GUIDE TO GIPS

Peter G. Ford

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

ABSTRACT

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

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

February 4, 1992

† UNIX is a trademark of Bell Laboratories.

© 1984–1992, Massachusetts Institute of Technology

GUIDE TO GIPS

Peter G. Ford

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

INTRODUCTION

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

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

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

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

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

---

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

CONCERNING IMAGES

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

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

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

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

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

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

GENERAL IMAGE FORMAT

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

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

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

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

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

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

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

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

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

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

Here are some examples:

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

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

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

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

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

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

Table 1. GIPS Input Filters

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

Table 2. Common GIPS Command Options

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

Table 3. GIPS Filters

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

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

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

GENERAL IMAGE HEADERS

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

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

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

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

Table 6. Contents of a typical GIPS header.

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

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

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

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

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

[7] *daxis* is not currently implemented.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Table 8a. Summary of *dscale* definitions

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

Table 8b. Summary of *xscale* definitions

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

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

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

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

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

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

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

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

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

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

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

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

outside the physical image limits.

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

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

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

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

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

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

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

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

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

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

IMAGE-PROCESSING COMMANDS

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

CARTOGRAPHIC PROJECTIONS

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

AZIMUTHAL PROJECTIONS

1. Gnomonic:

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

2. Stereographic:

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

3. Orthographic:

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

4. Azimuthal Equidistant:

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

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

5. Azimuthal Equivalent:

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

CONFORMAL PROJECTIONS

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

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

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

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

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

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

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

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

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

8. Mercator:

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

9. Polar Stereographic:

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

10. Mollweide:

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

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

EQUIVALENT PROJECTIONS

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

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

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

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

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

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

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

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

13. Lambert Cylindrical:

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

14. Lambert Polar Azimuthal:

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

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

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

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

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

16. Sanson-Flamsteed Sinusoidal:

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

17. Werner:

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

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

MISCELLANEOUS NON-CONIC PROJECTIONS

19. Aitoff

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

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

20. Hammer

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

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

21. Briesemeister

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

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

USER SOFTWARE WITHIN GIPS

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

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

Table 9a. Sample GIPS program written in Fortran

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

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

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

Table 9b. The same program written in C

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

BIBLIOGRAPHY

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

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

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

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

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

ACKNOWLEDGEMENTS

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

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