, MIT Center for Space Research,* 1990.
gipstool(1)

BUGS

The colormap handling commands should be re-written to use Xlib subroutine calls directly to manage individual color cells.

NAME

comheight, comlabel commarker, comwidth, memlabel, memmarker, setauxfont, setfont, setink, setorigin, settype – write text to image/graphic file or array

SYNOPSIS

char *setfont(name)
char *name;

char *setauxfont(name,fontnum)
char *name;

char setink(c)
char c;

settype(type)

comlabel(fid,nlines,ncols,baseline,basecol,label)
char *label;

memlabel(array,nlines,ncols,baseline,basecol,label)
char *array, *label;

commarker(fid,nlines,ncols,centerline,centercol,label)
char *label;

memmarker(array,nlines,ncols,centerline,centercol,label)
char *array, *label;

setorigin(origin)

comheight(label)
char *label

comwidth(label)
char *label

cc ... –lGI

DESCRIPTION

These routines are used for labeling images and graphics. They are contained in the library *'/usr/local/lib/libGI.a'* and should therefore be loaded at compile/load time by using the *–lGI* option.

Setfont sets the named (Versatec/Varian-type) font for use in making labels. If *name* is **NULL**, a pointer to a static character array that holds the name of the current font-definition file is returned. Otherwise, *setfont* returns either **NULL** to indicate that something wild happened, or a pointer to a static character array that holds the name of the previous font.

Setfont will also attempt to figure out from *name* the point size, rotation sense, and highlight level of the new font, and, if it is a **roman** type, will load the **italic**, **bold**, and **symbol** fonts of the same style, size, and orientation. If not **roman**, *setfont* only attempts to locate a **symbol** font of the same size and orientation.

Setauxfont is used to change the current definition of only one of the four possible fonts, without affecting the other three. It is passed the font name and the integer *fontnum* from 0 through 3 representing, respectively, **roman**, **italic**, **bold**, or **symbol**. If *name* is **NULL**, *setauxfont* returns a pointer to the name of the previous font of this type (or **NULL** if there was none). Otherwise, it returns **NULL** if unable to locate the specified font, or a pointer to the font filename.

Setink stores its argument as the pixel value to use in producing characters within images. When writing to graphic planes, a non-zero argument indicates that bits are to be turned on, a zero argument that they are to be turned off. *Setink* returns the previous *ink* value.

Settype indicates whether the *array* or *fid* variables passed to subsequent invocations of *comlabel*, *memlabel*, *commarker*, or *memmarker* refer to an 8-bits-per-pixel image (*type=0*), or to a 1-bit-per-pixel graphic (*type!=0*). *Settype* returns the previous value of *type*.

Comlabel uses *fid* as a file descriptor for a readable and writable file associated with an image or graphic which is taken to be *nlines* lines by *ncols* columns of 1-byte pixels. The label specified by *label* (a null-character-terminated string) is written to the image or graphic in the previously defined font using *baseline* and *basecol* as the starting point of the label, and, for images only, using the pixel value defined in the last call to *setink*.

Memlabel is like *comlabel* except that instead of modifying an external image or graphic, it modifies an array in main computer memory.

Commarker and *memmarker* are like *comlabel* and *memlabel* except that the *centerline* and *centercol* of the characters making up the label are specified rather than the base line and base column.

Setorigin specifies which portion of the string is to be aligned with the chosen *centerline* and *centercol* arguments in the calls to *commarker* and *memmarker*. The argument *origin* should be in the range **0** through **8**, and causes *centerline* and *centercol* to be mapped according to the following table:

1	2	3
4	0	5
6	7	8

Thus, specifying *setorigin(0)* causes text to be centered, *setorigin(3)* places text wholly below and to the left of *centerline* and *centercol*, and so forth. *Setorigin* returns the previous value of *origin* (whose initial value is 0). To determine the current value without changing it, specify a value of *origin* lying outside the permitted range.

Fonts whose names end in the character 'r' are taken to be rotated fonts; labels written in such fonts go down the array rather than across it.

Comheight and *comwidth* return, respectively, the maximum height and width (in pixels) of their calling strings, when written into an image or graphic with the currently defined font. For rotated fonts, *comheight* returns the maximum height of rotated characters, i.e. reading left to right across the image.

Two fonts have been developed for use with *comlabel:* '**7seg.2**' provides numerals only in a fixed-width five line by three column format; '**5by5.2**' provides all ASCII characters in a fixed-width five line by five column format. The rotated versions of these fonts are also available.

