NAME

gin – generate a dummy General Image

SYNOPSIS

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

DESCRIPTION

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

```
gin | ihrot | git
```

The default header is as follows:

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

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

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

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

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

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

FILES

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

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

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

DIAGNOSTICS

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

- bad argument: *arg*

NAME

gipsmongo – use Mongo to display a General Image file

SYNOPSIS

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

DESCRIPTION

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

OPTIONS

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

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

–o *device*

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

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

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

GIPSMONGO SUBCOMMANDS

contour *#levels level1 level2 ... levelN*

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

header

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

inform

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

mongo

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

next

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

plot3d *zenith azimuth scale origin*

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

quit

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

show

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

! *command*

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

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

MONGO SUBCOMMANDS

3dplot *alt az zmax*

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

angle *d [x1 y1 x2 y2]*

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

axis *a1 a2 asmall abig ax1 ay1 ax2 ay2 ilabel iclock*

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

box *[m n]*

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

buffer

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

connect

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

contour *n level1 ... leveln*

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

cursor *[x y]*

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

data *fspec*

data opens a data file for reading data. The file is assumed to have numerical data in columns separated by spaces, tabs or commas. Any column can be read as x or y coordinates by *x* and *ycolumn*. In the case that *mongo* has been called as a subroutine, *data* with no arguments specifies that data be read from that array passed to *mongo*.

define *xxx*

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

delete *[n1 n2]*

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

device *n*

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

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

draw *x y*

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

ecolumn *n*

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

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

erase

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

errorbar *k*

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

expand *e*

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

format *[...]*

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

grid *[n]*

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

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

halftone *[z1 z2 [contrast]]*

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

hardcopy

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

help *xxx*

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

histogram *[n [f]]*

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

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

input *fspec*

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

insert *[n]*

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

label ...

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

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

****

limits *[x1 x2 y1 y2]*

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

lines *l1 l2*

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

list *[xxx]*

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

location *gx1 gx2 gy1 gy2*

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

ltype *n*

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

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

lweight *n*

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

page *n*

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

pcolumn *n*

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

physical *lx1 lx2 ly1 ly2*

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

playback *[xxx]*

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

points

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

printer *[n [device]]*

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

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

A second optional argument specifies the device name.

ptype *n s*

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

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

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

putlabel *l ...*

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

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

relocate *x y*

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

reset

reset does something which I have not yet decided.

set *v x [op y]*

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

show

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

terminal *[n [device]]*

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

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

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

ticksize *smallx bigx smally bigy*

ticksize determines tick intervals for *box*. *smallx* refers to the interval between small tick marks on the x axis, *bigx* refers to the interval between large ticks, etc. If *big* is 0, the axis routine will supply its own intervals according to the label limits. If *small* < 0, the axis will have logarithmic tick spacing with large ticks at each decade and small ones at each integer. (Despite the axis labels, the limits are still logarithms, e.g. –2 and 2 refers to limits of .01 and 100.)

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

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

window *nx ny k*

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

write *xxx fspec*

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

xcolumn *n*

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

xlabel ...

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

xlogarithm *[n]*

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

ycolumn *n*

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

ylabel ...

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

ylogarithm *n*

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

FILES

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

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

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

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

ACKNOWLEDGEMENTS

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

DIAGNOSTICS

When an error is detected in accessing a General Image file, *gipsmongo* writes a message to the standard error stream *stderr*, and halts with system code 1. Messages generated by GIPS library routines are described in *ihread*(3i). Other possible messages are

- unsupported output device: *name*

BUGS

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

In the ‘‘things to do’’ category, the *contour* command should be completely rewritten to incorporate a contour smoothing algorithm with automatic level-indicator insertion.

NAME

gipstool – display a GIPS image in a Sun window

SYNOPSIS

gipstool [*Suntool options*] [**–** *depth*] [**–KNc**] [**–C** *map*] [**–Iamo** *file*] [**–n** *colors*] [**–r** *buffers*] [**–xy** *size,org,off*] [**–XY** *size,left,right*] [**–zZ** *zoom*] [name]

DESCRIPTION

Gipstool reads a general image file from *name*, and displays the image in a Sun window. If *name* is omitted, it reads the standard input stdin, instead. *Gipstool* then attempts to execute commands from the startup file "˜/*.gipstoolrc*", and, unless an error occurs, becomes an interactive image editor, accepting input from the Sun keyboard and mouse.

There are three principal ways of using *gipstool* to display an image. By default, the entire image is read into memory. If the image is too large to fit within the window area, the arrow keys may be used to roam around. Alternatively, if *gipstool* is invoked with the **–c** option, enough of the image is read to display within the window area. If the image is larger than this, the arrow keys may be used to roam, but this will take longer since fresh parts of the image must be read from disk. Finally, a "cellular" image, one that has been filtered through the *ihgfmt* command, will be treated as if the **–c** option had been used, but, because it is easy to locate sub-units of cellular images, the image can easily be roamed by the arrow keys.

Once the image has been loaded and the startup file executed, the cursor changes from an hour-glass to an arrow, signifying the start of interactive mode–each *gipstool* command is executed by entering a single character or clicking a mouse button. Depressing the RIGHT-hand mouse button will display a menu of all possible commands. Help about any command key is available by pressing and releasing **ˆH** (CTRL–H or backspace), followed by the key in question. The **ˆC** command (the CTRL and C keys held down simultaneously) causes *gipstool* to terminate, while **ˆZ** suspends the program for later resumption.

The remaining keystroke commands are grouped into three broad categories:

Control Keys:
(**ˆA** through **ˆZ**) are reserved for simple image or graphic operations. Control keys are executed immediately–they never result in prompting for further input.

Uppercase Keys:
are used to supply values to internal *gipstool* variables. They cause a message box and reply area to appear in the top left corner of the display. Subsequent input characters, terminated by a carriage return, will be assigned to the particular internal variable. During text entry, the **DEL** key erases the last character typed, and the **ˆH** (backspace) key displays information about the command.

Lowercase Keys:
execute ‘‘higher level’’ *gipstool* functions, some of which may also result in further prompts for text entry.

The startup file "˜/*.gipstoolrc*" contains a series of *gipstool* commands, one per line. The command character itself must begin in column 1. If the command would prompt the user to enter additional text, the latter should follow the command character on the same line. For convenience, commands that consist of control characters may be represented as two characters, a carat followed by an upper or lower case letter. (This convention is also adopted by the > command-logging command.)

OPTIONS

All options of *gipstool* that begin with **–W** are passed directly to the SunView manager. See the manual entry under *suntools*(1) for details.

– *depth* where *depth* can have the value **1** or **8**, selects the depth of frame buffer to use on systems with a choice, e.g. Sun/3–60C or 3/110C. In particular, you *must* use the **–8** flag to display color images on these models.

–C *name*
: only has meaning for 8-bit (color) displays. It specifies the symbolic "*colormap name*" to be given to the *gipstool* window. This is **not** the same entity as the colormap *file* name supplied by the **–a** option–if two copies of *gipstool* are running simultaneously on the same Sun display and they share the same *colormap* name, any change made to one window's color lookup table will **automatically** be applied to the other. If you don't supply a **–C** option, the symbolic colormap name will be taken from the name of the colormap file supplied by the **–a** option. The default colormap name is therefore *bw*.

–I specifies the name of a command file to be read when *gipstool* starts up, instead of the default file named "~/*.gipstoolrc*".

–K Executes a *fork*(2) system call immediately after startup. *Gipstool* will execute in the background. If the image is to be read from the standard input, control will not return to the command or shell that invoked *gipstool* until the image has been displayed.

–N only has meaning for 8-bit (color) displays. It tells *gipstool* to use a full 256-entry color table to display the image. It is equivalent to specifying **–n 256**.

–a *file* specifies the name of a file containing the red, green, and blue output intensities to use when displaying image pixels. *gipstool* searches for this file in the current directory, and then in "/*usr*/*local*/*lib*/*cmap*/∗.*cmap*". The default is "*bw.cmap*" – a linear grey-scale mapping.

–c may be specified when the input image is a disk file. *gipstool* will keep sufficient image pixel values in core to repair window damage, but subsequent image re-sizing or panning will cause the relevant portions of the input file to be re-read. For images that are smaller than the amount of real memory available to *gipstool*, this mode of operation is less efficient than keeping the entire image in storage. It is, of course, necessary to use this flag when working with images that are too large to fit in virtual memory, and it is set automatically when the input image header variable *dtype* has the value "*G*", indicating that the image is stored in the special *cellular* format created by passing an image through the *ihgfmt* filter.

–m *file* specifies the name of a General Image file whose header will be merged with that of the input image before any window sizing or pixel conversion is performed. Alternatively, can be a list of header variables of the form "*name=value*", and those values will be replaced in the input image header.

–n *colors*
: only has meaning for 8-bit (color) displays. It tells *gipstool* to use *n* entries from the colormap table specified by the **–a** option. *n* must be a power of 2 in the range from 2 to 256. When 256 colors are requested, all table entries are used, but these colors will only be displayed when the Sun cursor is within the current *gipstool* window. If 128 colors are requested, they are taken from lines 2, 4, 6, etc. of the colormap file specified by the **–a** option; for 64 colors, from lines 4, 8, 12, etc.

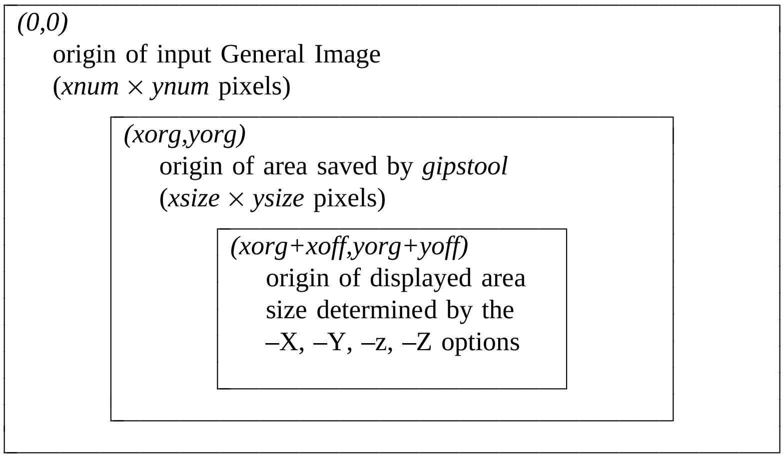
–o *file* specifies that all interactive *gipstool* keystroke and mouse commands will be copied to *file*. This may be changed within *gipstool* via the > function, described below.

–r *buffers*
: tells *gipstool* to allocate a number of buffers (default 32) when reading a *cellular* image of the type generated by *ihgfmt* . See the discussion of the *–c* flag, above.

–x *xsize,xorg,xoff*
–y *xsize,xorg,xoff*
: determine how much of the input image is to be saved in virtual memory, i.e. *xsize* pixels starting at *x*–pixel address *xoff* will be reformatted, and similarly in the *y*–direction. *xsize* and *ysize* default to the entire image area (the input header variables *xnum* and *ynum*). The top left corner of the initial display will be offset by an additional *xoff* and *yoff* pixels, although the entire saved *xsize* × *ysize* image can be displayed via *gipstool*'s pan commands. If *xorg* or *yorg* are omitted, they default to the values of the header variables *xorg* and *yorg*,

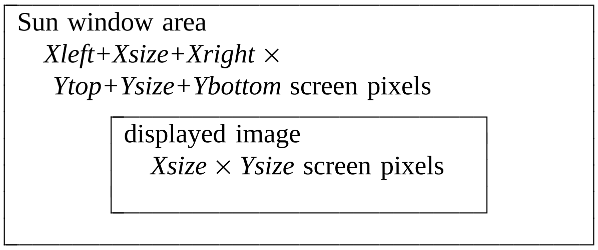
respectively, or to 0 if these too are undefined. *xoff* and *yoff* also default to 0. (If the **–c** flag is specified, *xsize* and *ysize* are disregarded – the full *xnum* × *ynum* image may be displayed via the pan commands, but the initial display is still offset by *xorg+xoff* pixels in the *x*–direction and *yorg+yoff* pixels in the *y*–direction, as measured from the top-left origin).

The following figure shows the three image areas – the area of the input general image, the (possibly smaller) area of image stored within *gipstool*, and the (maybe even smaller) area that can be displayed at any moment within the Sun window. All *(x,y)* values are in input image pixel coordinates:



–X *Xsize,Xleft,Xright*
–Y *Xsize,Xtop,Xbottom*

determine the size of Sunwindow to be used to display the image. The image area will occupy *Xsize* × *Ysize* screen pixels, with additional borders of *Xleft*, *Xright*, *Ytop*, and *Ybottom* screen pixels. By default, *gipstool* uses no borders and sizes the Sunwindow to fit the image exactly. If the input image is too large, the window will exactly fill the screen, leaving the borders as specified. The window and image area are related as follows:



–z *xzoom*
–Z *yzoom*

specify that each image pixel will occupy *xzoom* × *yzoom* screen pixels. **–z** sets a common zoom factor in both *x* and *y* directions (the default is, of course, 1), and **–Z** specifies a separate *y*–direction zoom factor. Because of this zoom feature, you must note carefully the difference between *image* pixels (described by the **–x** and **–y** options), and *screen* pixels (described by **–X** and **–Y**), which may possibly be zoomed.

On 8-bit displays, zooming is achieved by pixel replication, and the zoom factors may be any positive integers. On 1-bit (black-and-white) displays, zooming permits differing pixel values to be displayed as contrasting patterns. Such patterns are currently restricted to zoom factors of 1, 2, 3, or 4, and must be the same in both *x* and *y* directions.

On 1-bit displays only, the special value **–Z0** used in conjunction with the default **–z1** tells *gipstool* to display a one-bit-per-pixel image using an "*error diffusion*" algorithm. This technique is particularly effective in displaying quite subtle variations in areas of low contrast, but it prevents interactive *gipstool* commands from reporting accurate pixel values.

PIXEL VALUES

Gipstool performs a variety of transformations on the input General Image pixels before displaying them on the screen.

If the image has been passed through *ihfft*(1i) so that *dtype=c*, each complex pixel will be replaced by its absolute value before further conversion.

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

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

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

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

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

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

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

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

where *dmax* and *dmin* default to 0 and 256, respectively, if not defined in the input image header.

In many cases, the above steps are purely symbolic – *gipstool* optimizes the conversions whenever possible, and constructs lookup tables. The result, in any case, is that each pixel is transformed to an integer in the range 0–255. Values below 0 are set to zero, those above 255 are set to 255. From here on, 8-bit displays and 1-bit displays take different paths.

COLOR DISPLAYS

The mapping between pixel value and displayed color is determined by the contents of the colormap file specified by the **–a** option. A colormap file should contain exactly 256 lines, excluding comment lines beginning with the ‘‘#’’ character. Each line must contain a triplet of integer red, green, and blue intensity values in the range 0 through 255, separated by tabs or spaces. Additional comments may follow the blue value. The 256 RGB values determine the output color and intensity to be given to pixels with byte values from 0 through 255, respectively.

The actual number of colors to be displayed is determined by the value supplied to the **–n** option of *gipstool*. The default value is 128, permitting GIPS images to share the Sun display with other colored graphics and windows without forcing the Suntools manager to switch colormap tables as the different windows are selected. The 128 values are taken from lines 0, 2, 4, etc. of the colormap file, (ignoring comment lines). Therefore, pairs of pixel values 0 and 1, 2 and 3, and so forth will share a common output RGB value.

MONOCHROME DISPLAYS

Any **–a** option will be ignored. Each pixel is reduced to a number in the range 0 through 255, as outlined above. The pixel is then displayed according to the value of the **–z** option, which may be given the value 1 (the default), 2, 3, or 4. This value determines the size of display area (in screen pixels) to correspond to each input image pixel. For **–z1**, a single output pixel is used, and will be set "*on*" (i.e. black) if the input pixel byte value exceeds 127, and will be "*off*" (white) otherwise. For **–z2**, a 2×2 output pixel is used, For **–z3**, a 3×3 output pixel is used, and so on. The input pixel byte value determines how many of the $n \times n$ output pixels will be set "*on*", as follows:

–z	**Input Value**	**‘‘on’’ Bits**	**Input Value**	**‘‘on’’ Bits**	**Input Value**	**‘‘on’’ Bits**
1	0–127	0	128–255	1		
2	0–63	0	64–127	1	128–191	2
	192–254	3	255	4		
3	0–31	0	32–63	1	64–95	2
	96–127	3	128–159	4	160–191	5
	192–223	6	224–254	7	255	8
4	0–15	0	16–31	1	32–47	2
	48–63	3	64–79	4	80–95	5
	96–111	6	112–127	7	128–143	8
	144–159	9	160–175	10	176–191	11
	192–207	12	208–223	13	224–239	14
	240–254	15	255	16		

Alternatively, if the **–Z0** option is chosen, *gipstool* uses an "*error diffusion*" algorithm to assign on–off values to the one-bit-per-pixel image. The simplest version of this algorithm may be described as follows:

- Reduce the image to an array of elements A_{xy}, each in the range from 0 to 1.
- Inspect the A_{xy} values sequentially, raster by raster. If $A_{xy} \geq 0.5$, set the output pixel *on* (black), otherwise set it *off* (white).
- Compute the error e incurred by this assignment, i.e. $e{=}A_{xy}$ when $A_{xy} < 0.5$, and $e{=}1{-}A_{xy}$ when $A_{xy} \geq 0.5$.
- Distribute this error evenly to those surrounding pixels that have not yet been displayed, i.e. add $0.25e$ to elements $A_{x+1,y}$, $A_{x-1,y+1}$, $A_{x,y+1}$, and $A_{x+1,y+1}$.

THE MOUSE

While *gipstool* is loading a general image and executing the commands in the "*gipstoolrc*" file, the Sun cursor is displayed as an hour glass. If the image is too large to display in the window, the proportion of the total image that is already stored will be indicated by a flashing line across the image. Once the line reaches the bottom of the window, input will be complete, and the cursor will change from an hour glass to an arrow. During interactive operation, *gipstool* uses the mouse and its arrow cursor for three distinct purposes:

To issue a command:
The LEFT mouse button may be assigned to a particular *gipstool* command by means of the **1** (one) command. From then on, clicking the left button is equivalent to typing the key.

To determine the coordinates of a pixel:
Clicking or dragging the MIDDLE mouse button within the image area causes *gipstool* to display the coordinates and data value of the pixel that the cursor is pointing to. It also causes an insert to appear in which the area surrounding the cursor is displayed in zoomed (magnified) form.

To select a line or area:
Several commands, e.g. **ˆO**, **ˆS**, put *gipstool* into "*rubber band*" mode–the cursor changes into a small cross, waiting for the user to hold the LEFT mouse button down and drag the cursor to the required point.

To select commands from a menu:
When the RIGHT mouse button is depressed, a multi-level menu is displayed. All *gipstool* commands may be issued in this manner. The descriptions in the menus are laconic, so for more

details, hit the **ˆH** (backspace) key, followed by the key that you want to find out about. An informative message will appear on the screen. This feature will also work whenever *gipstool* is prompting for keyboard text. Hitting **ˆH** at this moment will generate a message describing the input that *gipstool* expects.

INTERACTIVE COMMAND KEYS

ˆA **Pan to left edge:** The **ˆA** key causes the display to pan to the extreme left edge of the image. If this is already visible, the key has no effect.

ˆB **Pan image left:** The **ˆB** (back) key causes the current image to move to the left in its display window by an amount that may be set by the **P** command. The default is to pan by half the current window size. If the left edge of the image is already visible, the key has no effect. This command may also be executed by the left-arrow key on the Sun keyboard.

ˆC **Cancel Gipstool:** The **ˆC** key terminates *gipstool* . You will be asked to confirm this action.

ˆD **Draw straight line:** The **ˆD** key draws a line from the image mark to the current cursor location. The image mark is reset to the current cursor location.

ˆE **Pan to right edge:** The **ˆE** (end) key causes the display to pan to the extreme right edge of the image. If this is already visible, the key has no effect.

