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

© 1984–1992 Massachusetts Institute of Technology
† 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

$$\begin{array}{ccc} 1 & 0 & 2 \\ 0 & 4 & 0 \\ 2 & 0 & 1 \end{array}$$

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.