ESCAPE CODES

Alternate fonts may be selected, and the location and size of characters in the graphic or image may be altered by the inclusion of special escape sequences within the strings passed to *comlabel*, *memlabel*, *commarker*, and *memmarker*. The syntax and action are identical to the following functions of the *troff*(1) typesetter, except that their function only lasts for the duration of the current string:

fR	switch to Roman font
fI	switch to Italic font
fB	switch to Bold font
s*n*	change point size to *n*
s+*n*	increase point size by *n*
s–*n*	decrease point size by *n*
s0	resume default point size
u	shift up one half line
d	shift down one half line
h*n*	shift up to *n* pixels above default
h+*n*	shift up by *n* pixels
h–*n*	shift down by *n* pixels
h0	resume default vertical position
v*n*	shift right to *n* pixels from default
v+*n*	shift right by *n* pixels
v–*n*	shift left by *n* pixels
v0	resume default horizontal position

\(*xx*	insert symbol character *xx* (see below)
\e	insert a back slash
\'	insert a back quote

The following escape sequences may be used to specify special characters in all fonts:

\(em	—	Em dash	**\(hy**	-	hyphen	**\(bu**	•	bullet
\(sq	□	square	**\(ru**	_	rule	**\(14**	¼	1/4
\(12	½	1/2	**\(34**	¾	3/4	**\(fi**	fi	fi
\(fl	fl	fl	**\(ff**	ff	ff	**\(Fi**	ffi	ffi
\(Fl	ffl	ffl	**\(de**	°	degree	**\(dg**	†	dagger
\(fm	′	foot mk	**\(ct**	¢	cent	**\(rg**	®	registered
\(co	©	copyright						

Those that are only defined in a symbol font are:

\(pl	+	math +	**\(mi**	–	math –	**\(eq**	=	math equals
\(**	*	math *	**\(sc**	§	section	**\(aa**	´	acute accent
\(ga	`	grave ‘	**\(ul**	_	u/rule	**\(sl**	/	slash
\(*a	α	alpha	**\(*b**	β	beta	**\(*g**	γ	gamma
\(*d	δ	delta	**\(*e**	ε	epsilon	**\(*z**	ζ	zeta
\(*y	η	eta	**\(*h**	θ	theta	**\(*i**	ι	iota
\(*k	κ	kappa	**\(*l**	λ	lambda	**\(*m**	μ	mu
\(*n	ν	nu	**\(*c**	ξ	xi	**\(*o**	ο	omicron
\(*p	π	pi	**\(*r**	ρ	rho	**\(*s**	σ	sigma
\(ts	ς	sigma	**\(*t**	τ	tau	**\(*u**	υ	upsilon
\(*f	φ	phi	**\(*x**	χ	chi	**\(*q**	ψ	psi
\(*w	ω	omega	**\(*A**	Α	Alpha	**\(*B**	Β	Beta
\(*G	Γ	Gamma	**\(*D**	Δ	Delta	**\(*E**	Ε	Epsilon
\(*Z	Ζ	Zeta	**\(*Y**	Η	Eta	**\(*H**	Θ	Theta
\(*I	Ι	Iota	**\(*K**	Κ	Kappa	**\(*L**	Λ	Lambda
\(*M	Μ	Mu	**\(*N**	Ν	Nu	**\(*C**	Ξ	Xi
\(*O	Ο	Omicron	**\(*P**	Π	Pi	**\(*R**	Ρ	Rho
\(*S	Σ	Sigma	**\(*T**	Τ	Tau	**\(*U**	Υ	Upsilon
\(*F	Φ	Phi	**\(*X**	Χ	Chi	**\(*Q**	Ψ	Psi
\(*W	Ω	Omega	**\(sr**	√	sqrt	**\(rn**	¯	root extender
\(>=	≥	>=	**\(<=**	≤	<=	**\(==**	≡	identic. equal
\(~=	≅	app =	**\(ap**	~	approx	**\(!=**	≠	not equal
\(->	→	rt arr	**\(<-**	←	lt arr	**\(ua**	↑	up arrow
\(da	↓	dn arr	**\(mu**	×	mult	**\(di**	÷	divide
\(+-	±	plus-mi	**\(cu**	∪	union	**\(ca**	∩	intersection
\(sb	⊂	subset	**\(sp**	⊃	superst	**\(ib**	⊆	improper subset
\(ip	⊇	imp sup	**\(if**	∞	infin	**\(pd**	∂	partial deriv.
\(gr	∇	grad	**\(no**	¬	not	**\(is**	∫	integral sign
\(pt	∝	prop to	**\(es**	∅	empty	**\(mo**	∈	member of
\(br	|	vert	**\(dd**	‡	2dagger	**\(rh**	⇒	right hand
\(lh	⇐	lt hand	**\(bs**		Bell	**\(or**	|	or
\(ci	○	circle	**\(lt**	⎧	top {	**\(lb**	⎩	bottom {
\(rt	⎫	top }	**\(rb**	⎭	bot }	**\(lk**	⎨	center {
\(rk	⎬	cent }	**\(bv**	⎪	bold |	**\(lf**	⎣	bottom [
\(rf	⎦	bot]	**\(lc**	⎡	top [	**\(rc**	⎤	top]

If there is any confusion between the end of an escape code and the text that follows it, you may follow the escape code with an extra back-slash. Alternatively, you may enclose numeric escape fields within single quotes, i.e. **\h'+24'2** shifts the remainder of the line by 24 pixels, then inserts a '2', whereas **\h+242** shifts the line upward by 242 pixels.