ˆF **Pan image right:** The **ˆF** (forward) key causes the current image to pan to the right in its display window by an amount that may be set by the **P** command. The default is to pan by half the current window size. If the right edge of the image is already visible, the key has no effect. This command may also be executed by the right-arrow key on the Sun keyboard.

ˆG **Run mongo in a separate window:** The **ˆG** key starts the Mongo graphics program in a new window. Mongo will wait for you to send it a histogram or image via the **H**, **J**, or **W** subcommands.

ˆH **Get subcommand help:** The **ˆH** key generates help messages. When typed during a prompt for text entry, you may cancel the current command by pressing the LEFT mouse button, or continue to enter text as requested. When typed from *gipstool* command level, **ˆH** prompts you to enter a key-stroke, after which it displays a description of that command.

ˆI **Display image info:** The **ˆI** key displays a pop-up panel of information about the state of *gipstool*. Most of the items can be changed by putting the cursor in the oval to the left of the item and clicking the LEFT mouse button.

Frame title	the text that appears in the top strip of the frame.
Program version	the current version of *gipstool*.
Original image	the name of the input image file.
Original size	the size recorded in the header of the input image.
In-core size	the size of the pixel array that was actually read from the input image.
In-core origin	the location of the top left in-core pixel relative to the top left of the input image.
Display size	the size of the rectangle that is currently being displayed on the screen.
Display origin	the location of the top left displayed pixel relative to the top left of the in-core image array.
Display margins	the width in pixels of the right hand, and bottom margins, respectively.
Image mark	The pixel coordinates within the in-core image array, of the pixel marker, which may be reset by the **ˆS** command.
Colormap name	the logical colormap name specified by the **–C** option when *gipstool* was invoked.
Colormap file	the name of the current colormap definition file.

Color stretch	the percentage of adaptive stretch applied to the current colormap by the `S´ subcommand.
Linear stretch	the limits of realworld pixel values that are to be mapped to the current colormap range by the `s´ subcommand.
Graphic index	the logical index number under which subsequent graphics will be stored.
Graphic color	the color value assigned to the current graphic index.
Graphic angle	the angle, clockwise from vertical, at which subsequent text will be drawn.
Indices 'on'	the indices of the graphics that are currently displayed.
Indices 'off'	the indices of the graphics that are not currently being displayed.
Roman font	the name of the default text drawing font.
Italic font	the name of the font that will be used to draw *italic* text.
Bold font	the name of the font that will be used to draw **boldface** text.
Symbol font	the name of the font that will be used to draw special symbols.
Text orientation	the current code, set by the **O** command, that determines the relationship between text to be entered into the image and the current cursor location.
Command log file	the name of the file into which is written a log of all *gipstool* commands executed during the current session.
Pixel log file	the name of the file into which the **l** command will write its results.
Tag file	the name of the file that contains tag information about features in the current image.
Left button	the name of the command currently assigned to the LEFT mouse button.
Middle button	the name of the command currently assigned to the MIDDLE mouse button.
Right button	the name of the command currently assigned to the RIGHT mouse button.

ˆK **Clear current graphic:** The **ˆK** key removes all graphics that were created under the current graphic index, including the paint overlay used by the **p** command. The image will be re-drawn without those graphics, which will be discarded. If you merely want to hide those graphics, with the possibility of restoring them later, use the **D** command. Note that the paint overlay cannot be hidden.

ˆL **Repaint image:** The **ˆL** key re-displays the image and all graphics that are currently turned "*on*". This is often necessary after temporary prompts and inserts have clobbered portions of your graphics.

ˆM **Clear all screen inserts:** The **ˆM** (return) key removes all pop-ups and error messages from the image display area. It is also used to terminate all requests for data entry.

ˆN **Pan image down:** The **ˆN** (next) key causes the current image to pan downward in its display window by an amount that may be set by the **P** command. The default is to pan by half the current window size. If the bottom of the image is already visible, the key has no effect. This command may also be executed by the down-arrow key on the Sun keyboard.

ˆO **Set image marker and draw box:** the image marker is reset to the pixel pointed to by the cursor. *Gipstool* enters "*rubber band*" mode, drawing a temporary box with the image marker at one corner and the current cursor position at the other. Pressing any other key removes the box, but the image marker location is preserved.

ˆP **Pan image up:** The **ˆP** (previous) key causes the current image to pan up in its display window by an amount that may be set by the **P** command. The default is to pan by half the current window size. If the top of the image is already visible, the key has no effect. This command may also be executed by the up-arrow key on the Sun keyboard.

ˆQ **Toggle cross-hairs:** Hitting the **ˆQ** key alternately displays and removes a pair of lines that more clearly mark the vertical and horizontal location of the cursor.

ˆR **Scroll image tags:** the **ˆR** key displays the list of currently *tagged* image pixels, i.e. those cursor locations that have been marked by the **T** command or read in from an external tag file by the { command. Use the **R** command to restore the cursor to one of these addresses.

ˆS **Set image marker and draw line:** the image marker is reset to the pixel pointed to by the cursor. *Gipstool* enters "*rubber band*" mode, drawing a temporary line from the image marker to the current cursor position. Pressing any other key removes the line, but the image marker location is preserved.

ˆT **Pan to top edge:** The **ˆT** (top) key causes the display to pan to the top of the image. If already visible, the key has no effect.

ˆU **Pan to bottom edge:** The **ˆU** (under) key causes the display to pan to the bottom of the image. If already visible, the key has no effect.

ˆV **Show pixel inserts:** The **ˆV** The **ˆV** key displays a zoomed image of the pixels surrounding the current cursor location. The size and zoom factor may be reset by the **Z** command. The display will appear at the bottom of the screen and will be continuously updated as the cursor is moved. While holding the SHIFT button down, the arrow keys will move the cursor by one image pixel; if the insert is too large to fit below the image area, it will overlay the image itself.

ˆW **Display color wedge:** A color wedge is displayed. Put the cursor within the wedge and press the right mouse button to display a menu allowing you to select the direction of increasing grey-scale values. Choose "*done*" from the "*Frame* =>" menu to remove the wedge.

ˆX **Set top left of image:** Hitting the **ˆX** key causes *gipstool* to resize the image display area: the current cursor location will become the top left corner of the new display. The bottom right corner may similarly be redefined by the **ˆY** key.

ˆY **Set bottom right of image:** Hitting the **ˆY** key causes *gipstool* to resize the image display area: the current cursor location will become the bottom right corner of the new display. The top left corner may similarly be redefined by the **ˆX** key.

! **Execute a UNIX command:** Enter a single command line to be executed by the UNIX */bin/sh* shell.

. **Locate image pixel:** The "**.**" key prompts you to specify a pair of pixel addresses, (*x* followed by *y*), and then positions the cursor on that pixel, possibly the image if necessary to bring that pixel within the display area. The addresses may be specified in several formats:

- a floating-point number denotes a realworld value
- an unsigned integer denotes an image pixel address
- a signed integer denotes an image pixel address relative to the current cursor location
- an integer preceded by "**:**" denotes a (possibly zoomed) display pixel address
- an integer preceded by "<" or ">" denotes a display pixel address relative to the current cursor location.

/ **Display or set GIPS variable:** The "/" key causes *gipstool* to prompt you to enter the name, and, optionally, a replacement value, for a GIPS header variable. If you only enter the variable name, *gipstool* displays the variable type, description, and current value. If you reply *name*=, the variable is deleted. If you reply *name*=*value*, the variable is set to *value.*

Begin a comment: The text that you enter, up to and including the first newline character, will be treated as a comment. If command logging is in effect (the > command), your comment will be copied to the log file.

< **Read command file:** Enter the name of an existing file containing *gipstool* commands, e.g. one that was created by the > command. If the name begins with a "|" character, interpret the remainder of the text as a shell command to be executed and its output to be a series of *gipstool*

commands to be executed. If an error is detected during any command, input will cease and *gipstool* will return to interactive input.

> **Begin command logging:** Enter the name of a file to receive a copy of your subsequent *gipstool* commands. If you enter a null line, any current logging will be terminated. If the name begins with a "|" character, interpret the remainder of the text as a shell command to be executed and its input to be the subsequent *gipstool* commands. To append the logging information to an existing file, enter a second > before the file name. In this latter case, if you omit the file name, *gipstool* will use the most recently selected log file.

1 **Reassign left button:** The **1** key causes *gipstool* to prompt you to redefine the command to be assigned to the LEFT mouse button. Your first keystroke will become the new command. Subsequent keystrokes, up to a carriage return, specify the default text to be entered when that command is invoked from the mouse button. The. LEFT button is initially unassigned.

A **Set text angle:** the angle (in degrees counter-clockwise from horizontal) at which to draw subsequent graphic text. The default is 0, i.e. upright and horizontal.

B **Set background color:** a triplet of integers in the range 0–255, to become, respectively, the new red, green, and blue intensities for background pixels. The initial background intensities are determined by the zero entry in the current colormap.

C **Set graphic color value:** an integer in the range 0–255 to specify the colormap lookup value for subsequently created graphics. Once a graphic entity has been created, it is no longer possible to change its lookup value, although you may, of course, change its displayed color by altering the current colormap.

D **Delete current graphic:** the image will be redrawn without any graphic items that were created under the specified graphic index. These items can subsequently be restored by executing the **G** command.

F **Set drawing font:** the name of the font file that defines the shape and size of characters used to create subsequent graphic text. If a relative pathname is specified, *gipstool* looks first in the current directory, then in "*/usr/lib/vfont*", and finally in "*/usr/lib/fonts/fixedwidthfonts*". Optionally, the the font name may be preceded with a style indicator, a single letter **r**, **i**, **b**,**or** **s**, or a corresponding number in the range 0–3, indicating that the font name that follows is to be used respectively for *roman*, *italic*, *bold*,*or symbol* text only (as selected by **\f** codes in the text string).

G **Set current graphic index:** the specified number, an integer in the range 0–255, becomes the current graphic index. Subsequent graphic text will be saved under this number, and can be either permanently deleted or temporarily removed from the display by the **ˆK** and **D** commands, respectively.

H **Write image histogram:** the current colormap and a pixel histogram are written to the specified file, as a 256 row by 7 column *General Image*.

Row	*Contents*
1	pixel number (0 through 255)
2	corresponding realworld pixel value
3–5	corresponding rgb colormap value
6	number of pixels with this value
7	number of pixels with this value or less

The histogram is taken over the sub-image defined by the image mark and the current cursor location. Pixels with the same value as the *dnull* image header keyword are not counted. If you enter a null line, the data is sent to the socket, from which it can be read and displayed by *mongo* commands.

I **Write a sub-image:** the rectangular sub-image whose corners are defined by the current pixel mark (set by the **ˆS** or **ˆO** commands) and the current cursor location, is written to the specified disk file. If the name is omitted, the sub-image is written to the socket.

J **Write line trace:** the line of pixel values stretching from the current pixel mark to the current cursor location is written to the specified disk file as an $n \times 7$ general image whose rows represent the *n* individual pixels in the line, and whose 7 columns represent the following:

Row	*Contents*
1	pixel number
2	pixel x–coordinate
3	real-world x–coordinate
4	pixel y–coordinate
5	real-world y–coordinate
6	integer pixel value
7	real-world pixel value

If the file name is omitted, this image is written to the *gipsmongo* socket.

K **Write image colormap:** the current colormap is written to the specified output file. The output file will reflect the current *stretch* status set by a prior **S** command.

L **Begin pixel logging:** enter the name of the output file to which output from the **l** command will be appended. If logging is in progress, enter a null line to close the file, or enter another file name to close the original file and open a new one.

M **Read new colormap file:** the file specifies the translation between pixel values and displayed colors. *Gipstool* first looks in the current directory, and then in "*/usr/local/lib/cmap*". The new colormap table is applied to the current window **and** to all other windows that share the same colormap name (see the description of the *gipstool* **–C** option, above).

N **Set numeric precision:** a digit in the range 1 through 14 specifying the number of decimal digits of precision desired for all floating-point numbers displayed by *gipstool*.

O **Set text origin code:** an integer in the range 0–8, specifying the relationship between the mouse cursor location and the origin of text strings created by the **t** command, as follows:

1	2	3
4	5	6
7	8	9

Thus, if the origin is **1**, text will be written to the right and below the cursor location, whereas origin **0**, will cause the text to be centered about the cursor, and so forth.

P **Set pan increment:** the increment, a non-negative integer, defines the number of screen pixels by which the display will be moved by one of the incremental panning commands: **ˆB**, **ˆF**, **ˆN**, and **ˆP**. The initial value is **0**, a special value that *gipstool* interprets as denoting half the current width (or height, if panning vertically) of the displayed image.

R **Recall a tagged location:** prompts for a tag name. This should be one of the names supplied by previous **T** commands to remember image locations. The result will be to reset the image cursor to point to that pixel location, possibly panning the image to do so. If no tag name is supplied, the last accessed image tag will be restored.

S **Stretch colormap:** applies a contrast-enhancement to the colormap, based on the distribution of pixel values in the stored image. This distribution is computed whenever the stored image is altered: usually, this occurs only during the initial load phase, but if you have specified the **–c** flag, the histogram will be recomputed whenever you resize or pan the displayed image.

No new colormap intensities are created; rather, the transformation selectively reassigns color/intensity values to increase the contrast between most-likely pixel values. At the same time, contrast will be lost between less-likely values. *Gipstool* prompts for an integer percentage in the range 0–100, setting the ‘‘strength’’ of the transformation. 100% denotes a ‘‘maximal’’ stretch, whereas 0% restores the original colormap values. Use of the **s** command will reset the stretch to 0%.

T **Set image tag:** prompts for a tag name. The current cursor location is then remembered under this tag name, and may be recalled via the **R** command. The list of current tag names may be scrolled through by means of the **ˆR** command.

V **Set foreground color:** a triplet of integers in the range 0–255, to become, respectively, the new red, green, and blue intensities for foreground pixels. The initial foreground intensities are determined by the zero entry in the current colormap.

W **Set paintbrush width:** sets the *x* and *y* dimension of the area that will be painted by the next **p** command.

Z **Set zoom pixel size:** supplies three parameters for the **ˆW** (zoomed window) command: the first determines the number of screen pixels that will be displayed in the zoomed window, the second specifies the additional zoom factor to be applied. The third specifies the color of the box and cross-hairs. If the screen image is already zoomed (i.e. *gipstool* was invoked with the **–z** or **–Z** options), the result will be a doubly zoomed sub-image.

a **Draw both axes:** prompt for changes to axis-drawing parameters (see below), and draw the axes, using the current graphic settings i.e. color, font, index, and numeric precision, specified by **C**, **F**, **G**, and **N** commands, respectively.

f **Set frame label:** prompts for a text string to replace the label in the stripe at the top of the image display. If the string is null, the stripe will be removed.

h **Display header item:** the header item of this name will be converted to ASCII and inserted in the display as a graphic item according to the current graphic settings, i.e. angle, color, font, index, and origin, specified by the **A**, **C**, **F**, **G**, and **O** commands, respectively.

l **Log pixel value:** the coordinate and data values of the current pixel are appended to the current pixel-logging file (opened via the **L** command). The format is specified by the user:

format	*action*
%x	replace with the x–pixel coordinate
%rx	replace with the realworld x–coordinate value
%tx	replace with the x–axis title

and similarly with **y** or **d** in place of **x .**

m **Draw non-local grid lines:** The **m** command is only valid for images in non-local projections, i.e. those created by the *ihgrid* and *ihgeom* programs, and therefore contain a header variable called *mapping* that defines the relationship between their *x* and *y* pixel addresses and their real-world coordinates. The spacing of the grid lines is given by the *xaxis* and *yaxis* header variables.

p **Paint pixel:** sets the pixel that is pointed to by the cursor to the current graphics drawing color. When the **p** command is assigned to the left mouse button, the painting will take place wherever the cursor is dragged with the left button held down, provided a **ˆS** or **ˆO** command isn't in progress. Alternatively, if the **SHIFT** key is held down at the same time, the pixels will be cleared instead. The size of the area that is drawn or cleared may be changed by the **W** command.

s **Enter colormap stretch points:** applies a linear contrast-enhancement to the colormap, specified by the minimum and maximum *realworld* values supplied. No new colormap intensities are created; rather, the transformation reassigns the pixel values between the limits to the available colors. Use of the **S** command will reset the linear stretch limits to the values of header variables *dmin* and *dmax*, respectively.

t **Enter text string:** the string will be converted to a graphic item according to the current graphic settings, i.e. angle, color, font, index, and origin, specified by the **A**, **C**, **F**, **G**, and **O** commands, respectively. The size and style of the text string may be changed by means of *troff*-style escape sequences embedded in the string. See *comlabel*(3i) for more details.

u **Undo last graphic:** the graphic most recently defined under the current graphic index is removed from the display. In the case of commands that generate multiple graphic items, e.g. axes, the **u** command removes one item at a time.

v **Enter cursor location** the *x* , *y* , and/or *data* values of the pixel pointed to by the cursor are converted to a graphic item according to the current graphic settings, i.e. angle, color, font, index, and origin, specified by the **A , C , F , G ,** and **O** commands, respectively. The number and order of the values are specified by the user: enter **x , y ,** and/or **d** in any order, and each optionally preceded by **r** signifying that the realworld value is to be used instead of the image value.

w **Write overlay:** write the 8-bit overlay plane to the specified disk file in GIPS format. This image will contain pixels set by execution of the **p** command. The output file will be overwritten by the overlay.

x **Draw x–axis:** prompt for changes to *x*–axis drawing parameters (see below), and then draw the *x*–axis.

y **Draw y–axis:** prompt for changes to *y*–axis drawing parameters (see below), and then draw the *y*–axis.

{ **Read tag file:** read tag definitions from an external file. The tags describe locations within the image. Null lines and lines that begin with # are discarded. Each remaining line must begin:

tag_name xreal yreal comment_string

where *tag_name* must contain no whitespace, and *xloc* and *yloc* are the realworld tag coordinates. The tag definitions are appended to the current tag list. They may be saved again by the } command.

} **Save tag file:** writes the current tag list to the specified file. A tag file can be read again by the { command.

AXIS DRAWING PARAMETERS

The *gipstool* axis-drawing commands **a**, **x**, and **y** can draw axis lines, scale marks, tick marks, scale annotation, axis labels, and grids. In addition, the **m** command draws grid lines on images with non-separable coordinates. The resulting graphic items will be displayed in the current graphic color and index, as displayed by the *gipstool* **ˆI** command.

The axis labels are drawn using the current definitions of the header variables *xtitle*, *xfont*, *ytitle*, and *yfont* of the input image. Both title and font must be defined or the title will be suppressed. The orientation is implied by the font file names: names ending in "*r*" denote ‘‘rotated’’ (i.e. vertical) titles, otherwise they will be horizontal. No intermediate angles are possible.

The remaining axis graphics are constructed from the *xaxis* and *yaxis* header variables of the input general image. Each variable should contain a list of sub-fields of the form "*item=value*", separated by blanks or tabs, for example:

xaxis="*mark=10 tic=1 font=I.6 by=15.0 place=tbci*"

When the **a**, **x**, or **y** command prompts you to "*Enter updated axis parameters:*", you may specify new sub-field values for *xaxis* and *yaxis*; i.e. replies to **x** command will update *xaxis*, replies to **y** will update *yaxis*, and replies to **a** will update both *xaxis* and *yaxis*. For instance, if you execute the **x** function, specifying "*font=B.6*", *gipstool* will generate an *x*–axis using the "*B.6*" font (6–point Hershey bold) for scale annotation. In addition, *gipstool* will alter its stored value of the *xaxis* header variable to include the new definition of the "*font=*" sub-field.

Here are the definitions of the possible *xaxis* and *yaxis* sub-fields:

mark= the maximum number of scale labels that may be drawn along the axis. If not specified or set to 0, the default is 1000.

tic= the number of tic marks to be inserted between each scale interval.

font= the font in which to write the scale annotation. *Gipstool* searches for the font file in the current directory, then in "*/usr/lib/vfont*", and finally in "*/usr/lib/fonts/fixedwidthfonts*".

place= one or more characters in the string "**lcriotmb**" determining to whether the titles and scale notation should be **L**eft, **C**enter, or **R**ight justified, inserted **I**nside or **O**utside the image area, and placed at the **T**op, **M**iddle, and/or **B**ottom of the image.

loc= defines the meaning of ‘‘middle’’ titles – i.e. within *xaxis*, *loc* specifies the *y*–value at which the *x*–axis is to be drawn.

from=
to=
by= together define the values at which the scale annotations are to be drawn. If specified in floating-point form, realworld values are assumed. Otherwise, they are assumed to represent input image pixel values. The same notation is used for specifying the **loc=** sub-field.

line= an 8-bit mask used to construct lines connecting the top and bottom of the image at each scale mark. If "*line=0*" (the default), no connecting lines will be drawn.

form= a non-default format in which to draw the scale annotation. The format is that of the *printf*(3S) function, and should contain no embedded white-space. The default is "*%.6g*" unless *scale=d* or *scale=h*, in which case the defaults are "*%d\013%d’%.12g’’*" and "*%dh%dm%.12gs*" respectively. Therefore, there must be at least three percent fields in these cases, and one in all others.

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

scale= a code letter which causes axis scale numbers to be formatted as:

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

FILES

/usr/local/lib/cmap/∗.cmap	colormap files
/usr/local/tables/cprofile	image header definition table
/usr/local/tables/gipstool.menu	command definition script

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

"*Guide to GIPS*" and "*A GIPS Tutorial*", by Peter Ford, MIT Center for Space Research, 1984.
gipsmongo(1), ihs(1), iged(1), suntools(1), comlabel(3)

ACKNOWLEDGEMENTS

The technique of "*error diffusion*" was first brought to my attention by John Tonry, the author of the *mongo* program.

DIAGNOSTICS

When an error is detected before the SunView frame is created, *gipstool* writes a message to the standard error stream *stderr*, and halts with system code 1. Messages generated by GIPS library routines are described in *ihread*(3). Other possible messages are

- bad –z value: *value*
- bad x–pixel origin, size, or offset: *value*
- bad x–view size, or left/right border: *value*
- bad y–pixel origin, size, or offset: *value*
- bad y–view size, or left/right border: *value*
- can’t create image sub-window
- can’t create tool
- can’t open image sub-window
- cannot create saved pixrect
- multiple input file(s): *name*
- x–limits outside image

- xzoom != yzoom not implemented for depth=1 display
- y–limits outside image

BUGS

The axis drawing commands **a**, **x**, and **y** have some bizarre problems. Although it is intended that they should generate "reasonable" scale numbers, they frequently need a careful choice of *from*, *to*, *by* parameters to work properly.

The text-entry pop-up is sometimes not fully repainted, particularly on color screens. The user can force a repaint by selecting the *Redisplay* action from the frame menu.

NAME

git – list the header or pixel values of a General Image

SYNOPSIS

git [**–achns**] [**–p** *precision*] [*name*]

DESCRIPTION

The *git* command converts the pixel values of a General Image to ASCII numbers, writing them to the standard output *stdout*. If *name* is specified, the image is read from the file of that name; otherwise, it is read from the standard input stream *stdin*. If the variable *dnull* is defined in the image header, repeated null pixels are represented by the notation "**[***n***]**", where *n* denotes the repetition count. The following flags and options apply:

–a applies to Fourier-transformed images with complex pixels (*dtype=c*). The pixel values are converted to real by taking their norms. Otherwise, they will be displayed as "**(***cos*,*sin***)**", representing the Fourier *cosine* and *sine* components.

–c compress repeated output pixel values, i.e. display them as "*count*∗*value*".

–h also copies the General Image header to *stdout*, one header item per output line. The image values follow this header.

–n turns on the **–h** flag, writing the General Image header to the standard output stream *stdout*, but suppressing the output of pixel values.

–p *prec* specifies the number of decimal digits of precision to be kept when converting floating-point pixels. *prec* must be an integer in the range 0 through 12, the default being 6. Non-zero pixel values that would be rounded to zero under the chosen precision are output as "˜ **0**".

–s inserts two newline characters after each output raster, and displays the raster number. Normally a single space separates each pixel value and a single newline signals the end of each raster.

FILES

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

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

gin(1), ihs(1), ihread(3)

DIAGNOSTICS

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

- dtype=*code* conflicts with –a
- illegal –p value: *num*

NAME

iged – interactive graphic-plane editor

SYNOPSIS

iged [*–command* ...]

DESCRIPTION

When invoked with arguments, *iged* is said to be in *pipeline* mode: it expects to read a Generalized Image header file from the standard input stream *stdin*, and its arguments are then interpreted as a series of subcommands, each of which consists of a series of words, the first of which must begin with a dash followed by a command character, e.g.

iged –g 1 –c red –v xfont=R.10 –a < image.one

If *iged* is invoked without arguments, it is said to be in *interactive* mode, and reads commands from the standard input stream *stdin*. The command format is identical to that in *pipeline* mode, except that each command must end in a "newline" character, and the initial dash is optional (except in subcommand macros – see the ´"´ command, below).

Iged first tries to read the file ".*igedrc* " in your home directory. If this file exists, its contents are interpreted as *iged* subcommands and are executed as if they had been issued at an interactive terminal. Some subcommands are not accepted – e.g. those that redirect input and output streams and those that generate listings.

Iged can maintain temporary and permanent updates to several graphic and image planes simultaneously, but for ease of command syntax, only one of each is *current* at any one moment. The **g** and **i** subcommands define, respectively, the current graphic and image. They also cause that plane to be displayed.

If you get stuck within interactive*iged*, on-line help is available by typing "**?**" – this displays a list of subcommands – or type "**??**" – which tells you about *iged* command syntax. In addition, a number of subcommands have "help" modes, e.g. the "**l**" subcommand (entered without arguments) displays a number of useful internal variables, "**f**" tells you which character-drawing fonts are currently loaded, and "**f**∗" lists all possible fonts.

Many *iged* sub-commands take arguments that refer to a particular pixel location within an image or graphic. Both *x* and *y* coordinates must be specified, in that order; there are seven ways of specifying either *x* or *y*, the last two of which are only accepted in interactive mode:

- An unsigned integer, representing a pixel address in the currently displayed image. Unless *iged* has read a General Image header (e.g. via the "**h**" subcommand, or from *stdin* in pipeline mode), this address is identical to the absolute x/y display address described under the next item below. Otherwise, it is an address relative to the top left corner of the image as defined by the respective *xorg* or *yorg* header variable.
- A colon followed by an unsigned integer represents an absolute pixel address in the current display. *x*–addresses increase from left to right, *y*–addresses from top to bottom; the top left pixel is "**:0:0**".
- A floating-point number (i.e. a numeric string containing a decimal point or the letter "e" or "d"), representing a "realworld" *x* or *y* coordinate value.
- The character @ representing the current *x* and *y* location of the display target. Unlike the **%** specification below, the user has no opportunity to re-position the target.
- The character < or > followed by an unsigned integer, representing that many x (y) pixels respectively to the left of (above) or to the right of (below) the current target location. *iged* gives the user no opportunity to reposition the target.
- The character **%** represents the *x* and *y* location of the display pointer. *Iged* will prompt you with a # character and you must then re-position the display pointer and hit a carriage return. Typing any other key before the carriage return will cause the *iged* sub-command to terminate immediately.
- A signed integer represents an offset from the current position of the display pointer (remember that

the x-axis coordinates increase left to right, and the y-axis coordinates increase *top to bottom*). *Iged* will prompt with a # in the manner described above.

Aside from the metacharacters @ and **%**, which must appear by themselves, the other formats may be freely mixed, i.e. " **-10** >**21** ", " **101.2 :4** ", etc., but remember that "**%**" and signed integers are only permitted in interactive mode.

Iged maintains a knowledge of real-world coordinates by reading General Image headers. Only one header may be *current* at any one time, although you can define special compound commands called *macros* which will change the *iged* environment with only a couple of keystrokes. These are described under the ´"´ subcommand, below, and may be used to switch between image headers.

In the following table, the subcommand character is shown in **boldface** and its skeleton argument list is *italicized*. All meta-characters in filenames will be expanded by *csh*(1) before execution. (The choice of shell may be changed via the `**!**∗´ command.)

A "*%*" sign in the following command definitions indicates the use of any of the 7 acceptable kinds of target designation, including a single percent sign itself. Square parentheses around an argument indicate that its presence is optional. Ellipsis, i.e. "...", indicates that one or more repetitions of the preceding argument are also accepted.

a *[parms]* — **Generate x- and y-axis scales and titles**, using header variables *xtitle*, *ytitle*, *xfont*, *yfont*, *xaxis*, and *yaxis*. Sub-fields in *xaxis* and *yaxis* may be overridden by *parms* (see under *Axis Specifications*, *below*). Any changes that you specify will update subfields in the incore *xaxis* and *yaxis* variables. If you wish to update *xaxis* and *yaxis* separately, use the **x** and **y** subcommands instead.

b — **Enter block-mode.** When executing in *block* mode, *iged* refreshes the graphic plane only when terminating, when changing planes, or when an explicit **w** command is issued to the plane. Block mode is the only mode permitted when *iged* takes its input from the standard input stream. (cf. single-command mode, the **s** subcommand).

c *[color]* — **Change the color of the currently displayed graphic.** If *color* is omitted, *iged* displays the current color setting. The following 16 'colorcode' mnemonics are recognized (they may be entered in upper or lower case):

BLA	black	LPI	light pink
BLU	blue	MAG	magenta
CYN	cyan	ORN	orange
DGR	dark green	PNK	pink
DTU	dark turquoise	RED	red
FGR	forest green	TUR	turquoise
GRN	green	LOR	light orange
YEL	yellow	WHT	white

d *%...%* — **Draw an open polygon**, i.e. a line is drawn from one screen location to the next. If more than two locations are specified, the line is continued to the third point, and so on. The type of line is specified by the "k" command.

e *%%* — **Insert a data wedge into the current image.** A box is inserted in the current graphic, and the corresponding region of the current image is filled with pixels in ascending value order. The orientation of the values within the box is determined by the order in which the extremities of the box are defined:

```
1-----+   2-----+   +-----2   +-----1
|     |   |     |   |     |   |     |
| →   |   | ←   |   | ↑   |   | ↓   |
|     |   |     |   |     |   |     |
+-----2   +-----1   1-----+   2-----+
```

The *1* and *2* denote the locations of the first and second pixel addresses specified, and the arrows denote the direction of increasing wedge value. If you make a mistake, you can remove the wedge with the **U** command, and undo the graphic border with **u**.

f *[name]* **Change the current title-drawing font** – *name* specifies a binary font file in *vfont*(5) format. If a relative pathname is used, *iged* first searches for a file of this name in the current directory, then in /*usr*/*lib*/*vfont*. Names ending in "*r*" are assumed to be rotated (vertical) styles. If the font is of "Roman" type, *iged* will also attempt load the italic, bold, and/or symbol fonts of that type and size. The alternate fonts may then be selected by troff-style in-line escape codes as described in *comlabel*(3). Alternate fonts may also be specified individually by the **F** subcommand. When *name* is omitted, *iged* displays the name of the currently used font. A listing of all available fonts in /*usr*/*lib*/*vfont* may be obtained by entering "*f* ∗", and any characters following the "∗" are interpreted as shell metacharacters used to select a subset of these names, e.g. "*f* ∗ *nonie*∗" displays all of the "*nonie*" fonts.

g *[#]* **Change the "current" graphic plane.** # specifies which graphic plane is to receive output. The contents of the current plane are saved and the new plane is restored (if previously saved in this *iged* session). If # is omitted, *iged* displays the numbers of the current plane (marked with a "∗") and all saved planes. To delete a saved plane, enter the negative of its number.

h *[name]* **Read a General Image header.** *Name* is expanded by *csh*(1) and, assuming that a file of that name is found, its GIPS header is read and stored. Subsequent knowledge of "realworld" coordinate and pixel values will be determined by this header. If *name* is omitted, *iged* displays the file name of the currently stored GIPS header.

i *[#]* **Change the "current" image plane.** Image plane number # will supply data values for subsequent **e**, **n**, **H**, **N**, and **U** commands. Initially, image plane 1 is current. If *name* is omitted, the current image plane number is displayed.

j *% item* **Insert header item into current graphic.** The value of incore header variable *item* is (if necessary) converted to a character string and that string is written into the current graphic at the specified pixel address.

k *[linetype]* **Change line drawing pattern.** *Linetype* specifies an 8-bit pattern to be repeated by all line-drawing subcommands, e.g. "**d**". *linetype* may be an unsigned decimal integer, an octal integer (with leading ’0’), or a string of 8 characters. When numeric, the line type is taken from the last 8 bits of the binary number; when alphanumeric, the bit pattern is specified by alphanumeric (bit=1) and non-alphanumeric (bit=0) characters in the string. e.g. "*k 23*", "*k 027*" and "*k ---x-xxx*" are equivalent. If *linetype* is omitted, iged displays the currently defined line pattern.

l *[item ...]* **Display incore header variables or GIPS statistics.** *item* represents the name of a GIPS header variable, whose value is displayed. If *name* is omitted, *iged* displays the current image and graphic numbers, and the most recent GIPS headers that have been read or merged. If "∗" is used for *item*, the entire header is displayed. If the "∗" is followed by a valid file name, that name is shell-expanded by *csh*(1), and the header is written to that file.

m *[name]* **Merge additional GIPS header fields.** *Name* is expanded by *csh* and, assuming that a file of that name exists, its GIPS header is used to augment the header that was last read via the "**h**" command. If *name* is omitted, *iged* displays the name of the last GIPS header file thar was merged.

n *%* **Display an image pixel value.** The 8-bit value of the specified pixel in the current image plane is transformed according to the information in the GIPS header and is displayed (along with its *x* and *y* coordinates), in its screen and realworld units, as follows:

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

where *xpixel* represents the pixel coordinate, *xmin* and *xmax* the realworld limits spanned by the pixel, etc. Realworld fields outside the display limits are reported as **MAX** or **MIN**, or, if no header has been read, as **NUL**.

o *[#]* **Change relative origin of graphic text** written into the current graphic by subsequent **t** commands. Values 0 thru 8 determine the text origin according to the following table

1	2	3
4	0	5
6	7	8

e.g. "*o 0*" (the default value) causes text to be centered about the chosen location, "*o 1*" causes it to be written wholly to the bottom right of it, etc. For rotated fonts (with names ending in 'r'), left, right, top, etc. refer to the rotated characters themselves, i.e. *top* indicates the direction to the *right* of the screen. If # is omitted, *iged* reports the current origin code.

p *[#]* **Change floating-point precision** (the maximum number of significant digits displayed) of all subsequent output of floating point quantities. If # is omitted, the current precision is displayed.

q **Terminate the** ***iged*** **session.** When *iged* is used interactively (i.e. reading commands from the standard input stream), the command must be repeated to become effective.

r *[name]* **Read a graphic plane.** File *name* overwrites the current *iged* graphic. If *name* is numeric, it is assumed to refer to the graphic plane of that number; if omitted, the current display graphic is read back into *iged*; otherwise, *name* is expanded by *csh* and the first matching file is read.

s **enter single command mode.** Updates to the current graphic will be displayed as they are made. (cf. the **b** subcommand).

t *% [text]* **Position target or write graphics text** – if *text* is omitted, position the display target at the specified pixel. If this is not currently in view, the image will be roamed to do so.

If *text* is supplied, that text string is written into the current graphic plane using the current font, in a location whose relation to the specified pixel is set by the **o** subcommand.

u **Undo the last change** applied to the current graphic plane. Another **u** immediately following will undo the undo.

v *item [= value] ...*

Update incore header variables – variable *item* is assigned a new value. If *value* contains blanks, enclose it in double quotes. If *value* is omitted, the variable is deleted. If "= *value*" is omitted entirely, the variable type and description of *item* are displayed. Multiple items may be processed on the same line, but only one at a time may be deleted. Also, it is only possible to delete the most recent *history* item from an incore header, but any number may be added.

w *[name]* **Write the current graphic** – to file *name* (expanded by *csh*). If *name* already exists, use "*w ! name*" instead. If *name* is numeric, the current graphic is written to the graphic plane of that number. If *name* is omitted, the current graphic is updated.

x *[parms]* **Draw x-axis** lines, scale marks and numbers, and title, using header variables *xtitle*, *xfont*, and *xaxis*, with the latter updated (for the remainder of this *iged* session) by any *parms* that you specify (see *Axis Specification*, below).

y *[parms]* **Draw y-axis** lines, scale marks and numbers, and title, using header variables *ytitle*, *yfont*, and *yaxis*, with the latter updated (for the remainder of this *iged* session) by any *parms* that you specify (see *Axis Specification*, below).

z *name %...%* **Report matching General Image pixels.** *Name* is C-shell-expanded and the General Image file of that name is scanned for pixels whose *realworld* x- and y-coordinates overlap the coordinates of the pixel(s) specified in the **z** subcommand.

C **Clear the current graphic** plane to binary zeroes.

D *%...%* **Draw a filled polygon.** The area enclosed by the specified series of pixel locations will be filled within the current graphic. If the starting and ending locations are not identical, the figure will be closed automatically.

F *[type [name]]* **Define alternate fonts.** *Name* specifies an alternate font definition file (cf. the **f** subcommand). If a relative pathname is used, *iged* first searches for a file of this name in the current directory, then in "*/usr/lib/vfont*". Names ending in "r" are rotated (vertical) styles. *Type* must be one of the following words or abbreviations:

TYPE	ABBREVIATION	ESCAPE CODE	DESCRIPTION
roman	R or r	\fR or \f1	Major (default) font
italic	I or i	\fI or \f2	Italic font
bold	B or b	\fB or \f3	Bold font
symbol	S or s	\fS or \f4	Symbol font

The alternate fonts may then be selected by the troff-style in-line escape codes as described in *comlabel*(3). When *name* is omitted, *iged* displays the name of the current alternate font of that type.

G *# oper [˜]* **Merge graphic planes.** The specified saved graphic plane number # is merged with the current graphic according to the operator *oper*, where *oper* must be one of the following:

|	logical ORing (inclusive OR)
&	logical ANDing
ˆ	logical XORing (exclusive OR)
=	bitwise copying

In addition, the "˜" operator may follow one of the above, indicating that the source should be the logical NOR of the saved graphic. The saved graphic plane is never altered.

H *[code]* **Start or stop automatic histogramming.** *Code* specifies one or more automatic histograming and pixel display functions to be executed by all subsequent line-drawing and area-filling subcommands, until canceled by a subsequent **H** command without any *code*. If *code* is "**s**" or "**t**", pixel values will be accumulated and written as a histogram table when histograming mode is canceled. The modes differ in that "**t**" histograms are displayed down the screen, whereas "**s**" histograms are displayed across the screen. If *code* is "**v**" or "**r**", pixel and coordinate values will be displayed whenever the corresponding graphic bits are set in subsequent *iged* commands. "**r**" differs from "**v**" in that *truncated* realworld units are displayed, rather than upper and lower data limits. If *code* contains an "**n**", null pixels will not be listed. A subsequent "**Hz**" subcommand will resume listing null pixels. "**s**" and "**t**" and "**v**" (or "**r**") mode may be active simultaneously.

L *% name [scale]* **Insert a graphic logo.** *Name* is expanded by *csh* and the file of that name is opened. The contents are translated into a bit pattern to be written into the current graphic at the specified location. Each newline-terminated line of *name* contributes to a single graphic raster. Within the line, blanks turn corresponding graphic bits *off* and non-blanks turn them *on*. Tabs will be expanded into spaces in the manner of the *expand*(1) command. Specifying *scale* as a positive floating-point value will cause the resulting graphic to enlarge or shrink accordingly. The primary use of this sub-command is to insert a logo or other often-drawn figure into a graphic.

N *%* **Report a truncated pixel value.** The 8-bit value of the specified pixel in the current image plane is transformed according to the information in the GIPS header and is reported in pixel and realworld units, as follows:

x: *xpixel xreal* **y:** *ypixel yreal* **d:** *dpixel dreal*

where *xpixel* represents the pixel coordinate, *xreal* the realworld value, etc. Unlike the **n** subcommand, which displays the upper and lower realworld limits, **N** reports a single average "truncated" value for the realworld x, y and data values.