FILES

/usr/lib/vfont/	directory of font definition files
/usr/local/lib/libGI.a	library containing *comlabel* (3) routines.

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

cfont(1), vfont(5)

ACKNOWLEDGEMENT

comlabel is developed from a routine written by Arthur Olson of the National Cancer Institute, Bethesda. It has been modified to implement multiple fonts, *setorigin()*, *settype()*, and the in-line escape codes.

BUGS

Some rotated fonts are given incorrect height information in their definition file headers.

NAME

color_get, color_put, command_open, function_open, graphic_open, image_open, pseudo_open, target_dump, target_get, target_put, target_roam – interface routines to virtual GIPS display

SYNOPSIS

image_open(ni,mode)

graphic_open(ng,mode)

pseudo_open(np,mode)

function_open(nf,mode)

FILE *command_open()

target_get(nt,xy)
int xy[2]

target_put(nt,xy)
int xy[2]

target_roam(nt)

target_dump(nt)

color_get(ng)

color_put(ng,nc)

cc ... –lFB

DESCRIPTION

These subroutines may be used to interface between C routines and a "*generic*" image or graphic display device. They were written to support the 3M/Comtal Vision ONE/20 image processor, but permit the user to write code that does not depend on the detailed characteristics of the display device.

The four "open" routines return file descriptors that may be used in subsequent calls to *read*, *write*, *lseek*, and *close*. The size and format of individual data elements depend on the nature of the physical display device. Alternatively, the file descriptor may be exchanged for a stream pointer via a call to *fdopen*.

Image_open returns a file descriptor to image plane number *ni*. In addition, that image is displayed and, if possible, pseudocolored.

Graphic_open returns a file descriptor to graphic plane number *ni*. The chosen graphic plane will be displayed.

Pseudo_open returns a file descriptor to pseudocolor lookup table number *np*, and adds that lookup table to the processing pipeline for the displayed image.

Function_open returns a file descriptor to function (contrast stretch) lookup table number *np*, and adds that lookup table to the processing pipeline for the displayed image.

command_open returns a *stream* pointer via which device-dependent commands may be sent to the display device. *command_open* will always return NULL if the display is unavailable.

Target_get returns in the *xy* array the x and y coordinates of target number *nt*.

Target_put repositions target number *nt* to the x and y coordinates specified in the *xy* array.

Target_roam enables "roaming" of target number *nt*, allowing the user to manually reposition the target prior.

Target_dump enables "dumping" of target number *nt*. The meaning of this operation depends on the display device.

Color_get returns a numeric code that defines the color in which graphic plane number *ng* is currently displayed. The mapping between colors and codes is device dependent.

Color_put resets the color in which graphic plane number *ng* is to be displayed to that defined by the numeric code *nc*. The mapping between colors and codes is device dependent.

No particular restrictions are imposed on the order in which these routines are called, but it is recommended that each application program contain an initial call to *command_open* to determine whether the display device is available.

DEVICE DEPENDENT MACROS

In addition to the above routines, the C-language header file "*/usr/gips/include/gips.h*" contains a series of #**define** and **typedef** statements that reflect the physical characteristics of the display device:

ROWS	number of rasters (i.e. y-values) in display device
COLS	number of pixels per raster (i.e. x-values) in display device
NIMAGE	number of bytes in an image plane
NGRAPH	number of bytes in a graphic plane
PSEUDO	is *typedef*'d to a pseudocolor lookup table element.
FUNCTION	is *typedef*'d to a function lookup table element.
ISCOL(*x***)**	true if *x* is a legal column number
ISROW(*y***)**	true if *y* is a legal row number
TOPIX(*x,y***)**	returns the element offset to pixel *(x,y)* in an image plane
PIXEL(*i,x,y***)**	addresses the *(x,y)* pixel in the incore image plane array *i*.
SETGRAPH(*g,x,y***)**	sets pixel *(x,y)* in the incore graphic plane *g*.
UNSETGRAPH(*g,x,y***)**	clears pixel *(x,y)* in the incore graphic plane *g*.