O *% %* **Draw a circle.** The first location determines the center, the second a point on the circumference of a circle to be inserted into the current graphic using the *linetype* specified by the **k** subcommand.

P *% n* **Change an image pixel value.** The value of the specified pixel in the current image is changed to *n*, an integer in the range 0 through 255.

R *[count] command*

Repeat another subcommand. The subcommand "*command*" is executed repetitively "*count*" times. If *count* is omitted, *command* is executed 100 times, or until it encounters an error. When repeating shell commands, i.e. "**R ! ...**", typing **CTRL-C** will terminate the loop; similarly, when repeating a subcommand that prompts with "#" for you to reset the display target, i.e. "**R n %**", hitting any other key before carriage-return will also exit from the repeat-loop. Some commands may not be executed repetitively, e.g. **R** itself, and this also applies to macros containing those commands.

S *[code] [%...%]* **Draw a smooth curve** passing through the specified pixels. *Code* represents one of more of the following options:

m*n*	select interpolation method *n*: **0**: spline (default), **1**: quartic, **2**: circular.
p*n*	select point-smoothing density *n*: **0**: coarse, **1**: medium (default), **2**: smooth.
q	don’t update the display after each segment.
r	prompt repetitively for next cursor location.
w	join last point to first.

and, unless option "**r**" is specified, these must be followed by at least 3 cursor location fields.

T *% code...* — **Insert pixel value into graphic** – the value of the chosen pixel is converted to *truncated* realworld units. *Code* must be one or more characters *x*, *y*, or *d*, signifying that the realworld values of either the x-coordinate, y-coordinate, pixel data value, or some combination of them, is to be written into the current graphic plane, using the current value of the origin offset to locate the text (see the **o** subcommand above).

U — **Undo most recent image update** – the effect of the most recent **e** or **P** subcommand is undone. Unlike the **u** subcommand, a second execution of **U** will *not* cancel the effect of the first.

< **[**</|**]** *name* — **Redirect the input command-stream.** File *name* is opened and its contents interpreted as *iged* subcommands, one per line. Unless a second "<" is included, the subcommands are echoed as they are executed. Alternatively, if a leading "|" is specified, *name* is interpreted as a command or pipe whose standard output is to be read as a series of *iged* commands. If *name* is omitted, *iged* displays the name of the last file executed.

> **[**>/|**]** *name* — **Redirect output listings.** Subsequent *iged* listings (as distinct from prompts or error messages) will be directed to file *name*. If a second ">" is used, the output will be appended to the file; otherwise any existing file will be overwritten. Alternatively, if a leading "|" is specified, *name* is interpreted as a command or pipe whose standard input is to receive subsequent *iged* listing output. If *name* is omitted, listings will be sent to the standard output stream; normally, this will be the terminal.

! *text* — **Execute a shell command.** *text* is passed to *csh* and executed immediately. It is possible to execute other commands that reference the display, including a fresh copy of *iged* itself. The choice of shell program may be changed by issuing the command `**!***shell*´, where *shell* is the absolute pathname of the shell command, e.g. `/*bin*/*sh*´. This shell will be used instead of the default, `/*bin*/*csh*´ for all subsequent filename expansions performed by *iged*.

’’ *char [value]* — **Define or invoke an** *iged* **macro** – the macro is named *char* (a lower-case letter a–z). and represents a sequence of *iged* subcommands that will be executed together whenever the macro is invoked. *value* specifies the string of subcommands, in which each subcommand character must be preceded by a minus sign. If *value* is omitted, the macro *char* is executed. If *char* is omitted, all currently defined macros are listed.

[*% %* — **Draw an open rectangle** – a box is drawn with the specified locations at its corners. The sides of the box will be oriented parallel to the x- and y-axes.

] *% %* — **Draw a closed rectangle** – a filled box is drawn with the specified locations at its corners. The sides of the box will be oriented parallel to the x- and y-axes.

? *[command]* — **Return information about an** *iged* **subcommand.** If *command* is omitted, a list of all *iged* subcommands and features is displayed; otherwise, detailed information is displayed about that particular subcommand or feature. *Iged* actually reads its UNIX manual file and pipes the relevant sections through *nroff*(1).

. *[text]* — **Pass a command to the image display.** Assuming that the display device can accept ASCII command strings, *text* following the period is converted to upper case and is executed as if it were entered directly at the display keyboard. If *text* is omitted, the display device is "reset".

= *[text]* — **Re-execute an** *iged* **subcommand** - where *text* begins with a subcommand character. Any additional text will be appended to the most recent instance of that subcommand. When *text* is omitted, the " = " subcommand lists the most recent instances of each *iged* subcommand. " == " executes the most recent non-macro subcommand.

AXIS SPECIFICATION

The axis titles and the fonts in which they are written are specified by GIPS header variables **xtitle**, **ytitle**, **xfont**, and **yfont**, but they are located with respect to the axes themselves by information in the **xaxis** and **yaxis** string items, each of which contains a number of sub-items of the form "**item=value**". A sample pipeline-mode x-axis drawing command might be specified as

iged –x mark=10 tic=1 font=I.6 by=15.0 place=tbci

In interactive mode, the leading dash may be omitted and the entire sequence must end with a carriage return. The permitted sub-fields are as follows:

mark specifies the maximum number of scale labels that these commands attempt to insert along the axis line.

tic specifies the number of tic marks to be inserted between each scale interval.

font specifies the Versatec/Varian font to be used to write the scale numbers.

place specifies one or more characters in the string "**lcriotmb**" according to whether the titles and scale notation should be **l**eft, **c**enter, or **r**ight justified, inserted **i**nside or **o**utside the image area, and placed at the **t**op, **m**iddle, and/or **b**ottom of the image.

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

from,to,by
specify the steps to be used in drawing the axis scale notations. If specified in floating-point form, i.e. with an explicit decimal point or exponent, realworld values are assumed. Otherwise, they are assumed to represent pixel values. The same notation is used for specifying the **loc** option.

line specifies an 8-bit mask used to construct grid lines connecting each scale mark. Refer to the **k** subcommand for details. If *line=0* (the default), no connecting lines will be drawn.

form specifies a non-default format in which to draw the axis scale numbers. The format is specified as for the UNIX *printf*(3S) function, and should contain no embedded whitespace. The default is "*%.6g*" unless *scale=d* or *scale=h*, in which case the defaults are "*%d\(de%d\'%.12g\'\'*" and "*%dh%dm%.12gs*" respectively. Therefore, there must be at least three percent fields in these cases, and one in all others.

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

scale specifies a code letter which causes axis scale numbers to be formatted as: **s** (simple fixed or float numbers), **l** (ten to the power of ...), **d** (degrees, minutes, and seconds), or **h** (hours, minutes, and seconds).

COMMAND EDITING

In interactive mode, *iged* maintains a list of the most recently issued instance of each subcommand. That command can then be reissued by prefixing an equals sign to the command character. Any text following the subcommand character will be appended to the command. E.g. if the last instance of the "**t**" command was "**t <20 >30 Tag**", then typing "**=t 2**" results in the issuing of "**t <20 >30 Tag 2**".

Similarly, the "ˆ" character invokes string replacement in the most recent instance of a subcommand. Continuing the example begun in the previous paragraph, typing "**tˆ30ˆ40ˆ**" would result in the subcommand: "**t <20 >40 Tag 2**". The third carat may be omitted if the replacement string contains no blank characters, and the second carat can be omitted if it is your intention to remove the search string entirely. The change is applied once only to the subcommand string, unless a "*g*" character follows the third carat, when the change is applied *globally* to all occurrences of the search string.

FILES

/usr/gips/tables/cprofile GIPS header definition

/usr/gips/lib/man.iged	nroff macros for help messages
/usr/lib/vfont/*	graphics font definition files

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihs(1), comsubs(3), comlabel(3)

BUGS

So far, *iged* has only been implemented on the Comtal Vision One/20 display system. See *comsubs*(3) for details on the display interface.

NAME

iha – perform pixel arithmetic on one or more General Images

SYNOPSIS

iha [**–BSp**] [**–d** *dir*] [**–amo** *file*] [**–l** *libs*] [*expr*...] [*name*...]

DESCRIPTION

The *iha* command reads one or more input General Images and generates an output General Image that represents the result of a given arithmetical operation applied pixel-by-pixel. All input and output images must contain the same number of *x* and *y* pixels, (defined by header variables *xnum* and *ynum*), although the pixels may be represented by different *dtype* formats or *dscale* mappings.

The operation, which we shall refer to as a **script**, must be written in C source code, and must either be stored in the file specified by the **–a** option or must appear as "*expr*" in the *iha* command line. Unless any of the input or output images contain complex pixels (*dtype=c*), the form of the *expr* should be

$$\mathbf{p} = \mathit{funct}(\mathbf{p_1,p_2,...p}_n\mathbf{,x,y})$$

specifying that the output pixel *p* should be created by an operation performed on the corresponding pixels of input images 1 through *n*. *x* and *y* may be used to reference, respectively, the column and row index of the image, counting the first pixel (i.e. the top left of a displayed image) as *x=0, y=0.* For example, the following script

iha "p = (x>=20 && y>=20) ? p1+p2 : 0" gi.one gi.two > gi.three

will create "*gi.three*" from "*gi.one*" and "*gi.two*" by summing corresponding pixels, but setting the first 20 rows and columns to zero. The script syntax is C language. In fact, *iha* uses the C compiler to create a temporary program, so the script can refer to other subroutines provided these are made known to the UNIX loader (see the **–l** option). Note that the *expr* script is enclosed within quotation marks to "hide" any meta-characters from the shell, and to instruct *iha* to treat it as a single argument.

The following header variables quantities may also be used in the script, provided they are defined in the header of the input image, or have been added via the **–m** option:

dmax, dmin, dnull
xmax, xmin, xlfft, xnum
ymax, ymin, ylfft, ynum

The following example shows how you might use *iha* to apply a user-written function *fun1()*, a C function returning an *int* value, that you have compiled into the object file *fun1.o*, to the image file *gi.one*:

iha "extern fun1(); p=fun1(p1)" –l fun1.o gi.one > gi.two

The script syntax for complex pixels is similar, but they are referred to as *r* and *i* instead of *p*, e.g.

iha "r=r1+i2 ; i=r2–i1" gi.one gi.two > gi.three

Note the semi-colon separating the two assignment operators, just as there would be between two C statements.

–B instructs *iha* to use "brute force" conversion even when the program determines that it would be quicker to generate a conversion table, e,g, to transform two byte-pixel images into a third, a 64Kbyte conversion table would otherwise be constructed.

–S writes processing statistics to *stderr*.

–p writes the temporary C source file to the standard output stream *stdout*. Only image headers are processed. Use this flag for debugging purposes only.

–a *file* specifies the name of a file that contains the C-language expression to be applied to the input images. If **–a** is omitted, *iha* assumes that the expression is its first non-flag-related argument.

–d *dir* overrides "*/tmp*" as the directory in which to allocate temporary files.

–l *libs* specifies one or more additional object files or libraries to be loaded along with the temporary C program. *iha* automatically includes the GIPS routine library and the C math library. You must supply any others that are used by your *expr* script.

–m *file* specifies a file whose header is to be merged with that of the first input image to form the header of the output image.

–o *file* directs the output General Image to the named file, the default being the standard output stream *stdout*.

Remaining arguments in the *iha* command line are the script itself (only if no **–a** file was specified) followed by the input file name(s). If file names are omitted entirely, the standard input stream *stdin* is read. To read *stdin* along with other files, use "**–**" as a file name.

FILES

/tmp/iha∗.c	temporary script source file
/tmp/iha∗	temporary compiled script command
/usr/gips/tables/cprofile	image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihm(1)

DIAGNOSTICS

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

- bad dscale=p for *file*
- can't compile
- can't execute
- cannot create pipe
- compiler error: *text*
- different xnum: *file*
- different ynum: *file*
- dscale=f unsupported: *file*
- expr missing*file*
- *file*: bad dtype
- not an assignment: *file* or *expr*
- open error on pipe
- write error to pipe

BUGS

The output image cannot be defined as *dscale=p*, and no image may be defined as *dscale=f*.

The syntax is not close enough to *awk*.

Bizarre error messages occur when running *iha* on Sun–3 systems if the value of the FLOAT_OPTION environment variable differs from that specified when the GIPS library was compiled.

NAME

ihbidr – access a Magellan BIDR as a GIPS cellular image

SYNOPSIS

ihbidr [**–D**] *file*

DESCRIPTION

The *ihbidr* constructs a GIPS header to describe Magellan BIDR image *file* so that the *gcell*(3) routines can access it as a GIPS cellular image. *file* must be located on a disk—it cannot be specified as a pipe. The header may be read by any GIPS command that accepts cellular images, e.g. *gipstool*(1), *ihgrid*(1). or *ihgeom*(1).

If a file named *file*.aux is found in the same directory as *file*, it is assumed to contain an index file created by the *bidrindx* program, and is used to initialize tables that assist the *gcell* routines in locating particular BIDR data blocks. Otherwise, *gcell* must read through the BIDR sequentially until the requested block is located.

–D instructs the *gcell* routines to write the following information information to the standard error stream, *stderr*, whenever a new BIDR block is accessed:

- latitude of first pixel in block
- longitude of first pixel in block
- x-offset of first pixel relative to image origin
- y-offset of first pixel relative to image origin
- file offset of first pixel in block

FILES

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

SEE ALSO

gcell(3)

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

- cannot read *file*
- missing file name
- multiple file names: *args*
- unknown option: *text*

NAME

ihbox – apply a boxcar convolution filter to a General Image

SYNOPSIS

ihbox [**–nu**] [**–mo** *file*] [**–0** *dzero*] [**–1** *dfact*] *array* [*name*]

DESCRIPTION

The *ihbox* command reads a General Image from file *name*, and applies a spatial filter defined by the contents of the file specified by the (required) *array* argument. If *name* is omitted, *ihbox* reads the standard input stream *stdin*.

Array specifies a file containing an $x \times y$ array of fixed or floating point numbers in ASCII coded format, beginning with a pair of integers specifying the column and row dimensions of the array (both of which must be odd and greater than 1) and continuing with the numeric array elements themselves in row-major order (all of row 1, followed by all of row 2, etc). Repeated elements of the same value may be denoted by an integer repeat count, followed by an asterisk, followed by the value to be assigned that many times, e.g. "*32*20.5*".

Ihbox first looks for *array* in the current directory, then in "/*usr*/*gips*/*lib*/*boxcar*".

Alternatively, if *array* contains any whitespace characters (blanks, tabs, or newlines), *ihbox* interprets it as a string containing the array dimensions and element values themselves. This string format is particularly useful when *ihbox* is invoked within a script.

If the input General Image contains "integer" pixels (i.e. *dtype* = *b, u, s, h, i,* or *l*), the array elements will, if necessary, be converted to signed integers. *ihbox* then transforms the image by replacing each pixel with the convolution of the user-defined array with the pixel (and its neighbors), using signed integer or floating-point arithmetic (depending on *dtype*). If header variable *dnull* is defined, pixels with that value will not be included in the sum, although they may be replaced by the filtering. If *xscale=w*, the convolution wraps around the left- and right-hand map extremities. This feature may be used when filtering images in which the x-axis represents an azimuthal variable, e.g. longitude.

When 8-bit or unsigned integer pixels are filtered, it is quite likely that the resulting values cannot be represented correctly (e.g. a low-pass filter applied to a "*dtype=u*" image may generate negative values, which cannot be represented by that pixel format). To avoid this problem, if the input image is in "*dscale=n*" format, new values of *dzero* and *dfact* may be specified for the output image by means of the **–0** and **–1** options, respectively. The pixels will then be transformed to floating point values before filtering, using the original *dzero* and *dfact*, and back to *dtype* before they are written out, using the new *dzero* and *dfact*.

–o *file* directs the output General Image stream to *file*. If omitted, the output will be written to the standard output stream *stdout*.

–m *file* specifies a second General Image file whose header is to be merged with that of the input image before filtering.

–n directs *ihbox* to filter *only* those pixels with value equal to the header variable *dnull*. Otherwise, all pixels are filtered.

–u no normalization is performed on the convolution-sum output. Otherwise, each output pixel is normalized by the sum of the absolute values of the contributing *array* elements.

–0 *val*
–1 *val* specify new values of *dzero* and *dfact* to apply to output pixels of *dscale=n* images (see above).

FILES

/usr/gips/tables/cprofile image header definition table
/usr/gips/lib/boxcar/ directory containing specimen arrays

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihc(1), ihread(3)

DIAGNOSTICS

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

- multiple input files: *file*
- missing array file
- bad array limits: *file*

BUGS

Output from *ihfft*(1) (signified by header variable *dtype=c*) and all images in polynomial format (*dscale=p*) cannot be handled by *ihbox*.

NAME

ihc – create a General Image

SYNOPSIS

ihc [**–** *size]* [**–bfirw**] [**–d** *dir*] [**–hmo** *file*] [**–s** *bytes*] [*name*]

DESCRIPTION

The *ihc* command combines an ASCII header and a binary image array or pixel stream to produce a General Image that can be processed by subsequent GIPS routines. If *name* is specified, the binary array or pixel stream is read from a file of that name; otherwise, it is read from the standard input stream *stdin*.

–*size* specifies the (maximum) physical block size of the binary input file. This should always be specified when input comes directly from magnetic tape. The default is 1024 bytes (which would be specified as "*–1024*").

–h *file* specifies the name of the input header file that will describe the output General Image. If a relative pathname is used, *ihc* will search for it in the current directory, then in "**/usr/gips/tables/**". The format of image headers is described in *ihread*(3).

–m *file* specifies a secondary header file whose contents are to be merged with the header specified by the **–h** option. The same search rules apply.

–o *file* specifies the destination of the output General Image. If omitted, it will be written to the standard output stream *stdout*.

–i inverts the input array, i.e. from top to bottom.

–r reverses the input array, i.e. from left to right.

–b byte-swaps the input array (interchanging consecutive pairs of bytes).

–s *n* skips (ignores) *n* bytes at the beginning of the input stream.

–w specifies that the input stream contains pixel values in random order. The input consists of sets of 3 binary fields: a 4-byte integer representing the *x*-coordinate, a 4-byte integer specifying the *y*-coordinate, and a 4-byte floating-point field specifying the pixel value itself.

–f has the same effect as **–w**, but the pixels that are read from *stdin* are in ASCII format: pairs of (optionally signed) integers, each followed by a floating-point pixel value.

–d *dir* overrides the default directory "/*tmp*" in which any necessary scratch files will be written. For example, it is necessary to create a temporary file if the input is from *stdin* and the array is to be inverted, or if from tape and the specified block size is not a multiple of the raster length, or if the **–w** or **–f** flags are used.

FILES

/tmp/ihc∗	default spool file
/usr/gips/tables/cprofile	image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

gin(1), ihs(1), ihm(1), ihread(3)

DIAGNOSTICS

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

- multiple input files
- odd raster length - can’t byte swap
- unexpected EOF: *file*

NAME

ihfft – Fourier transform each raster of a General Image

SYNOPSIS

ihfft [**–i**] [**–mo** *file*] [*name*]

DESCRIPTION

The *ihfft* command reads a General Image from file *name* (or, if omitted, from the standard input stream *stdin*), and applies a one-dimensional "fast" Fourier transform to each raster – i.e. each line of pixels parallel to the x-axis.

The output image, written to the file specified by the **–o** *file* option (default output to the standard output stream *stdout*), contains pairs of 4-byte floating-point pixels, representing respectively, the cosine and sine components of the transform. If the number of pixels in the raster is not an exact power of 2, the raster is extended: if "*xscale=w*", the raster is "wrapped" from end to end before the transform is taken; otherwise it is extended with zeroes. The number of original input pixels per raster is saved in header variable *xlfft*.

Use *ihrot*(1) to apply *ihfft* to columns of the transformed image, or *ihm*(1) or *iha*(1) to "edit" images in Fourier space. When a complex image is passed through *ihs*(1), each sine-cosine pixel pair is replaced by its modulus – sqrt(sin**2 + cos**2).