COMTAL IMPLEMENTATION

When an interface routine other than *command_open()*, *color_get()*, or *color_put()* is called, it may direct one or more commands to the display device. These commands may be tailored by the user by placing in his or her home directory a file named "*.clrc*", containing one or more "skeleton" command lines. The following shows the **default** form of these commands:

```
image_open      SU A SU B D I %(ni) A PS
graphic_open    CO G %(ng) %(nc) A G %(ng)
pseudo_open     A PS %(np)
function_open   A F %(nf)
display_put     S D %(x) %(y)
target_get
target_put      A T S TAR %(x) %(y)
target_roam     SU A A B A T RO I
target_dump     SU A A B A T DU I
```

Each line begins with a subroutine name, followed by a single (required) tab, followed by the text (if any) of the command to be sent to the display device whenever that subroutine is called. The constructs "*%(...)*", which actually appear as written in the file, are replaced at execution time by the *values* of the parameters specified in parentheses; i.e. if a call were made to *function_open(2,0)*, the command "**A F 2**" (i.e. "ADD FUNCTION MEMORY 2") would be sent to the display device.

The above example displays the *default* values, so you don't need to include any of those exact lines in your own *~/.clrc* profile, only the lines that you want to change. For instance, if it irks you that *image_open()* always adds pseudocolor tables to the images that it displays, you could create a file "*~/.clrc*" containing

```
image_open      SU A SU B D I %(ni)
```

Here is some more information about the individual subroutines, as they have been implemented for the display device.

command_open*()*

If the stream opened by a previous call to *command_open* has since been closed by the user, it is reopened and the display device keyboard input buffer is cleared (i.e. the equivalent of hitting the <**ESC**> key on the keyboard). The user should only close the keyboard stream if it is desirable that another process access the display device while the current process is running (or stopped). Otherwise, it is more efficient if at least one display device feature is kept open throughout the process.

image_open*(ni,mode)*

The returned file descriptor refers to a "pseudofile" of 262144 bytes of physical image plane numbered *ni*, which may be read, written, or lseek'ed, (according to *mode*) but not extended.

graphic_open*(ng,mode)*

The returned file descriptor refers to a "pseudofile" of 32768 bytes of physical graphic plane number *ng*, which may be read, written, or lseeked, (according to *mode*) but not extended. *color_get()* is called to determine the color of the graphic (*nc*). If *nc* is zero (black), it is changed to *15* (white). *color_put(ng,nc)* is then called.

pseudo_open*(np,mode)*

The returned file descriptor refers to a "pseudofile" of 512 bytes of physical pseudocolor memory number *np*, (where np=0,1,2 corresponding to red, green, or blue), which may be read, written, or seeked, (according to *mode*) but not extended.

function_open*(nf,mode)*

The returned file descriptor refers to a "pseudofile" of 512 bytes of physical function lookup memory number *nf*, which may be read, written, or seeked, (according to *mode*) but not extended.

target_get*(nt,xy)*

The physical target coordinates are transformed according to the current zoom and roam settings for physical image number *nt*, and their values returned in the two-element *xy* array.

target_put*(nt,xy)*

The physical target is moved to coordinates given by the two-element *xy* array. If physical image number *nt* is currently roamed or zoomed, the coordinates will be transformed accordingly so as to refer to the physical address within that image plane rather than to the screen coordinates themselves.

color_get*(ng)*

The color number of physical image plane number *ng* is returned, as follows:

0:	black	8:	light pink
1:	blue	9:	magenta
2:	cyan	10:	orange
3:	dark green	11:	pink
4:	dark turquoise	12:	red
5:	forest green	13:	turquoise
6:	green	14:	light orange
7:	yellow	15:	white

color_put*(ng,nc)*

The color of physical graphic plane number *ng* is set according to the value of *nc*, as tabulated above. To alter the command that is sent to the display device, add a line in "~/*.clrc*" that begins *graphic_open*. The subroutines *graphic_open()* and *color_put()* share this definition.

FILES

/usr/gips/lib/libFB.a	library containing comsubs routines.
~/.clrc	file containing user profile
/usr/gips/include/gips.h	device definitions and error codes

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihread(3), ihs(1), iged(1)

DIAGNOSTICS

With two exceptions, all the above routines return the value **–1** to indicate that an error has occurred. These exceptions are *command_open* which returns **NULL**, and *target_put* which returns **+1** to indicate that the specified target coordinates are non-physical. (Note: *target_put* returns **–1** if the display device is unavailable). When an error occurs, the external variable *gipserrno* will be set to an error code as defined in *ihread*(3). This will typically be either **GIPSBADOPEN**, **GIPSBADREAD**, or **GIPSBADWRITE**.

BUGS

The current AED version of *comsubs*(3) only implements *image_open(1,mode)*, i.e. a single image plane. *command_open() always returns a valid stream* but all I/O to that stream will be ignored.

NAME

gcell, gclose, ginterp, gopen – access GIPS cellular image

SYNOPSIS

#include "**/usr/gips/include/gips.h**"

gimage *gopen(stream, nblock)
FILE *stream;

char *gcell(x,y,image)
gimage *image;

char *ginterp(x,y,image)
float x, y;
gimage *image;

gclose(image)
gimage *image;

cc ... –lGI

DESCRIPTION

Gopen sets up a *gimage* storage table, from which subsequent calls to *gcell* and *ginterp* can rapidly access pixel values. *gopen* is passed a pointer *stream* that was returned by a previous call to *ihread*(), which read a General Image file that had been created by *ihgfmt*(1). *Gopen* constructs a series of *nblock* buffers in which to save input blocks. If *nblock* is non-positive, a default of 8 blocks will be used. If *gopen* completes normally, it returns a non-NULL pointer value which should be saved by the caller and used in subsequent calls to *gcell*, *ginterp*, and *gclose*.

Gcell is passed the integer *x* and *y* coordinates of a given pixel, and the value returned by the prior call to *gopen* that saved that image. It returns a pointer to the binary value of the nearest pixel.

Ginterp is similar to *gcell()*, but it is passed the floating-point *x* and *y* coordinate values. It returns a pointer to a pixel value constructed by interpolating the four nearest neighbor image pixels.

Gclose terminates *gcell* calls to a particular General Image. Internal buffers are released and may be made available for subsequent *gopen* calls with other filename arguments.

The actual internal format of *gcell* files is transparent to the user, and may change without notice. The format presently used is that of 1K–byte blocks, chosen so as to optimize disk I/O within UNIX 4.2. Each block represents a rectangular image cell, whose *x* and *y* dimensions vary according to the pixel length, but are always chosen to be powers of 2 to facilitate address decoding. *Nblock* data blocks are held in core simultaneously, and are refreshed from disk on a first-in-first-out basis.

FILES

/usr/gips/lib/libGI.a	library containing GIPS routines.
/usr/gips/include/gips.h	typedefs and error codes

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihgfmt(1), ihread(3), gips(3)

DIAGNOSTICS

When an error is encountered, *gopen* stores an error code in the external variable *gipserrno*, and returns the value NULL. The error codes are documented in *ihread*(3).

When called with coordinates *x* or *y* that lie outside the defined image, *gcell* and *ginterp* return a pointer to the *dnull* value in the image header. *Gcell* returns a *NULL* value when *nimage* does not correspond to the value returned from a (non-failing) *gopen* call.

Gclose returns zero if the image was released, otherwise it stores an error code in *gipserrno*, and returns the value –1.

NAME

gips, egips – translate between GIPS realworld values and pixel coordinates

SYNOPSIS

#**include** "**/usr/gips/include/gips.h**"

gips(real,pixel,nitem,mode)
double ∗**real;**
int ∗**pixel;**

egips(pixel,real,nitem,mode)
int ∗**pixel;**
double ∗**real;**

extern gipserrno;
extern char ∗**gipserrs[];**

cc ... –lGI

When called from Fortran, declare **real** to be a double precision array, **pixel** to be an integer array, and **nitem** and **mode** to be integers. Treat *gips*() and *egips*() as integer functions.

When *gips()* is called with *mode=2*, **real** is an array of elements whose data type is given by the *dtype* variable in the current incore General Image header.

DESCRIPTION

These routines translate arrays of General Image realworld coordinate values into their corresponding pixel-space values, and vice versa. Before either routine is called, a legal GIPS image header must be read into core via a call to *ihread*().

nitem specifies the number of items in the *pixel* and *real* arrays. *mode* specifies the data type – **mode=0** for x–axis coordinates, **mode=1** for y–axis coordinates, and **mode=2** for pixel values themselves. Note that each element of *pixel* and *real* must refer to the same type of data – x, y, or d. To convert, for instance, x–axis and y–axis coordinates requires *two* calls to *gips*() or *egips*(), one with **mode=0** and the other with **mode=1.** Alternatively, specify **mode=3,** in which case *real* and *pixel* point to *2∗nitem* elements, specifying alternating *x* and *y* values.

After a call to *gips*(), each *x* or *y* pixel value that is outside the limits of the image will be translated to a **real** value of *GIPSLOW* or *GIPSHIGH*, depending on whether it falls on the high or low side of the imaging space.