–m *file* specifies a second General Image file whose header is to be merged with that of the input image before performing the transform.

–o *file* overrides the destination of the output General Image, which, by default, goes the the standard output stream *stdout*.

–i directs *ihfft* to take the inverse transform. The number of (complex) output pixels will be determined by the value of header variable *xlfft*. When performing a double Fourier transform, use "*ihrot*" to rotate the image for the second forward transform, and "*ihrot –ri*" to rotate it back again before the second reverse transform.

FILES

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

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihrot(1), ihs(1), ihread(3)

DIAGNOSTICS

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

- bad arg: *name*

BUGS

Images in polynomial format (*dscale=p*) cannot be transformed.

The Fourier transform code is optimized for the VAX, not for the MC68881.

NAME

ihfromfits – convert a FITS image into GIPS format

SYNOPSIS

ihfromfits [**–bh**] [*name*]

DESCRIPTION

Ihfromfits is a filter that converts an input image file in the Flexible Image Transport System (FITS) format into the GIPS General Image Processing System format, suitable for subsequent pipeline processing and display on a suitable image frame buffer.

–b causes the binary input image to be "*byte-swapped*", i.e. adjacent pairs of bytes are interchanged in the binary image array.

–h causes all **HISTORY** entries in the input header to be retained in the output GIPS header. If omitted, only a single history entry will be generated, composed of a concatenation of the **OBJECT, DATE, DATE-OBS, ORIGIN, INSTRUME,** and **OBSERVER** fields of the input FITS header.

The *ihfromfits* command reads from the named file, or, if omitted, from the standard input *stdin*, and writes its output to the standard output *stdout*. It generates an output header by using the contents of file **FITSinit** as a skeleton. If this file is not found in the current directory, it is read from **/usr/gips/tables/**. This header is then updated and augmented with information from the header of the FITS file. The following FITS header variables are recognized, with the restrictions as noted. Variables not contained in this list are ignored.

END	must be present to denote the end of the FITS header.
SIMPLE	must be present and have the value **T**.
BITPIX	must be present and have the value 8, 16, or 32, corresponding to unsigned bytes, signed 16-byte integers and signed 32-byte integers, respectively. There is no provision for handling either unsigned integers or floating-point pixel fields.
NAXIS	must be present and have the value 2.
NAXIS*n*	Both **NAXIS1** and **NAXIS2** must be present. The output image header will specify pixel limits as *xnum*=**NAXIS1**, *ynum*=**NAXIS2**.
CRVAL*n*	Both **CRVAL1** and **CRVAL2** must be present. They are used, along with **CRPIX***n* and **CDELT***n* to generate header variables *x*/*ymin* and *x*/*ymax*.
CRPIX*n*	Both **CRPIX1** and **CRPIX2** must be present.
CDELT*n*	Both **CDELT1** and **CDELT2** must be present.
CROTA*n*	Both **CROTA1** and **CROTA2** must be present.
CTYPE*n*	The 8-byte FITS mnemonics are, where possible, translated into longer, more descriptive titles, in the manner described below, and are written to header variables *x*/*ytitle*.
DATAMAX	This field is copied to the output variable *dmax*.
DATAMIN	This field is copied to the output variable *dmin*.
BSCALE	If both **BSCALE** and **BZERO** are present, they are copied to header variables *dfact* and *dzero*, and *dscale* is given the value **n**. Otherwise, *dscale* is set to **s**.
BZERO	See **BSCALE** above.
BUNIT	The 8-byte FITS mnemonic is, where possible, translated into a longer, more descriptive title, in the manner described below, and is passed to header variable *dtitle*.
BLANK	This field is passed to header variable *dnull*.
OBJECT	This is copied to the header variable *title*.
DATE	This is copied to the header variable *date*.
DATE-OBS	The character-string value is copied to a *history* entry.
ORIGIN	The character-string value is copied to a *history* entry.
INSTRUME	The character-string value is copied to a *history* entry.
OBSERVER	The character-string value is copied to a *history* entry.
HISTORY	Provided the **–h** flag is set, all HISTORY fields in the FITS header are copied to separate *history* entries in the output header.

In addition, the FITS variables that specify unit-types are translated into more descriptive titles, according to the following scheme:

K	Kelvins
JY/BEAM	Jy per beam area
JY/PIX	Jy per pixel
MAG/PIX	Magnitudes per pixel
M/SEC	Meters per second
DEGREES	Angle in degrees
RA	Right Ascension (deg)
DEC	Declination (deg)
LL	Tangent plane E-W (deg)
MM	Tangent plane N-S (deg)
GLON	Galactic longitude (deg)
GLAT	Galactic latitude (deg)
ELON	Ecliptic longitude (deg)
ELAT	Ecliptic latitude (deg)
TIME	Time (sec)
FREQ	Frequency (Hz)
LAMBDA	Wavelength (meters)
VELO	Velocity (m/sec)
VELO-LSR	Velocity wrt local rest
VELO-HEL	Velocity wrt Sun
VELO-OBS	Velocity wrt observer
PIXEL	Pixel
STOKES	Stokes polarization
COMPLEX	Complex

FILES

/usr/gips/tables/FITSinit	skeleton image header
/usr/gips/tables/cprofile	header definition table

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihtofits(1), ihs(1), ihread(3)

FITS is a product of the data processing sections of Kitt Peak National Observatory and the National Radio Astronomy Observatory, and is described in a paper presented at the "International Workshop on Image Processing in Astronomy", held in June 1979 at the International Center for Theoretical Physics in Miramare, Trieste, Italy. More information about the format can be obtained from

Dr. Eric W. Greisen
National Radio Astronomy Observatory
Edgemont Road
Charlottesville, VA 22901

DIAGNOSTICS

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

- multiple input files: *file*
- EOF while reading FITS header: *file*
- variables missing from FITS header: *item*
- error while reformatting *file*
- This is not a SIMPLE image

- NAXES not .GE. 2: *item*
- Unsupported BITPIX: *item*
- EOF before end of image: *file*

NAME

ihfromgif – translate a GIF image file to GIPS format

SYNOPSIS

ihfromgif [**–Sv**] [**–amo** *file*] [*name*]

DESCRIPTION

The *ihfromgif* command reads an image in Compuserve's GIF format and writes it out to the standard output, or to the file specified by the **–o** flag, in GIPS image format. This operation is essentially the reverse of that performed by the *ihtogif*(1) command.

–S writes processing statistics to *stderr*.

–a *file* specifies where to write any colormap information present in the input image. See the manual entry for *gipstool*(1) for the format of a *colormap* file.

–o *file* overrides the default destination for the output GIPS image (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the output image.

–v invokes verbose mode, writing statistics about the input image to the standard error stream, *stdout*.

FILES

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

SEE ALSO

ihtogif(1)

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

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

- GIF buffer overflow
- bad GIF header
- corrupt GIF file - bad end-of-header
- insufficient memory for output buffer
- missing GIF image separator
- multiple input files
- unexpected EOF in file header
- unexpected EOF in image header
- unexpected end of image

NAME

ihfrompix – translate a Sun rasterfile to GIPS image format

SYNOPSIS

ihfrompix [–S**]** [**–amo** *file*] [*name*]

DESCRIPTION

The *ihfrompix* command reads a Sun *rasterfile* image and writes it out to the standard output, or to the file specified by the **–o** flag, in GIPS image format. This operation is essentially the reverse of that performed by the *ihtopix*(1) command.

–S writes processing statistics to *stderr*.

–a *file* specifies where to write any colormap information present in the input image. See the manual entry for *gipstool*(1) for the format of a *colormap* file.

–o *file* overrides the default destination for the output GIPS image (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the output image.

FILES

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

SEE ALSO

gipstool(1), ihfrompix(1), rasfilter8to1(1), rasterfile(5), rastrepl(1)

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

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

- multiple input files

NAME

ihgeom – transform random-access General Image via a control grid

SYNOPSIS

ihgeom [*–buffers*] [**–DSfnx**] [**–mo** *file*] *file* [*grid*]

DESCRIPTION

The input control grid file *grid* (or, if omitted, the standard input stream *stdin*), is used to apply a transformation to a "random-access" General Image file, and the resulting General Image is written to the file specified by the **–o** option (or, if omitted, to the standard output stream *stdout*).

The *file* argument is required. It is the name of the input random-access General Image, which must be a disk file that has been created from a normal General Image by the *ihgfmt*(1) command.

–buffer specifies the number of image cells to be allocated. Access to the input image is optimized by storing several such cells in an in-core ring buffer; by default, 8 are stored, but this number may be altered, viz: "*ihgeom –32*" will use 32 buffers (and, therefore, more data memory).

–f specifies that the input *grid* file is written in formatted ASCII code, rather than in binary integers and floats.

–n suppresses the output image. It may be used in conjunction with the **–D** flag to verify or display an input control grid stream.

–x each output pixel is generated by averaging the four nearest-neighbors of the required location in the input image; otherwise, only the nearest pixel value is used.

–m *file* specifies the name of a file containing a General Image header whose contents are to be merged with the input random-access General Image before the transformation is attempted.

–D instructs *ihgeom* to write a one-line description of the input and output coordinates of each control grid cell to the standard error stream *stderr*.

–S causes *ihgeom* to write UNIX usage statistics to *stderr* when it has completed its task.

CONTROL GRID FORMAT

The control grid file specifies the transformation by breaking up the output image into *blocks* of rasters, and further sub-dividing each block into a series of rectangular *cells*. In general, the shape of a region of input image that corresponds to a rectangular output cell will not itself be rectangular. *ihgeom* assumes that the cells have been chosen small enough that, to find the input pixel coordinate that corresponds to an output pixel within a given cell, it is sufficient to perform a bi-linear interpolation using the coordinates of the *corners* of the input and output cells.

The control grid may be supplied either in binary or ASCII format, the latter being indicated by the **–f** flag. In binary format, each input field is either a *long* integer (i.e. INTEGER∗4), or a *float* (REAL∗4). *Ihgrid*(1) always generates a binary control grid.

The control grid begins with two lines of ASCII characters, each ending in a linefeed character. The first line specifies the number of columns and rows (rasters) to be assigned to the output General Image, and the number of rasters to be allocated to the *ihgeom* output buffer. The number of columns and rows are also assigned to variables *xnum* and *ynum* respectively, in the output image header. This line may be decoded by *sscanf()* using the format string GIPS_GRID defined in "*/usr/gips/include/gips.h*". The second line defines the parameters of the transformation; its *sscanf()* format is defined by GIPS_MAPPING in the same file. The second line will become the value of the *mapping* variable in the header of the output image created by *ihgeom*.

The remainder of the control grid file contains sets of 12 numbers: 4 integers followed by 8 floats. The integers define the top-left (*x,y*) and bottom-right (*x,y*) pixel coordinates of a rectangular cell in the output image. The 8 floats specify the (*x,y*) pixel coordinates of the corners of the corresponding cell in the input image, in the order: *top-left*, *top-right*, *bottom-left*, and *bottom-right*.

The output image will be generated as a set of blocks, each containing a fixed number of *y*-coordinate rasters (the number specified by the third header field), except the last block, which may, of course, be shorter. The control grid cells must refer to the blocks according to increasing *y* value, but, within each block, they may fill at random. A cell may not overlap output blocks.

Here is a simple example of a single output block and a single control grid cell. If the **–f** flag is specified, this would represent a perfectly legal *ihgrid* input stream of ASCII digits and (ignored) blanks, tabs, and newline characters. Decimal points on the floats are optional if the field has no fractional part:

```
Gips control grid: xnum=512 ynum=512 rasters=512
name=Tiepoint type=0 origin=0,0
        0 0
                512 512
                        300.    100.            800.    250.
                        200.    390.            900.    450.
```

This control grid will generate a 512×512 image from a segment of an input image whose (*x*,*y*) coordinates are (300,100), (800,250), (200,390), and (900,450). In this case, the control grid defines a single rectangular cell. Note that the bottom-right output coordinates (512 and 512 in this example) are one more than the pixel address of the corresponding bottom-right output pixel, viz. (511,511). A more realistic example might be the following:

```
Gips control grid: xnum=2000 ynum=1500 rasters=750
name=Tiepoint type=0 origin=0,0
        0 0
                1000 750
                        6.      0.              1081.   21.
                        42.     673.            983.    814.
        1000 0
                2000    750
                        1081.   21.             1932.   17.
                        983.    814.            1811.   791.
        0 750
                1000 1500
                        42.     673.            983.    814.
                        80.     1405.           1527.   695.
        1000 750
                2000 1500
                        983.    814.            1811.   791.
                        1527.   695.            1928.   1527.
```

This control grid is composed of four cells and two output blocks: the first pair of cells refers to block 1. Cell 1 specifies that the output rectangle with coordinates (0,0), (1000,0), (0,750), (1000,750) is to be linearly interpolated from input image coordinates (6,0), (1081,21), (42,673), (983,814), respectively. The output pixels will fill the box from *x*=0 through *x*=999 and *y*=0 through *y*=749. The second cell maps the output rectangle (1000,0), (2000,0), (1000,750), (2000,750) from input coordinates (1081,21), (1932,17), (983,814), (1811,791). These first two cells define the mapping into the first 750-raster-long output buffer. The remaining pair of cells defines the mapping into the second buffer.

FILES

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

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihgfmt(1), ihgrid(1), ihread(3)

DIAGNOSTICS

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

- bad arg: *arg*
- bad geometry: *list*
- bad *x*/*y*-coordinate in cell: *n*
- cell spans output buffers: *n*
- input not generated by ihgfmt: *file*
- no input image file specified

BUGS

The Control Grid file format is not "compact" – it is, in fact, about twice as long as it needs to be for most transformations.

NAME

ihgfmt – create a random-access General Image

SYNOPSIS

ihgfmt [–*size*] [**–NS**] [**–d** *dir*] [**–mo** *file*] [*name*]

DESCRIPTION

The input General Image file *name* (or, if omitted, the standard input stream *stdin*), is reformatted and written to the file named by the **–o** option (or, if omitted, to the standard output stream *stdout*). The output image is reformatted into cells whose size in bytes will be a power of 2.

The reformatted General Image files, which contain special headers, are suitable for subsequent processing by the *ihgeom* command. They are distinguished from "normal" General Images by the special value "*G*" assigned to header variable *dscale*, which prevents these images from being processed by most other GIPS commands.

–*size* specifies the cell size of the output image. *size* must be a power of 2, i.e. "*ihgfmt –4096*" will write 4096–byte cells. The default cell size is 1024 bytes, which is chosen to be the same as the physical blocksize of many UNIX disk files.

–m *file* specifies the name of a file containing a General Image header whose contents are to be merged with the input General Image before reformatting is attempted.

–N begins writing the output image cells immediately after the output header. Normally, *ihgfmt* adds null bytes between the header and the first output cell so that each cell begins at a byte-offset from the start of the file that is a multiple of the cell size itself. Note that most GIPS commands cannot read an image that has been generated with the **–N** flag of *ihgfmt*.

–S causes *ihgfmt* to write UNIX usage statistics to *stderr* when it has completed its task.

–d *dir* overrides the name of the directory (by default, "*/tmp*") into which *ihgfmt* will write a temporary scratch file when the input General Image is read from the standard input stream *stdin*.

FILES

/tmp/ihg∗ default scratch file
/usr/gips/tables/cprofile image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihgeom(1), ihgrid(1), gcell(3), ihread(3)

DIAGNOSTICS

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

- bad arg: *arg*
- bad –*size* specification

NAME

ihgrid – generate control grid for General Image transformation

SYNOPSIS

ihgrid [*–nproj*] [**–DSf**] [**–C** *long,lat*] [**–l** *long,lat,rho*] [**–Pp** *ang*] [**–XY** *org*] [**–r** *rad*] [**–R** *fact*] [**–bnxy** *num*] [**–mo** *file*] [*name*]

DESCRIPTION

A control grid is generated for transforming a random-access General Image into a given cartographic coordinate system. Consult *ihgeom*(1) for a discussion of control grids.

–*nproj* specifies the desired transformation:

0	**Tie-Point**	Polynomial	Bi-variant
1	**Gnomonic**	Azimuthal	r=tan(d)
2	**Stereographic**	Azimuthal	r=2tan(d/2)
3	**Orthographic**	Azimuthal	r=sin(d)
4	**Equidistant**	Azimuthal	r=d
5	**Equivalent**	Azimuthal	r=2sin(d/2)
6	**Lambert**	Conformal	One parallel
7	**Lambert**	Conformal	Two parallels
8	**Mercator**	Conformal	Equatorial
9	**Polar**	Conformal	Stereographic
10	**Mollweide**	Equivalent	Non-conic
11	**Albers**	Equivalent	One parallel
12	**Albers**	Equivalent	Two parallels
13	**Lambert**	Equivalent	Cylindrical
14	**Lambert**	Equivalent	Polar azimuthal
15	**Bonne**	Equivalent	Pseudo-conical
16	**Sinusoidal**	Equivalent	
17	**Werner**	Equivalent	
18	User defined projection via "**user()**"		
19	**Aitoff**	Equivalent	Non-conical
20	**Hammer**	Equivalent	Non-conical
21	**Briesemeister**	Equivalent	Non-conical
22	**Cylindrical**	Equivalent	Non-conical

When a cartographic transformation is specified (types 1 through 21), *ihgrid* expects that the realworld *x*- and *y*- coordinates of the input image are defined in *degrees* of, respectively, longitude and latitude. Also, if the input longitude extends through the full 360 degrees, the input image header should specify that "*dtype=w*", i.e. that the coordinate system *wraps*.

–D generates "debug" output on *stderr,* detailing set-up parameters.

–P *ang* specifies the second standard parallel (in degrees) for Lambert Conformal (–7) and Albers Equivalent (–12) transformations.

–R *fact* multiplies the default transform scaling factor by the floating-point quantity *fact*, thereby "fine tuning" the size of the output image. Alternatively, the user may supply a scaling factor via the **–r** option (qv.) This option is ignored for tiepoint transformations (–0).

–S writes a line of UNIX "statistics" to *stderr* upon command completion.

–X *org*
–Y *org* specify, respectively, the *x* and *y* origins of the transformed image in the output pixel space. This is, typically, the point into which the input origin defined by the **–l** option is mapped, although this is not true for all transformations. The origin may also be specified by the **–C** option. These options are ignored for tiepoint transformations (–0).

–b *num* specifies the number of buffer rasters that *ihgeom* will require to transform an image via the control grid that is to be created. This information must be supplied to *ihgrid* and is passed to *ihgeom* in the output stream header. The default value is 32. The larger the value, the more efficient the transformation and the smaller the control grid file, but *ihgeom* will use more memory. The size of the actual output buffer allocated by *ihgeom* will be this number times the length of each raster.

–f generates formatted ASCII output. Use the **–f** flag of *ihgeom*(1) to read it.

–C *long,lat* specifies the longitude and latitude origin of the transformed image in the output pixel space. This is, typically, the point into which the input origin defined by the **–l** option is mapped, although this is not true for all transformations. The origin may also be specified by the **–X** and **–Y** options. This option is ignored for tiepoint transformations (–0).

–l *long,lat,rho*
defines the Euler angles (in degrees) of a rotation to be applied to the (longitude,latitude) coordinate system before the Cartographic transformation is applied. The origin of longitude will become *long*, the origin of latitude will become *lat*, and the resulting figure will be rotated clockwise by *rho* . If unspecified, these angles will be set to zero. This option will be ignored for tiepoint transformations.

–m *file* specifies the name of another General Image file whose header is to be merged with that read from *name* or *stdin*.

–n *order* is used only with tie-point transformations, and specifies the order of bi-linear polynomials to be used.

–o *file* the control grid will be written to *file*, rather than to the standard output stream *stdout*.

–p *ang* specifies the central latitude (or first standard parallel) of the transformed image (in degrees). If this option is omitted, *ihgrid* uses the arithmetic mean of the realworld *y*-axis limits of the input General Image. It is ignored for tiepoint transformations (–0).