After a call to *egips*(), any **real** value that is outside the realworld limits of the image will be translated to a *pixel* value of "–1" or "num", where "num" represents the appropriate image dimension, i.e. *xnum*, *ynum*, or *256*, for *mode=0,1,2* respectively.

FILES

/usr/gips/lib/libGI.a	library containing GIPS routines.
/usr/gips/include/gips.h	symbol and error number definitions

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihread(3)

DIAGNOSTICS

When an error is encountered, *gips*() and *egips*() return the error codes described in *ihread*(3). These codes are also available to C callers in the external variable *gipserrno*.

NAME

g_setup, g_map, g_unmap – apply non-local transformation to image coordinates

SYNOPSIS

#include "**/usr/gips/include/gips.h**"

int g_setup()

int g_map(lon, lat, xy)
double lon, lat, xy[2];

int g_unmap(x, y, lonlat)
double lonlat[2];

cc ... –lGI

DESCRIPTION

The *g_setup*() function sets up data structures to transform between input and output coordinates. Before calling *g_setup*(), it is first necessary to invoke *ihread()* to read a GIPS header file, and the header variable named "*mapping*" must be initialized to the following character string:

name=*cccc* **type=***n* **origin=***x,y* **radius=***r* **euler=**ϕ,θ,ϕ′ **lat=**θ′,θ″ **xyrot=***sw*

cccc Name of projection—unused but must be present.

n Projection number from 0 through 21. See the *ihgrid*(1) manual entry and the ``Guide to GIPS´´ document for more details.

x,y Pixel address in the output image corresponding to the center of the input image, i.e. longitude θ and latitude ϕ.

r The mapping scale factor, whose meaning depends of the particular transformation selected. Generally, increasing *r* will magnify the output image.

ϕ,θ,ϕ′ The Euler rotation angles to be applied to latitudes and longitudes in the input image before the cartographic mapping itself. The rotation brings input longitude θ and latitude ϕ to the center of the output image (i.e. output pixel address *x,y*), and rotates the result by ϕ′. These angles are disregarded for tiepoint transformations (type 0).

θ′,θ″ The special latitude or latitudes that have significance for some of the cartographic transformations, e.g. Lambert conic (types 6 and 7) and Albers (types 11 and 12).

sw An optional field. If the value of *sw* is non-zero, the image is given a final 90° counterclockwise rotation.

After a successful call to *g_setup*(), *g_map*() translates from longitude *lon* and latitude *lat* in the input image to *x*–pixel address *xy*[0] and *y*–pixel address *xy*[1] in the output image. If either *lat* or *lon* is invalid, *g_map*() returns a value of 1. Otherwise, it returns 0.

After a successful call to *g_setup*(), *g_unmap()* translates from pixel address *x,y* in the output image to longitude *lonlat*[0] and latitude *lonlat*[1] in the input image. If the specified output pixel address corresponds to a physical input longitude and latitude, *g_unmap*() returns a value of 0. Otherwise it returns 1. No check is made to determine whether a physical latitude and longitude is actually represented in the input image that is being processed, i.e. the user should test that *lonlat*[0] lies between *xmin* and *xmax*, and that *lonlat*[1] lies between *ymin* and *ymax*.

Latitudes, longitudes, and Euler angles are expressed in degrees, except for tie-point transformations (when the *lon*, *lat*, and *lonlat* variables refer to input pixel coordinates), and user-defined projections (when the definition of *lon*, *lat*, and *lonlat* is left to the user).

FILES

/usr/gips/lib/libGI.a	library containing GIPS routines.
/usr/gips/include/gips.h	symbol and error number definitions

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

``Guide to GIPS´´, Peter G. Ford, MIT Center for Space Research report.

ihgrid(1), gips(3), ihread(3)

DIAGNOSTICS

When an error is encountered in reading GIPS header variables, *g_setup*() return one of the integer codes described in *ihread*(3). The error code is also stored in the external variable *gipserrno*.

NAME

gipserr, gipsint, ihbcl, ihbin, ihbout, iherrm, ihget, ihmerg, ihput, ihread, ihtype, ihwrit – manipulate a GIPS image header

SYNOPSIS

#include <stdio.h>
#include "/usr/gips/include/gips.h"

FILE *ihread(name,profile)
char *name, *profile;

FILE *ihmerg(name)

FILE *ihwrit(name,length)
int length;

ihget(item,value*[,lvalue]***)**
char *item, *value;

ihput(item,value)

ihdel(item)

ihtype(name,value*[,lvalue]***)**
char *name, *value;

ihbin(buffer,file,length)
char *buffer;
FILE *file;

ihbout(buffer,file,length)

ihbcl(file)

gipserr();

gipsint(proc);
void (*proc)();

extern gipserrno;
extern char *gipserrs[];

call iherrm

cc ... –lGI

When called from Fortran routines, omit the *lvalue* argument to *ihget()* or *ihtype()*, and treat all the "*ih*" routines as integer functions. If an error condition is detected, "**call iherrm**" will write a disgnostic message to the standard error stream, *stderr*, and then return to the Fortran caller.

DESCRIPTION

Ihread reads a General Image header from file *name*, verifying its contents against the expected format stored in file *profile*. If *name* is the string ("-"), input is taken from the standard input stream *stdin*. If *name* or *profile* specifies a relative pathway, the current directory is searched first, then "/*usr*/*gips*/*tables*/". The contents of any previously-read header are lost. *Ihread* returns either the stream-buffer pointer to *name*, or **NULL**, (zero in Fortran), signifying an error while reading.

Ihmerg adds information to that gathered in a previous call to *ihread* by reading a secondary stream. If *name* is the string ("-"), input is taken from the standard input stream *stdin*. If *name* specifies a relative pathway, the current directory is searched first, then "/*usr*/*gips*/*tables*/".

Ihwrit writes an existing General Image header that has been previously input via a successful call to *ihread*. The maximum output line length may be specified via the *length* parameter: at least one item will be written on each line. When *length* is zero, negative, or larger than some implementation limit, the default of 72 characters per line will be used. The output is to the file *name*, unless the latter is the string ("-"), when the standard output stream *stdout* is used. *Ihwrit* returns either the stream-buffer

pointer to *name*, or **NULL**, (zero in Fortran), signifying an error while writing.

After successful completion, *ihread* and *ihmerg* leave the input stream positioned such that the next byte to be read is the first data pixel. *Ihwrit* flushes its output buffer so that the next item written should be the first data pixel.

If the *name* parameter of *ihread*, *ihmerg*, or *ihwrit* begins with a "|" character, the remainder of the parameter string is interpreted as UNIX command-shell input. It is passed to "/*bin*/*sh*" and the standard input or output of that sub-shell is assumed to be a legal general image header.

Ihget and *ihput* are used to access header values after a successful call to *ihread*. For both functions, *item* is the name of a header item, and *value* is a pointer to a variable of the appropriate type located within the caller's program. When called from C, *ihget* needs to know the maximum length of character array that may be returned, specified by *lvalue*. *Lvalue* should be omitted when calling *ihget* from Fortran.

Ihdel is used to remove an item from the in-core header, so that it will not be written out by a subsequent *ihwrit* call. However, the definition is not discarded, so that the variable can be "put back" with a later call to *ihmerg* or *ihput*.

Allowed variable names and their data types are defined by entries in the file specified in the *profile* variable of the prior *ihread* call. Each variable is described by the following three fields, separated by blanks and/or tabs, ending with a newline character:

name The name of the variable. It may contain any number of alphameric characters provided the total line length does not exceed 200 bytes.

type A single character code describing the variable:

- **R** A required character-string variable which *must* be present in every header and whose value *must* be identical to the description field that follows this entry in the *profile* file. Such variables identify headers and software version numbers.
- **V** A required character-string variable which *must* be present in every header and whose value *must* be identical to one of the (space-delimited) sub-fields in the description field that follows this entry in the *profile* file. Such variables identify hardware formats.
- **h** A character-string *history* variable that may be *added* more than once to the header. Each addition will be prefixed by a character string representation of the current date and time. Only the most recent added *history* variable may be removed by *ihdel*().
- **s** A character string variable.
- **c** A single character variable.
- **i** An integer variable.
- **d** A double-precision floating-point variable.
- **E** An End-of-header delimiting variable, which is never given an actual value but merely serves to identify the end of the header.

description One or more words of description, usually serving merely to document the variable but, for data-type **R**, actually defining the value that that string variable must have when encountered in a header.

Within the header, each variable is represented by an ascii string "*name* = *value*", where the three subfields may be separated by any number of whitespace characters. If the *value* of a string-variable contains whitespace characters, the *external* value must be enclosed within double quotes, and imbedded double quotes must be preceded by a back-slash. *ihread* will remove these (and only these) back-slashes when reading the header, and *ihwrit* will insert them when writing it out again, so you do not therefore have to pay any special attention to the presence of double quotes in string values passed to *ihput* or returned from *ihget*.

Ihtype returns the type and description fields of a particular header item: it fills *value* with the type character (as shown above), followed by the description string. The maximum length of *value* is specified by *lvalue*. If *lvalue* is set to 0 or 1, only the **type** character will be returned. *Lvalue* must be omitted entirely when calling *ihtype* from Fortran.