–r *rad* specifies the transform scaling factor – whose meaning depends on the particular non-tiepoint transformation selected. If not specified, *ihgrid* will compute a value that attempts to maximize the area of input image mapped into the output image. Some clipping and/or bordering is almost guaranteed! Use the **–R** option for fine tuning.

–x *num*
–y *num* specify, respectively, the x- and y-pixel extents of the output image to be constructed via the control grid.

name specifies the name of the General Image file to be transformed. If omitted, *ihgrid* expects to read the image (in fact, its header only) from the standard input stream "*stdin*".

TIE POINTS

When *ihgrid* performs a transformation via tie-points (–0 flag), the points are read from the standard input stream *stdin* in groups of 4 ASCII floating-point numbers. If *name* is omitted, *ihgrid* will also expect to read a General Image header from *stdin* before reading the tie points. Non-ASCII bytes in the input stream will cause *ihgrid* to quit. The numbers, separated by any number of spaces, tabs, or newlines, must be in the following order:

- Pixel x-coordinate of point in input image,
- Pixel y-coordinate of point in input image,
- Pixel x-coordinate of corresponding point in output image, and
- Pixel y-coordinate of corresponding point in output image.

Sufficient tie points must be specified so that the polynomial mapping developed by *ihgrid* is not singular. This is generally assured provided the points are not co-linear, and there must be at least (num+1)×(num+1) of them, where *num* represents the order of bi-linear mapping polynomial specified via the **–n** option.

USER-DEFINED PROJECTION

The user may generate a "customized" version of *ihgrid* by writing a C routine, named *user()* in "user.c", and compiling it via the command:

```
cc /usr/gips/lib/ihgrid.o user.c –o ./ihgrid –lGI –lm
```

Here is an example of a *user()* routine:

```
user(sw,x,y,xy)
        double  x, y, xy[2];
{
        switch (sw) {
        case 0:
                /* Initialization call. Precalculate what you want,  */
                /* using x = –p value, y = –P value, and xy[0] = –r value. */
                /* If –r flag not specified, user() will be called twice with sw=0*/
                /*
                if /* initialization ok */
                        return(0);
                else
                        return(1);
        case 1:
                /* Transform realworld xy[0] and xy[1]
                   to pixel coordinates (x,y) */
                xy[0] = ...;
                xy[1] = ...;
                if (/* transformation non-singular */)
                        return(0);
                else
                        return(1);
        case 2:
                /* Transform pixel coordinates (x,y)
                   to realworld xy[0] and xy[1] */
                xy[0] = ...;
                xy[1] = ...;
                if (/* xy[0] and xy[1] are within input image */)
                        return(0);
                else
                        return(1);
        default:
                break;
        }
        /* illegal sw value (or any other abort situation) */
        return(–1);
}
```

FILES

/usr/gips/lib/ihgrid.o	partially linked *ihgrid* for user-defined projection
/usr/gips/tables/cprofile	image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihgeom(1), ihgfmt(1), gtrans(3)

Ford, P., *Guide to GIPS*, MIT Center for Space Research, 1984.

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

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

DIAGNOSTICS

When an error is detected, *ihgrid* halts with system code 1, writing a message to the standard error stream *stderr*. Possible error messages are

- bad –n value: *value*
- bad arg: *value*
- forward mapping for *projection* not implemented
- illegal numeric in input stream: *code*
- inverse mapping for *projection* not implemented
- mapping is singular for this origin choice
- no –P value specified
- setup for *projection* not implemented
- singular tie-point transformation
- unable to compute –r value

BUGS

The Sun–3 version of *user()* must be compiled and linked with the same value of the FLOAT_OPTION environment variable as was in effect when *ihgrid* and the GIPS library were created.

NAME

ihm – merge and/or resample one or more General Images

SYNOPSIS

ihm [**–q**] [**–d** *dir*] [**–hmo** *file*] [**–l** *oper*] [*name*...]

DESCRIPTION

The *ihm* command reads one or more input General Images and generates an output General Image that represents the merging or resampling of the input according to specifications in the output header.

The merge is performed by a weighted average over each input pixel that overlaps a given output pixel. A faster, less accurate merge may be achieved by specifying the **–q** flag, in which case each output pixel is given the value of the first non-null input pixel that overlaps it. Alternatively, the **–l** option lets you combine images according to a set of Boolean operators.

–d *dir* overrides "/*tmp*" as the directory in which to allocate temporary files.

–h *file* specifies an existing General Image file whose header is to form the basis for the output image description. Otherwise, the header of the first input file is used.

–l *oper* indicates that the image pixels are to be treated as unsigned logical quantities and to be combined by Boolean operations. *oper* is a list of operators and subfields, one per input General Image, specifying how each input pixel from that particular image is to be handled. This option overrides **–q** and is invalid for floating-point input or output pixels. Valid subfields in *oper* begin with one of the following symbols

= assign pixel value to output field.
| inclusively **OR** pixel to output field.
& logically **AND** pixel to output field.
ˆ exclusively **OR** pixel to output field.

The symbol may be followed by the "˜" character, indicating that the input pixel is to be complemented before the operation. Four optional numeric fields may follow, separated by commas: the offset and length (in bits) of the input field, and the offset and length (in bits) of the output field. Offsets are counted from the most significant bit. For example, the command

ihm –l "=3,5,0,5 =˜5,3,5,3" gi.1 gi.2

produces a General Image in which each pixel contains the last 5 bits of a pixel from "gi.1" concatenated to the complement of the last 3 bits of each corresponding pixel in "gi.2". In general, *oper* should contain as many sub-fields as there are input General Image files, but if fewer sub-fields are specified, the remaining files are processed according to the last sub-field specified.

Bit-field and arithmetic operations of arbitrary complexity may also be performed by the *iha*(1) command, which is limited, however, to input and output images containing the same number of *x* and *y* pixels.

–m *file* specifies a second file whose header is to be merged with that specified by the **–h** option or with that supplied by the first input file.

–o *file* directs the output General Image to the named file, the default being the standard output stream *stdout*.

The remaining arguments in the *ihm* command line are the input file names themselves. If omitted entirely, the standard input stream *stdin* is read. To merge *stdin* with other files, use "**–**" as a file name.

ihm attempts to process all input files synchronously, and always writes the output in increasing y-raster order. If this requires that an input file be read in a non-sequential manner, and if it is not being read directly from disk (e.g. it is a pipe or a tape file), *ihm* will copy that input stream to disk before proceeding. This will increase execution time and generate a warning message on the standard error stream *stderr*.

FILES

/tmp/ihm*n	default temporary spooled images
/usr/gips/tables/cprofile	image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

iha(1), ihc(1), ihread(3)

DIAGNOSTICS

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

- –q flag illegal for dtype=p input: *file*
- bad –l field offset: *n*
- bad –l value: *oper*
- bad dtype for –l option: *file*
- illegal change of dtype: *file*
- inconsistent wrap limits: *file*
- spool sequence error: *file*
- too many –l fields: *oper*

BUGS

The **–q** option is sometimes disregarded, e.g. when *dscale=f* or when *dscale=n* and the input and output values of *dzero* and *dfact* are not identical.

The format of the **–l** option used to specify Boolean bit-field operations is a poor hack.

NAME

ihmed – apply a median filter to a General Image

SYNOPSIS

ihmed [**–S**] [**–mo** *file*] [**–xy** *dim*] [*name*]

DESCRIPTION

The *ihmed* command reads a General Image from file *name*, and applies a median filter defined by the **–x**, and **–y** options. If *name* is omitted, *ihmed* reads the standard input stream *stdin*. Each pixel is replaced with the median value of the pixels in a rectangular box surrounding it, with dimensions specified by the **–x** and **–y** options. Null pixels, i.e. those with the same value as the *dnull* header variable, are ignored when computing the median.

If the input General Image contains "integer" pixels (i.e. *dtype = b, u, s, h, i,* or *l*), the calculation will be performed in fixed-point arithmetic. Otherwise, floating-point arithmetic will be used. If *xscale=w*, the convolution wraps around the left- and right-hand map extremities. This feature may be used when filtering images in which the x-axis represents an azimuthal variable, e.g. longitude.

–S write processing statistics to *stderr*.

–m *file* specifies a second General Image file whose header is to be merged with that of the input image before filtering.

–o *file* directs the output General Image stream to *file*. If omitted, the output will be written to the standard output stream *stdout*.

–x *num* the number of x-pixels to be averaged. The default is 3.

–y *num* the number of y-pixels to be averaged. The default is 3.

FILES

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

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihbox(1), ihread(3)

DIAGNOSTICS

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

- multiple input files: *file*

NAME

ihnull – copy a General Image, possibly updating header fields

SYNOPSIS

ihnull [**–S** *file*] [**–amo** *file*] [**–n** *factor*] [*name*]

DESCRIPTION

The *ihnull* command reads a General Image from file *name*, updates the header from an file specified by the **–m** option, and writes it out to the standard output, or to the file specified by the **–o** flag, without altering the pixels in any way. If *name* is in cellular format, i.e. it has been created by the *ihgfmt*(1) program, *ihnull* will turn it into a "flat" image that can be piped into other GIPS commands.

–S *file* writes processing statistics to *stderr*.

–a *file* writes an additional copy of the output file to *file*.

–n *fact* shrinks the output image by a positive integer factor, *fact*. If the input image header contains a *mapping* variable, *ihnull* will also reduce the *origin* and *radius* fields by an amount equal to *fact*, thereby preserving the mapping information for most projections.

–o *file* overrides the default output destination (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the input image.

FILES

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

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

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

- multiple input files
- bad –n value
- unexpected EOF
- bad cell format

NAME

ihpds – translate a Planetary Data System image to GIPS format

SYNOPSIS

ihpds [**–hlvS**] [**–mo** *file*] [*name*]

DESCRIPTION

The *ihpds* command reads a PDS image label file, locates the image, and writes it out to the standard output, or to the file specified by the **–o** flag, in GIPS image format.

–h write only the image header to the standard output, removing carriage returns and adding newlines where necessary.

–l instead of converting the image itself, write a list of image parameters to the standard output.

–v while converting the image, write the name and dimension of the image file to the standard error stream, *stderr*.

–S write processing statistics to *stderr*.

–o *file* overrides the default destination for the output GIPS image (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the output image.

FILES

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

SEE ALSO

gipstool(1)

Standards for the Preparation and Interchange of Data Sets, T. Z. Martin, *et al.*, NASA Planetary Data System, Version 1.1, Jet Propulsion Laboratory, 1989.

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

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

- *file*: unknown label format
- bad bits/pix: *count*
- bad histogram format: STREAM
- bad image format: *code*
- bad image format: STREAM
- image location not specified in PDS label
- *keyword*: bad value: *value*
- multiple input file(s): *name*
- *text*: bad image structure
- unsupported encoding type: *code*

NAME

ihpho – create and update a cellular GIPS image file

SYNOPSIS

ihpho [**–DHSZ**] [**–c** *color*] [**–m** *file*] [**–x** *num*] [**–y** *num*] *image* [*script*]

DESCRIPTION

The *ihpho* program creates and/or updates a GIPS cellular *image* file with a mixture of sub-images, graphics, and text, specified through a sequence of sub-commands read from a *script* file. If *script* is omitted, the sub-commands will be read from the standard input stream, *stdin*.

If the *image* file doesn't exist, it will be created with the number of pixels specified by the **–x** and **–y** options, and each pixel will be initialized to the value of the **–c** option. In this situation, **–x** and **–y** must be specified, and if **–c** is omitted, the pixels will be initialized to zero. When invoked with the **–H** flag, *ihpho* creates an image containing two-byte pixels. Otherwise, the image will contain one-byte pixels. If *image* already exists, **–x** and **–y** will be ignored. In either case, the **–c** value becomes the default drawing color, unless subsequently changed by the *color* sub-command.

The image can be written as a VICAR2 image or Sun Rasterfile by the *vicarfile* and *rasterfile* subcommands. These output images may be either monochrome (one-byte pixels) or full-color (three-byte pixels). There is no connection between the number of bytes-per-pixel in the photoproduct image file (1 or 2) and the number in the VICAR2 or rasterfile (1 or 3). The conversion is performed by table lookup from an external *colormap* file.

OPTIONS

–D generates debugging output, listing each command before executing it.

–H any newly created image file will contain 2 bytes per pixel. An existing image file must have been created with 2 bytes per pixel.

–S writes command statistics to the standard error stream, *stderr*, after successful program execution.

–Z override input pixel length checking when loading an image via the *image* command. Unless this flag is specified, *ihpho* will only load images whose pixel size matches the output image, i.e. one byte per pixel unless the **–H** flag is used, in which case 2 bytes per pixel.

–c *color* specifies the default color value for subsequent sub-commands. New images will have all pixels set to this value. It may subsequently be changed by the *color* subcommand.

–m *file* specifies the name of a GIPS file whose header variables are to be merged with a new cellular *image* file. It is illegal to use this option if the *image* file already exists.

–x *num* specifies the number of *x*–pixels in a new *image* file. If *image* already exists, the **–x** value will be ignored.

–y *num* specifies the number of *y*–pixels in a new *image* file. If *image* already exists, the **–y** value will be ignored.

SUBCOMMANDS

The *ihpho* sub-commands should be entered on separate lines. White space surrounding sub-command text will be ignored. Blank lines and lines whose first character is '#' will be ignored. Sub-command lines can be continued by ending them with a back-slash character.

arc *x y radius angle1 angle2 width*

draw an arc of given radius and width, centered at *x,y*. subtending angles between *angle1* and *angle2* from the vertical (y-axis). If the width is zero, extend the arc into a wedge.

bold *name*

set the font to be used to write boldface text. This will be selected by **fB** or **f3** within strings in **text** or **label** commands.

circle *x y radius width*

draw a circular ring of given radius and width, centered at *x,y*. If the width is zero, draw a solid circle.

color *color*
set the color used for subsequent text and graphic items, an integer between 0 and 255 (for single-byte pixels) or between 0 and 65535 (for two-byte pixels).

colormap *file*
read a file containing a series of lines specifying a triplet of *red*, *green*, and *blue* intensities in the range 0 through 255 to use when writing *vicar* or *rasterfile* image files. If *ihpho* is using one byte per pixel, the colormap file will contain 256 lines; if the **–H** argument was used, the colormap file must contain 65536 lines. Null lines and lines beginning with '#' are comments, and will be ignored.

font *file* reset the text font file. If a relative pathname is used, the current working directory will be searched first, then "*/usr/lib/vfont*", and finally "*/usr/lib/fonts/fixedwidthfonts*".

header *file*
read the header from the GIPS image *file*. Subsequent *latitude* and *longitude* sub-commands will then work correctly, without the overhead of reading the entire image file. Alternatively, if *file* begins with the vertical bar `|´, the remainder of the line will be executed as a shell command, and the output of this command interpreted as a GIPS image header that defines the action of subsequent *latitude* and *longitude* sub-commands.

image *x y file*
insert the GIPS image from *file*, with its top left corner at *x,y*. Alternatively, if *file* begins with the vertical bar `|´, the remainder of the line will be executed as a shell command, and the output of this command interpreted as a GIPS image to be inserted in the image file. Input image pixels are inserted into the output file without any translation. If the **–Z option is specified, input pixels** zero-filled on the left to match the output size. Otherwise, the input and output pixels must have the same length.

include *file*
temporarily suspend reading the input file and execute the commands in *file*. *Include* commands cannot be nested, i.e. an included file cannot contain *include* commands. During execution of an included file, the action of the **–D** debug flag is suppressed.

italic *name*
set the font to be used to write italic text. This will be selected by **\fI** or **\f2** within strings in **text** or **label** commands.

label *x y string*
insert the text string at *x,y*. For origin codes 0, 4, and 5, the text will be vertically aligned to its baseline.

latitude *lat lon1 lon2 incr width*
draw a parallel of latitude at *lat* degrees between longitudes *lon1* and *lon2* degrees. The curve will use straight-line segments of *incr* degrees of longitude, and *width* specifies the line width in pixels. The information required to map latitude and longitude into the image array is derived from the most recent GIPS header read by an *image* or *header* sub-command.

line *x1 y1 x2 y2 width*
draw a straight line of given width between *x1,y1* and *x2,y2*.

longitude *long lat1 lat2 incr width*
draw a meridian of longitude at *long* degrees between latitudes *lat1* and *lat2* degrees. The curve will use straight-line segments of *incr* degrees of latitude, and *width* specifies the line width in pixels. The information required to map latitude and longitude into the image array is derived from the most recent GIPS header read by an *image* or *header* sub-command.

origin *code*

reset the text origin code. See *comlabel*(3) for details.

rasterfile *shrink depth name*
copy the photoproduct file to the Sun raster file *name*, as a monochrome or RGB image, depending on whether *depth* is 1 or 3. The image is reduced (by nearest-pixel resampling) by a factor of *shrink*, which must be a positive integer. If *name* begins with `|´, the remainder of the name is interpreted as a UNIX command which is processed by the Bourne shell (*/bin/sh*), which is supplied the raster file as its standard input stream, *stdin*.

rect *x1 y1 x2 y2 width*
draw a rectangular frame of given width with outer corners at *x1,y1* and *x2,y2*. If the width is zero, draw a solid rectangle.

reset *x y* reset the drawing origin to *x,y*. Until the next *reset*, *x,y* addresses in subsequent commands will be interpreted relative to this origin.

symbol *name*
set the font to be used to write symbol characters. This will be selected by **fS** or **f4** within strings in **text** or **label** commands, or when special **(cc** strings are interpolated.

text *x y string*
insert the text string at *x,y*. For origin codes 0, 4, and 5, the text will be vertically aligned to its mid-point. See *comlabel*(3) for information on how to select special characters and how to change font size and character position within *string*.

vicarfile *depth name volser file*
copy the photoproduct file to the VICAR2 file *name*, as a monochrome or RGB image, depending whether *depth* is 1 or 3. If *name* begins with "*/dev/*", *ihpho* assumes that the output is to tape, and creates ANSI standard labels (level 3), with volume serial number *volser*, and file name *file*.

vwedge *x1 y1 x2 y2 color1 color2 [m v]*
draw a rectangular *color wedge* between *x1,y1* and *x2,y2*, with colors running from *color1* on the top to *color2* on the bottom. If *m* and *v* are defined, the color value is then masked with *m* (logical AND), and then the value of *v* is logically OR'ed with the result.

wedge *x1 y1 x2 y2 color1 color2 [m v]*
draw a rectangular *color wedge* between *x1,y1* and *x2,y2*, with colors running from *color1* on the left to *color2* on the right. If *m* and *v* are defined, the color value is then masked with *m* (logical AND), and then the value of *v* is logically OR'ed with the result.

ximage *x y file*
is identical to the *image* subcommand except that only non-zero valued pixels of the input image will replace pixels in the photoproduct file. Non-zero input image pixels are inserted into the output file without any translation. If the **–Z option is specified, input pixels** zero-filled on the left to match the output size. Otherwise, the input and output pixels must have the same length.

AUTHOR

Peter G. Ford, MIT CSR.

SEE ALSO

comlabel(3), gcell(3), ihgfmt(1), rasterfile(5), vfont(5)

DIAGNOSTICS

- **bad arg:** *arg*
  too many *ihpho* arguments specified.
- **bad args:** *text*
  an unexpected error occurred when parsing the specified subcommand.
- **bad** #*n* **arg:** *text*

there is an error in subcommand argument number *n*.

- *file:* **bad command:** *text*
  the command word is illegal.
- *file:* **bad dtype:** *dtype*
  the secondary image file is in cellular format, or it has more than one byte per pixel.
- **bad frame number:** *frame*
  the *number* argument of the **frame** command must be 1, 2, or 3.
- **bad** *x/y* **origin:** *org*
  the coordinates of the *reset* command must be within the image.
- **illegal nesting:** *file*
  the named file cannot be **included** within an included file.
- **no photoproduct file opened**
  an updating command was specified before any image was created or initialized by a prior **open** command.

NAME

ihproj – apply a non-local projection to a General Image

SYNOPSIS

ihproj [–*nproj*] [**–DSf**] [**–R** *fact*] [**–XY** *org*] [**–d** *dir*] [**–mo** *file*] [**–l** *long,lat,rho*] [**–Pp** *ang*] [**–r** *rad*] [**–bnxy** *num*] [*name*]

DESCRIPTION

The input General Image in *name* is transformed into a given cartographic coordinate system. If *name* is omitted, the input is taken from the standard input stream, *stdin*. *Ihproj* first calls *ihgfmt*(1) to reformat the input image into a form more suitable for random access, and then calls *ihgrid*(1) to generate transformation coefficients and *ihgeom*(1) to apply those coefficients to the random access file. *Ihproj* writes its output to the file specified by the **–o** option, or, if omitted, to the standard output stream, *stdout*. The following options may be specified:

–*nproj* specifies the desired transformation. See the manual entry under *ihgrid*(1) for possible values.

–D generates "debug" output from *ihgrid* and *ihgeom* on *stderr*, detailing set-up parameters.

–P *ang* specifies the second standard parallel (in degrees) for Lambert Conformal (–7) and Albers Equivalent (–12) transformations.

–R *fact* multiplies the default transform scaling factor by the floating-point quantity *fact*, thereby "fine tuning" the size of the output image. Alternatively, the user may supply a scaling factor via the **–r** option (qv.) This option is ignored for tiepoint transformations (–0).

–S writes a line of UNIX "statistics" to *stderr* upon command completion.

–X *org*
–Y *org* specify, respectively, the *x* and *y* origins of the transformed image in the output pixel space. This is, typically, the point into which the input origin defined by the **–l** option is mapped, although this is not true for all transformations. These options are ignored for tiepoint transformations (–0).

–b *num* specifies the number of buffer rasters to be used. The default value is 32. The larger the value, the more efficient the transformation, but *ihproj* will use more memory. The size of the actual output buffer allocated by *ihproj* will be this number times the length of each raster.

–f *dir* specifies the directory that *ihproj* will use for its scratch files. These files will occupy approximately the same space as the original input image itself.

–l *long,lat,rho*
defines the Euler angles (in degrees) of a rotation to be applied to the (longitude,latitude) coordinate system before the Cartographic transformation is applied. The origin of longitude will become *long*, the origin of latitude will become *lat*, and the resulting figure will be rotated clockwise by *rho* . If unspecified, these angles will be set to zero. This option will be ignored for tiepoint transformations.

–m *file* specifies the name of another General Image file whose header is to be merged with that read from *name* or *stdin* before the mapping is begun.

–n *order* is used only with tie-point transformations, and specifies the order of bi-linear polynomials to be used.

–o *file* the transformed General Image will be written to *file*, rather than to the standard output stream *stdout*.

–p *ang* specifies the central latitude (or first standard parallel) of the transformed image (in degrees). If this option is omitted, *ihproj* uses the arithmetic mean of the realworld *y*-axis limits of the input General Image. It is ignored for tiepoint transformations (–0).

–r *rad* specifies the transform scaling factor – whose meaning depends on the particular non-tiepoint transformation selected. If not specified, *ihproj* will compute a value that attempts to maximize the area of input image mapped into the output image. Some clipping and/or bordering is almost guaranteed! Use the **–R** option for fine tuning.

–x *num*
–y *num* specify, respectively, the x– and y–pixel extents of the output image.

name specifies the name of the General Image file to be transformed. If omitted, *ihproj* expects to read the image (in fact, its header only) from the standard input stream "*stdin*".

FILES

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

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihgeom(1), ihgfmt(1), ihgrid(1), gtrans(3)

Ford, P., *Guide to GIPS*, MIT Center for Space Research, 1984.

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

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

DIAGNOSTICS

When an error is detected, *ihgeom* halts with system code 1, writing a message to the standard error stream *stderr*. Possible error messages are

- bad –n value: *value*
- bad arg: *value*
- forward mapping for *projection* not implemented
- illegal numeric in input stream: *code*
- inverse mapping for *projection* not implemented
- mapping is singular for this origin choice
- no –P value specified
- setup for *projection* not implemented
- singular tie-point transformation
- unable to compute –r value

NAME

ihrot – rotate, reverse, or invert a General Image

SYNOPSIS

ihrot [–*size*] [**–inr**] [**–d** *dir*] [**–hmo** *file*] [*name*]

DESCRIPTION

The *ihrot* command reads a General Image from file *name* and rotates it by 90 degrees in the counter-clockwise sense. If *name* is omitted, input is taken from the standard input stream *stdin*.

–*size* specifies the maximum number of bytes to allocate to the in-core buffer used to rotate the image. The default is ¼ megabyte, i.e. "–*262144*". Larger images will be rotated by writing them into a series of temporary scratch files.

–o *file* overrides the default output destination (the standard output stream *stdout*), and directs it to *file* instead.

–r reverses each input raster from right to left before rotation.

–i inverts the input image from top to bottom before rotation.

–n instructs *ihrot* to perform no rotation; the **–i** and/or **–r** flags will then invert an image without rotating it.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the input image. If the image is to be rotated (**–n** flag omitted), any file specified by a **–h** option is read *before* the *x* and *y* header items are swapped. Conversely, any **–m** file is read *after* swapping the fields, but before commencing the rotation.

–d *dir* overrides the default directory "/*tmp*" into which *ihrot* will write scratch files if the rotation cannot be performed in core.

FILES

/tmp/ihr*n	default temporary spool file(s)
/usr/gips/tables/cprofile	image header definition file

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihread(3)

DIAGNOSTICS

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

- multiple input files
- *item* < 1
- bad operation for this *item*
- bad dtype value: *value*
- cannot invert
- interrupt

NAME

ihs – strip header and convert image to 1-byte per pixel format

SYNOPSIS

ihs [**–cinqs**] [**–hm** *file*] [**–fo** *file*/#] [**–** *size*] [**–l** *length*] [**–xy** *org*] [**–XY** *lim*] [*name*]

DESCRIPTION

ihs reads an image file from *name*, or, if *name* is omitted, from the standard input stream *stdin*. It converts it to byte-format (one pixel per byte) using the information contained in the image header. If the image has been passed through *ihfft*(1) so that *dtype=c*, each complex pixel will be replaced by its absolute value before further conversion. What happens next to each pixel (*pix*) is determined by the value of header variable **dscale**:

- **s** $pix \rightarrow \text{float}(pix)$
- **n** $pix \rightarrow dfact \times pix + dzero$.
- **l** $pix \rightarrow \log_{10}(pix)$
- **p** $pix \rightarrow dparm_0 + dparm_1 \times pix + dparm_2 \times pix \times pix + ...$ (*npoly*+1 terms)
- **f** $pix \rightarrow n,\ f[n] \le pix < f[n+1]$, for *f* in the file *dfile*.

In all cases except *dscale=f*, *pix* is further transformed by

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

where **dmax** and **dmin** default to 0 and 256, respectively, if not defined in the header. The above description is purely symbolic – *ihs* optimizes the conversions whenever possible by constructing lookup tables, using bit-shifts when multiplying or dividing by powers of 2, etc. etc.

–size sets the output blocksize, in bytes, when no re-formatting is to be performed (the **–s** option), and no header generated (i.e. no **–h** option specified).

–c specifies that the input file (or pipe) is not read past that point when the output array has been written. If the input came from a pipe, this may result in a "broken pipe" error message.

–h *file* the header will be saved in the named file. To write the header to the standard output, use "**–h –**". If the **–o** flag specifies that the image is being written directly to the display device, the header will not be written to the standard output until *after* the image is complete; otherwise, the header is written *before* the image. This logic ensures that *ihs* and *iged* do not attempt to write to the display at the same time.

–q eliminates certain warning messages (see under "Diagnostics", below).

–s bypasses all image reformatting. The input image array is copied, without any alteration whatever, to the output designated by the **–o** option. This flag may therefore be used to remove a header from an image. The output blocksize may be specified via the *–size* option.

–f *file* writes to *file* a *stretch-table* consisting of 256 unsigned short integers. The table is generated from a histogram of output 8-bit pixel values, and may be used to maximize the usable contrast of the displayed image. If *file* is numeric, the table is written directly to the stretch table of that number in the display device itself.

–m *file* specifies the name of a General Image file whose header will be merged with that of the input image.

–n instructs *ihs* not to process the input image itself, but to write the output header to the output stream defined by the **–h** option. This is a useful way to extract a header from a General Image. It is equivalent to "*gin -n*".

–l *size* specifies a maximum line length (in bytes) for the output header. This is most useful in conjunction with the **–n** option; i.e. "*ihs –n –l1 name*" writes the header of General Image file *name* to the standard output with one item per line.

–o *file* specifies that the stripped image will be written to the file specified, or, if omitted, to the standard output *stdout*. If the file name is numeric, the image will be written directly to the display image plane of that number.

–i specifies that the image will be *inserted* into the image plane – surrounding pixels will not be cleared; otherwise, the entire image plane will be rewritten. This option is only valid when the output is written directly to an image plane.

–x *org*
–y *org* override any definition of image origin supplied by the header variables *xorg* and *yorg*.

–X *lim*
–Y *lim* override the image display width variables *xlim* and *ylim*.

FILES

/dev/image.∗	image plane pseudofile (see *comsubs*(3)).
/dev/function.∗	function table pseudofile (see *comsubs*(3)).
/usr/gips/tables/cprofile	image header definition table

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihc(1), iged(1)

DIAGNOSTICS

If the image display will be truncated by the particular choice of *org* or *lim* variables, the message

"**ihs: warning - image truncated on:** *left*/*right*/*top*/*bottom*"

is written to the standard error stream *stderr*, unless the **–q** (quiet) option is specified.

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

- multiple input files: *file*

BUGS

The *ihs* command is only implemented (fully) for the Comtal Vision ONE/20 display system, and (partially) for the AED 767 frame buffer.

NAME

ihstokes – translate a JPL AIRSAR image to GIPS image format

SYNOPSIS

ihstokes [**–DLSb**] [**–c** *te,ta,re,ra*] [**–mo** *file*] [*name*]

DESCRIPTION

The *ihstokes* command reads a JPL AIR-SAR image file, selects a particular radar polarization, and writes it out to the standard output, or to the file specified by the **–o** flag, in GIPS image format, with *dtype=f* (float) pixels.

–S selects verbose mode and writes the following statistics to the standard error stream, *stderr*:

- Transmitted phase and eccentricity
- Transmitted Stokes Vector
- Received phase and eccentricity
- Received Stokes Vector

–L write the output in deciBels, instead of the default which is linear in cross-section.

–S writes processing statistics to *stderr*.

–c *te,ta,re,ra*

specifies the polarization states desired in the output image:

te	transmitted eccentricity
ta	transmitted phase (degrees)
re	transmitted eccentricity
ra	transmitted phase (degrees)

The default value for all 4 values is 0.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the output image.

–o *file* overrides the default destination for the output GIPS image (the standard output stream *stdout*), and directs it to *file* instead.

FILES

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

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

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

- bad –c value: *text*
- cannot find start of data
- image dimensions missing from header
- multiple input files
- not a Stokes image

NAME

ihtofits – translate a GIPS image into FITS format

SYNOPSIS

ihtofits [**–bS** *file*] [**–mo** *file*] [*name*]

DESCRIPTION

The *ihtofits* command reads a General Image from file *name*, updates the header from an file specified by the **–m** option, and writes it out to the standard output, or to the file specified by the **–o** flag, in "*fits*" format (Flexible Image Transport System). This operation is essentially the reverse of that performed by the *ihfromfits*(1) command.

–S *file* writes processing statistics to *stderr*.

–o *file* overrides the default destination for the output *fits* file (the standard output stream *stdout*), and directs it to *file* instead. When written to a tape or pipe, the blocksize will always be the *fits* standard, i.e. 2880 bytes.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the input image.

FILES

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

SEE ALSO

ihfromfits(1)

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

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

- multiple input files

NAME

ihtogif – translate a GIPS image into GIF format

SYNOPSIS

ihtogif [**–MS**] [**–amo** *file*] [*name*]

DESCRIPTION

The *ihtogif* command reads a General Image from file *name*, updates the header from an file specified by the **–m** option, and writes it out to the standard output, or to the file specified by the **–o** flag, in Compuserve's GIF format. This operation is essentially the reverse of that performed by the *ihfromgif*(1) command.

–M transform the specified RGB colormap into a greyscale map, preserving the luminance of each pixel.

–S writes processing statistics to *stderr*.

–a *file* appends the *colormap* file *file* to the output *rasterfile*. See the manual entry for *gipstool*(1) for the format of a *colormap* file.

–o *file* overrides the default destination for the output *rasterfile* (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the input image.

FILES

/usr/gips/tables/cprofile	image header definition file
/usr/gips/lib/cmap	directory containing colormap files

SEE ALSO

ihfromgif(1)

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

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

- bad colormap record: *text*
- multiple input file(s)
- unable to write GIF file

NAME

ihtopix – translate a GIPS image into Sun raster format

SYNOPSIS

ihtopix [**–S**] [**–amo** *file*] [*name*]

DESCRIPTION

The *ihtopix* command reads a General Image from file *name*, updates the header from an file specified by the **–m** option, and writes it out to the standard output, or to the file specified by the **–o** flag, in Sun "*rasterfile*" format. This operation is essentially the reverse of that performed by the *ihfrompix*(1) command.

–S writes processing statistics to *stderr*.

–a *file* appends the *colormap* file *file* to the output *rasterfile*. See the manual entry for *gipstool*(1) for the format of a *colormap* file.

–o *file* overrides the default destination for the output *rasterfile* (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the input image.

FILES

/usr/gips/tables/cprofile	image header definition file
/usr/gips/lib/cmap	directory containing colormap files

SEE ALSO

gipstool(1), ihfrompix(1), rasfilter8to1(1), rasterfile(5), rastrepl(1)

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

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

- multiple input files

NAME

ihtrans – convert a General Image generated by an unlike CPU

SYNOPSIS

ihtrans [**–S** *file*] [**–mo** *file*] [*name*]

DESCRIPTION

The *ihtrans* command reads a General Image from file *name*, updates the header from an file specified by the **–m** option, and writes it out to the standard output, or to the file specified by the **–o** flag, translating the pixel fields to or from a *version* that is not supported on the local processor.

This command can be used to convert a General Image that is piped through the *rsh* command to or from a remote node of a network. To be consistent in an environment of unlike or unknown processors, *ihtrans* should always be the first command executed by *rsh* and the standard output of *rsh* should be immediately piped through *ihtrans*.

? Invoked with a single "?" argument, *ihtrans* writes a table of known *version* and *dtype* combinations to *stdout*.

–S *file* writes processing statistics to *stderr*.

–o *file* overrides the default output destination (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the input image.

FILES

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

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

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

- input conversion error: *file*
- output conversion error: *file*
- unsupported dtype: *char*
- unsupported version: *code*

NAME

ihvicar – translate a VICAR-2 file to GIPS image format

SYNOPSIS

ihvicar [**–bDS**] [**–mo** *file*] [*name*]

DESCRIPTION

The *ihvicar* command reads an image file in VICAR-2 format, writes it out to the standard output, or to the file specified by the **–o** flag, in GIPS image format.

–b byte-swap integer pixel fields. This operation may be necessary when reading VICAR-2 images that were written by a CPU that writes integers in a different format.

–D write the VICAR-2 header keywords and their values on the standard error stream, *stderr*.

–S write processing statistics to *stderr*.

–o *file* overrides the default destination for the output GIPS image (the standard output stream *stdout*), and directs it to *file* instead.

–m *file* specifies the name of an existing General Image file whose header will be merged with that of the output image.

FILES

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

SEE ALSO

gipstool(1)

AUTHOR

Peter G. Ford, MIT CSR

DIAGNOSTICS

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

- VICAR label error in byte *n*
- bad *keyword* value
- *file*: unexpected EOF
- illegal FORMAT value: *format*
- keyword not in header: *keyword*
- multiple input files
- non-numeric *keyword* value
- non-string *keyword* value

NAME

xgips – display a GIPS image in an X window

SYNOPSIS

xgips [*Xview options*] [*gipstool arguments*]

DESCRIPTION

Xgips reads a general image file from *name*, and displays the image in an X window. It is a direct port of the *gipstool*(1) command to the XView application interface. All options and modes of that command are implemented, and this manual file only deals with those that differ in their effects.

The principal difference lies in the allocation of colors. The *xgips* default is to allocate 64 to the image (*gipstool* allocates 128), and their definitions are taken from every fourth entry in any specified colormap file. These colors **cannot** be changed during program execution, i.e. the **B**, **F**, **M**, **S**, and **s** subcommands are disabled. If the –N option is specified on the *xgips* command line, a "dynamic" colormap will be obtained and the colors can be changed, but at the penalty that the colors will only be displayed correctly when *xgips* has the "colormap focus". Some window managers, e.g. *olwm*(1), permit this focus to be retained intependent of the window focus itself.

Another difference lies in the way that *xgips* starts up. In order to be able to display the image as it is being read into the program, it is necessary to obtain the "input focus" directly at startup time. For this reason, it it is necessary that the cursor points within the image window when it first appears on the screen. Although *xgips* tries to place the cursor in the center of the window, some X window managers do not permit this to happen, and the user must physically move the cursor to point there.

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

"*Guide to GIPS" and "A GIPS Tutorial", by Peter Ford, MIT Center for Space Research,* 1990.
gipstool(1)

BUGS

The colormap handling commands should be re-written to use Xlib subroutine calls directly to manage individual color cells.

NAME

comheight, comlabel commarker, comwidth, memlabel, memmarker, setauxfont, setfont, setink, setorigin, settype – write text to image/graphic file or array

SYNOPSIS

char *setfont(name)
char *name;

char *setauxfont(name,fontnum)
char *name;

char setink(c)
char c;

settype(type)

comlabel(fid,nlines,ncols,baseline,basecol,label)
char *label;

memlabel(array,nlines,ncols,baseline,basecol,label)
char *array, *label;

commarker(fid,nlines,ncols,centerline,centercol,label)
char *label;

memmarker(array,nlines,ncols,centerline,centercol,label)
char *array, *label;

setorigin(origin)

comheight(label)
char *label

comwidth(label)
char *label

cc ... –lGI

DESCRIPTION

These routines are used for labeling images and graphics. They are contained in the library *'/usr/local/lib/libGI.a'* and should therefore be loaded at compile/load time by using the *–lGI* option.

Setfont sets the named (Versatec/Varian-type) font for use in making labels. If *name* is **NULL**, a pointer to a static character array that holds the name of the current font-definition file is returned. Otherwise, *setfont* returns either **NULL** to indicate that something wild happened, or a pointer to a static character array that holds the name of the previous font.

Setfont will also attempt to figure out from *name* the point size, rotation sense, and highlight level of the new font, and, if it is a **roman** type, will load the **italic**, **bold**, and **symbol** fonts of the same style, size, and orientation. If not **roman**, *setfont* only attempts to locate a **symbol** font of the same size and orientation.

Setauxfont is used to change the current definition of only one of the four possible fonts, without affecting the other three. It is passed the font name and the integer *fontnum* from 0 through 3 representing, respectively, **roman**, **italic**, **bold**, or **symbol**. If *name* is **NULL**, *setauxfont* returns a pointer to the name of the previous font of this type (or **NULL** if there was none). Otherwise, it returns **NULL** if unable to locate the specified font, or a pointer to the font filename.

Setink stores its argument as the pixel value to use in producing characters within images. When writing to graphic planes, a non-zero argument indicates that bits are to be turned on, a zero argument that they are to be turned off. *Setink* returns the previous *ink* value.

Settype indicates whether the *array* or *fid* variables passed to subsequent invocations of *comlabel, memlabel*, *commarker*, or *memmarker* refer to an 8-bits-per-pixel image (*type=0*), or to a 1-bit-per-pixel graphic (*type!=0*). *Settype* returns the previous value of *type*.

Comlabel uses *fid* as a file descriptor for a readable and writable file associated with an image or graphic which is taken to be *nlines* lines by *ncols* columns of 1-byte pixels. The label specified by *label* (a null-character-terminated string) is written to the image or graphic in the previously defined font using *baseline* and *basecol* as the starting point of the label, and, for images only, using the pixel value defined in the last call to *setink*.