Once a GIPS header has been read or written, rasters (i.e. arrays of pixels corresponding to the same y-value and increasing x-values) may be read or written via calls to *ihbin* and *ihbout*, respectively. The second *(file)* parameter must be taken from the value returned by the last (successful) call to *ihread* or *ihwrit*. The *length* parameter specifies the length of the buffer, in bytes. If the actual length of the raster, as defined by the header variables *xnum* and *dtype*, is greater than *length*, *ihbin* truncates the input raster and *ihbout* fills the remainder of the output raster with pixels given the *dnull* value. Both routines return the number of bytes *actually* read or written. *ihbin* returns 0 on an end-of-file. If any other error occurs, these routines return the *negative* of the GIPS error code (see below).

The *ihbcl* routine closes the GIPS file opened by a call to *ihread*, *ihmerg*, or *ihwrit*. Thus, the following C and Fortran examples are equivalent:

```
#include "/usr/gips/include/gips.h"
FILE    *f, *ihread();
char    buf[512];
f = ihread("my.image","cprofile");
if (ihbin(buf,f,512) < 0) gipserr();
ihbcl(f);
```

```
integer  i
character*512 buf
i = ihread("my.image","cprofile")
if (ihbin(buf,i,512).lt.0) call iherrm
call ihbcl(i)
```

For Fortran programmers, here is a more lengthy example of a program that copies General Image file "gi.file1" to "gi.file2", adding a simple history entry to the header:

```
      character*4097 buf
      data n /0/

      kin = ihread("gi.test","cprofile")
      if (kin.eq.0) stop 1
      call ihput("history","Copied to ""gi.test2""")
      kout = ihwrit("gi.test2",0)
      if (kout.eq.0) goto 30

10          in = ihbin(buf,kin,4097)
            if (in.le.0.or.in.eq.4097) goto 20
            n = n + 1
            iout = ihbout(buf,kout,4097)
            if (iout.gt.0) goto 10

20    if (in.lt.0.or.iout.lt.0) call iherrm
      if (in.gt.4096) write(0,'(" raster GT 4096 bytes")')

      write (0,'(" copied ",i5," rasters")') n
      call ihbcl(kout)
30    call ihbcl(kin)
      stop
      end
```

FILES

/usr/gips/lib/libGI.a	library containing GIPS routines.
/usr/gips/include/gips.h	error codes, etc

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

gcell(3), gips(3)

DIAGNOSTICS

Ihread, *ihmerg*, and *ihwrit* return **NULL** after detecting an error. The remaining routines return integer error codes. All routines save their return code in the external variable **gipserrno**, and C-language routines that include the file "/*usr*/*gips*/*include*/*gips.h*" may generate an error message via the statement **gipserr().** (From Fortran, execute "call iherrm"). The possible non-zero **gipserrno** values and associated error messages are as follows:

1	**GIPSBADARG**	bad GIPS routine argument
2	**GIPSNOHEAD**	no GIPS header in core
3	**GIPSBADOPEN**	open error: *file*
4	**GIPSBADREAD**	read error: *file*
5	**GIPSBADDATA**	bad data read: *file*
6	**GIPSNOVAR**	missing header variable: *item*
7	**GIPSNOVAL**	no value for header item: *item*
8	**GIPSNOEOF**	no end-of-header indicator
9	**GIPSBADVAR**	bad header entry: *item*
10	**GIPSBADVAL**	bad header item value: *item*
11	**GIPSBADTYPE**	bad header variable type: *item*
12	**GIPSTRUNC**	*item* value truncated
13	**GIPSBADLIM**	inconsistent *item* value
14	**GIPSUNSCALE**	unsupported *item* value
15	**GIPSNOSTR**	null *item* value
16	**GIPSNOCORE**	insufficient core
17	**GIPSBADWRITE**	write error: *file*

When a GIPS routine encounters an error, it stores the appropriate error code in *gipserrno*; the corresponding error messages are available in the external variable *gipserrs[gipserrno]*. If the message requires an inserted value (denoted above by *italics*), the insert is saved in *gipserrs[0]* and a *%s* placeholder is included in *gipserrs[gipserrno]*. The error message may therefore be generated within a "C" program by the function call

fprintf(stderr,gipserrs[gipserrno],gipserrs[0]);

Normally, a GIPS routine would wish to abort at this point, and this can most conveniently be achieved by using the *gipserr()* macro, which calls the *error()* routine in *-lGI*. If it is necessary to perform some cleanup task before exiting, *error* can be told to execute a user-specified routine by means of a prior call to *gipsint(proc)*, whose argument *proc* is the name of your cleanup routine. If, at some later stage of your program, this routine is no longer required, you must invoke *gipsint(0)* to signal that fact.

PERMUTED INDEX