Memlabel is like *comlabel* except that instead of modifying an external image or graphic, it modifies an array in main computer memory.

Commarker and *memmarker* are like *comlabel* and *memlabel* except that the *centerline* and *centercol* of the characters making up the label are specified rather than the base line and base column.

Setorigin specifies which portion of the string is to be aligned with the chosen *centerline* and *centercol* arguments in the calls to *commarker* and *memmarker*. The argument *origin* should be in the range **0** through **8**, and causes *centerline* and *centercol* to be mapped according to the following table:

1	2	3
4	0	5
6	7	8

Thus, specifying *setorigin(0)* causes text to be centered, *setorigin(3)* places text wholly below and to the left of *centerline* and *centercol*, and so forth. *Setorigin* returns the previous value of *origin* (whose initial value is 0). To determine the current value without changing it, specify a value of *origin* lying outside the permitted range.

Fonts whose names end in the character 'r' are taken to be rotated fonts; labels written in such fonts go down the array rather than across it.

Comheight and *comwidth* return, respectively, the maximum height and width (in pixels) of their calling strings, when written into an image or graphic with the currently defined font. For rotated fonts, *comheight* returns the maximum height of rotated characters, i.e. reading left to right across the image.

Two fonts have been developed for use with *comlabel:* '**7seg.2**' provides numerals only in a fixed-width five line by three column format; '**5by5.2**' provides all ASCII characters in a fixed-width five line by five column format. The rotated versions of these fonts are also available.

ESCAPE CODES

Alternate fonts may be selected, and the location and size of characters in the graphic or image may be altered by the inclusion of special escape sequences within the strings passed to *comlabel*, *memlabel*, *commarker*, and *memmarker*. The syntax and action are identical to the following functions of the *troff*(1) typesetter, except that their function only lasts for the duration of the current string:

\fR	switch to Roman font
\fI	switch to Italic font
\fB	switch to Bold font
\s*n*	change point size to *n*
\s+*n*	increase point size by *n*
\s–*n*	decrease point size by *n*
\s0	resume default point size
\u	shift up one half line
\d	shift down one half line
\h*n*	shift up to *n* pixels above default
\h+*n*	shift up by *n* pixels
\h–*n*	shift down by *n* pixels
\h0	resume default vertical position
\v*n*	shift right to *n* pixels from default
\v+*n*	shift right by *n* pixels
\v–*n*	shift left by *n* pixels
\v0	resume default horizontal position

\\(*xx*	insert symbol character *xx* (see below)
\e	insert a back slash
\\'	insert a back quote

The following escape sequences may be used to specify special characters in all fonts:

\\(em	—	Em dash	\\(hy	-	hyphen	\\(bu	•	bullet
\\(sq	□	square	\\(ru	_	rule	\\(14	¼	1/4
\\(12	½	1/2	\\(34	¾	3/4	\\(fi	ﬁ	ﬁ
\\(fl	ﬂ	ﬂ	\\(ff	ff	ff	\\(Fi	ﬃ	ﬃ
\\(Fl	ﬄ	ﬄ	\\(de	°	degree	\\(dg	†	dagger
\\(fm	′	foot mk	\\(ct	¢	cent	\\(rg	®	registered
\\(co	©	copyright						

Those that are only defined in a symbol font are:

\\(pl	+	math +	\\(mi	−	math −	\\(eq	=	math equals
\\(**	∗	math ∗	\\(sc	§	section	\\(aa	´	acute accent
\\(ga	`	grave ‘	\\(ul	_	u/rule	\\(sl	/	slash
\\(*a	α	alpha	\\(*b	β	beta	\\(*g	γ	gamma
\\(*d	δ	delta	\\(*e	ε	epsilon	\\(*z	ζ	zeta
\\(*y	η	eta	\\(*h	θ	theta	\\(*i	ι	iota
\\(*k	κ	kappa	\\(*l	λ	lambda	\\(*m	μ	mu
\\(*n	ν	nu	\\(*c	ξ	xi	\\(*o	ο	omicron
\\(*p	π	pi	\\(*r	ρ	rho	\\(*s	σ	sigma
\\(ts	ς	sigma	\\(*t	τ	tau	\\(*u	υ	upsilon
\\(*f	φ	phi	\\(*x	χ	chi	\\(*q	ψ	psi
\\(*w	ω	omega	\\(*A	Α	Alpha	\\(*B	Β	Beta
\\(*G	Γ	Gamma	\\(*D	Δ	Delta	\\(*E	Ε	Epsilon
\\(*Z	Ζ	Zeta	\\(*Y	Η	Eta	\\(*H	Θ	Theta
\\(*I	Ι	Iota	\\(*K	Κ	Kappa	\\(*L	Λ	Lambda
\\(*M	Μ	Mu	\\(*N	Ν	Nu	\\(*C	Ξ	Xi
\\(*O	Ο	Omicron	\\(*P	Π	Pi	\\(*R	Ρ	Rho
\\(*S	Σ	Sigma	\\(*T	Τ	Tau	\\(*U	Υ	Upsilon
\\(*F	Φ	Phi	\\(*X	Χ	Chi	\\(*Q	Ψ	Psi
\\(*W	Ω	Omega	\\(sr	√	sqrt	\\(rn	‾	root extender
\\(>=	≥	>=	\\(<=	≤	<=	\\(==	≡	identic. equal
\\(~=	≅	app =	\\(ap	~	approx	\\(!=	≠	not equal
\\(->	→	rt arr	\\(<-	←	lt arr	\\(ua	↑	up arrow
\\(da	↓	dn arr	\\(mu	×	mult	\\(di	÷	divide
\\(+-	±	plus-mi	\\(cu	∪	union	\\(ca	∩	intersection
\\(sb	⊂	subset	\\(sp	⊃	superst	\\(ib	⊆	improper subset
\\(ip	⊇	imp sup	\\(if	∞	infin	\\(pd	∂	partial deriv.
\\(gr	∇	grad	\\(no	¬	not	\\(is	∫	integral sign
\\(pt	∝	prop to	\\(es	∅	empty	\\(mo	∈	member of
\\(br	|	vert	\\(dd	‡	2dagger	\\(rh	⇒	right hand
\\(lh	⇐	lt hand	\\(bs		Bell	\\(or	|	or
\\(ci	○	circle	\\(lt	⎧	top {	\\(lb	⎩	bottom {
\\(rt	⎫	top }	\\(rb	⎭	bot }	\\(lk	⎨	center {
\\(rk	⎬	cent }	\\(bv	⎪	bold |	\\(lf	⎣	bottom [
\\(rf	⎦	bot]	\\(lc	⎡	top [	\\(rc	⎤	top]

If there is any confusion between the end of an escape code and the text that follows it, you may follow the escape code with an extra back-slash. Alternatively, you may enclose numeric escape fields within single quotes, i.e. **\h'+24'2** shifts the remainder of the line by 24 pixels, then inserts a '2', whereas **\h+242** shifts the line upward by 242 pixels.

FILES

/usr/lib/vfont/	directory of font definition files
/usr/local/lib/libGI.a	library containing *comlabel*(3) routines.

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

cfont(1), vfont(5)

ACKNOWLEDGEMENT

comlabel is developed from a routine written by Arthur Olson of the National Cancer Institute, Bethesda. It has been modified to implement multiple fonts, *setorigin()*, *settype()*, and the in-line escape codes.

BUGS

Some rotated fonts are given incorrect height information in their definition file headers.

NAME

color_get, color_put, command_open, function_open, graphic_open, image_open, pseudo_open, target_dump, target_get, target_put, target_roam – interface routines to virtual GIPS display

SYNOPSIS

image_open(ni,mode)

graphic_open(ng,mode)

pseudo_open(np,mode)

function_open(nf,mode)

FILE *command_open()

target_get(nt,xy)
int xy[2]

target_put(nt,xy)
int xy[2]

target_roam(nt)

target_dump(nt)

color_get(ng)

color_put(ng,nc)

cc ... –lFB

DESCRIPTION

These subroutines may be used to interface between C routines and a "*generic*" image or graphic display device. They were written to support the 3M/Comtal Vision ONE/20 image processor, but permit the user to write code that does not depend on the detailed characteristics of the display device.

The four "open" routines return file descriptors that may be used in subsequent calls to *read*, *write*, *lseek*, and *close*. The size and format of individual data elements depend on the nature of the physical display device. Alternatively, the file descriptor may be exchanged for a stream pointer via a call to *fdopen*.

Image_open returns a file descriptor to image plane number *ni*. In addition, that image is displayed and, if possible, pseudocolored.

Graphic_open returns a file descriptor to graphic plane number *ni*. The chosen graphic plane will be displayed.

Pseudo_open returns a file descriptor to pseudocolor lookup table number *np*, and adds that lookup table to the processing pipeline for the displayed image.

Function_open returns a file descriptor to function (contrast stretch) lookup table number *np*, and adds that lookup table to the processing pipeline for the displayed image.

command_open returns a *stream* pointer via which device-dependent commands may be sent to the display device. *command_open* will always return NULL if the display is unavailable.

Target_get returns in the *xy* array the x and y coordinates of target number *nt*.

Target_put repositions target number *nt* to the x and y coordinates specified in the *xy* array.

Target_roam enables "roaming" of target number *nt*, allowing the user to manually reposition the target prior.

Target_dump enables "dumping" of target number *nt*. The meaning of this operation depends on the display device.

Color_get returns a numeric code that defines the color in which graphic plane number *ng* is currently displayed. The mapping between colors and codes is device dependent.

Color_put resets the color in which graphic plane number *ng* is to be displayed to that defined by the numeric code *nc*. The mapping between colors and codes is device dependent.

No particular restrictions are imposed on the order in which these routines are called, but it is recommended that each application program contain an initial call to *command_open* to determine whether the display device is available.

DEVICE DEPENDENT MACROS

In addition to the above routines, the C-language header file "/*usr*/*gips*/*include*/*gips.h*" contains a series of #**define** and **typedef** statements that reflect the physical characteristics of the display device:

ROWS	number of rasters (i.e. y-values) in display device
COLS	number of pixels per raster (i.e. x-values) in display device
NIMAGE	number of bytes in an image plane
NGRAPH	number of bytes in a graphic plane
PSEUDO	is *typedef*'d to a pseudocolor lookup table element.
FUNCTION	is *typedef*'d to a function lookup table element.
ISCOL(*x***)**	true if *x* is a legal column number
ISROW(*y***)**	true if *y* is a legal row number
TOPIX(*x*,*y***)**	returns the element offset to pixel *(x,y)* in an image plane
PIXEL(*i*,*x*,*y***)**	addresses the *(x,y)* pixel in the incore image plane array *i*.
SETGRAPH(*g*,*x*,*y***)**	sets pixel *(x,y)* in the incore graphic plane *g*.
UNSETGRAPH(*g*,*x*,*y***)**	clears pixel *(x,y)* in the incore graphic plane *g*.

COMTAL IMPLEMENTATION

When an interface routine other than *command_open(), color_get(),* or *color_put()* is called, it may direct one or more commands to the display device. These commands may be tailored by the user by placing in his or her home directory a file named "*.clrc*", containing one or more "skeleton" command lines. The following shows the **default** form of these commands:

```
image_open	SU A SU B D I %(ni) A PS
graphic_open	CO G %(ng) %(nc) A G %(ng)
pseudo_open	A PS %(np)
function_open	A F %(nf)
display_put	S D %(x) %(y)
target_get
target_put	A T S TAR %(x) %(y)
target_roam	SU A A B A T RO I
target_dump	SU A A B A T DU I
```

Each line begins with a subroutine name, followed by a single (required) tab, followed by the text (if any) of the command to be sent to the display device whenever that subroutine is called. The constructs "*%(...)*", which actually appear as written in the file, are replaced at execution time by the *values* of the parameters specified in parentheses; i.e. if a call were made to *function_open(2,0),* the command "**A F 2**" (i.e. "ADD FUNCTION MEMORY 2") would be sent to the display device.

The above example displays the *default* values, so you don't need to include any of those exact lines in your own ˜/*.clrc* profile, only the lines that you want to change. For instance, if it irks you that *image_open()* always adds pseudocolor tables to the images that it displays, you could create a file "˜/*.clrc*" containing

```
image_open	SU A SU B D I %(ni)
```

Here is some more information about the individual subroutines, as they have been implemented for the display device.

command_open*()*

If the stream opened by a previous call to *command_open* has since been closed by the user, it is reopened and the display device keyboard input buffer is cleared (i.e. the equivalent of hitting the <**ESC**> key on the keyboard). The user should only close the keyboard stream if it is desirable that another process access the display device while the current process is running (or stopped). Otherwise, it is more efficient if at least one display device feature is kept open throughout the process.

image_open*(ni,mode)*

The returned file descriptor refers to a "pseudofile" of 262144 bytes of physical image plane numbered *ni*, which may be read, written, or lseek'ed, (according to *mode*) but not extended.

graphic_open*(ng,mode)*

The returned file descriptor refers to a "pseudofile" of 32768 bytes of physical graphic plane number *ng*, which may be read, written, or lseeked, (according to *mode*) but not extended. *color_get()* is called to determine the color of the graphic (*nc*). If *nc* is zero (black), it is changed to *15* (white). *color_put(ng,nc)* is then called.

pseudo_open*(np,mode)*

The returned file descriptor refers to a "pseudofile" of 512 bytes of physical pseudocolor memory number *np*, (where np=0,1,2 corresponding to red, green, or blue), which may be read, written, or seeked, (according to *mode*) but not extended.

function_open*(nf,mode)*

The returned file descriptor refers to a "pseudofile" of 512 bytes of physical function lookup memory number *nf*, which may be read, written, or seeked, (according to *mode*) but not extended.

target_get*(nt,xy)*

The physical target coordinates are transformed according to the current zoom and roam settings for physical image number *nt*, and their values returned in the two-element *xy* array.

target_put*(nt,xy)*

The physical target is moved to coordinates given by the two-element *xy* array. If physical image number *nt* is currently roamed or zoomed, the coordinates will be transformed accordingly so as to refer to the physical address within that image plane rather than to the screen coordinates themselves.

color_get*(ng)*

The color number of physical image plane number *ng* is returned, as follows:

0:	black	8:	light pink
1:	blue	9:	magenta
2:	cyan	10:	orange
3:	dark green	11:	pink
4:	dark turquoise	12:	red
5:	forest green	13:	turquoise
6:	green	14:	light orange
7:	yellow	15:	white

color_put*(ng,nc)*

The color of physical graphic plane number *ng* is set according to the value of *nc*, as tabulated above. To alter the command that is sent to the display device, add a line in "˜/*.clrc*" that begins *graphic_open*. The subroutines *graphic_open()* and *color_put()* share this definition.

FILES

/usr/gips/lib/libFB.a	library containing comsubs routines.
˜/.clrc	file containing user profile
/usr/gips/include/gips.h	device definitions and error codes

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihread(3), ihs(1), iged(1)

DIAGNOSTICS

With two exceptions, all the above routines return the value **–1** to indicate that an error has occurred. These exceptions are *command_open* which returns **NULL**, and *target_put* which returns **+1** to indicate that the specified target coordinates are non-physical. (Note: *target_put* returns **–1** if the display device is unavailable). When an error occurs, the external variable *gipserrno* will be set to an error code as defined in *ihread*(3). This will typically be either **GIPSBADOPEN**, **GIPSBADREAD**, or **GIPSBADWRITE**.

BUGS

The current AED version of *comsubs*(3) only implements *image_open(1,mode)*, i.e. a single image plane. *command_open() always returns a valid stream* but all I/O to that stream will be ignored.

NAME

gcell, gclose, ginterp, gopen – access GIPS cellular image

SYNOPSIS

#include "/usr/gips/include/gips.h"

gimage *gopen(stream, nblock)
FILE *stream;

char *gcell(x,y,image)
gimage *image;

char *ginterp(x,y,image)
float x, y;
gimage *image;

gclose(image)
gimage *image;

cc ... –lGI

DESCRIPTION

Gopen sets up a *gimage* storage table, from which subsequent calls to *gcell* and *ginterp* can rapidly access pixel values. *gopen* is passed a pointer *stream* that was returned by a previous call to *ihread*(), which read a General Image file that had been created by *ihgfmt*(1). *Gopen* constructs a series of *nblock* buffers in which to save input blocks. If *nblock* is non-positive, a default of 8 blocks will be used. If *gopen* completes normally, it returns a non-NULL pointer value which should be saved by the caller and used in subsequent calls to *gcell*, *ginterp*, and *gclose*.

Gcell is passed the integer *x* and *y* coordinates of a given pixel, and the value returned by the prior call to *gopen* that saved that image. It returns a pointer to the binary value of the nearest pixel.

Ginterp is similar to *gcell()*, but it is passed the floating-point *x* and *y* coordinate values. It returns a pointer to a pixel value constructed by interpolating the four nearest neighbor image pixels.

Gclose terminates *gcell* calls to a particular General Image. Internal buffers are released and may be made available for subsequent *gopen* calls with other filename arguments.

The actual internal format of *gcell* files is transparent to the user, and may change without notice. The format presently used is that of 1K–byte blocks, chosen so as to optimize disk I/O within UNIX 4.2. Each block represents a rectangular image cell, whose *x* and *y* dimensions vary according to the pixel length, but are always chosen to be powers of 2 to facilitate address decoding. *Nblock* data blocks are held in core simultaneously, and are refreshed from disk on a first-in-first-out basis.

FILES

/usr/gips/lib/libGI.a	library containing GIPS routines.
/usr/gips/include/gips.h	typedefs and error codes

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihgfmt(1), ihread(3), gips(3)

DIAGNOSTICS

When an error is encountered, *gopen* stores an error code in the external variable *gipserrno*, and returns the value NULL. The error codes are documented in *ihread*(3).

When called with coordinates *x* or *y* that lie outside the defined image, *gcell* and *ginterp* return a pointer to the *dnull* value in the image header. *Gcell* returns a *NULL* value when *nimage* does not correspond to the value returned from a (non-failing) *gopen* call.

Gclose returns zero if the image was released, otherwise it stores an error code in *gipserrno*, and returns the value –1.

NAME

gips, egips – translate between GIPS realworld values and pixel coordinates

SYNOPSIS

#**include "/usr/gips/include/gips.h**"

gips(real,pixel,nitem,mode)
double *real;
int *pixel;

egips(pixel,real,nitem,mode)
int *pixel;
double *real;

extern gipserrno;
extern char *gipserrs[];

cc ... –lGI

When called from Fortran, declare **real** to be a double precision array, **pixel** to be an integer array, and **nitem** and **mode** to be integers. Treat *gips*() and *egips*() as integer functions.

When *gips()* is called with *mode=2,* **real** is an array of elements whose data type is given by the *dtype* variable in the current incore General Image header.

DESCRIPTION

These routines translate arrays of General Image realworld coordinate values into their corresponding pixel-space values, and vice versa. Before either routine is called, a legal GIPS image header must be read into core via a call to *ihread*().

nitem specifies the number of items in the *pixel* and *real* arrays. *mode* specifies the data type – **mode=0** for x–axis coordinates, **mode=1** for y–axis coordinates, and **mode=2** for pixel values themselves. Note that each element of *pixel* and *real* must refer to the same type of data – x, y, or d. To convert, for instance, x–axis and y–axis coordinates requires *two* calls to *gips*() or *egips*(), one with **mode=0** and the other with **mode=1.** Alternatively, specify **mode=3,** in which case *real* and *pixel* point to *2*nitem* elements, specifying alternating *x* and *y* values.

After a call to *gips*(), each *x* or *y* pixel value that is outside the limits of the image will be translated to a **real** value of *GIPSLOW* or *GIPSHIGH*, depending on whether it falls on the high or low side of the imaging space.

After a call to *egips*(), any **real** value that is outside the realworld limits of the image will be translated to a *pixel* value of "–1" or "num", where "num" represents the appropriate image dimension, i.e. *xnum*, *ynum*, or *256*, for *mode=0,1,2* respectively.

FILES

/usr/gips/lib/libGI.a	library containing GIPS routines.
/usr/gips/include/gips.h	symbol and error number definitions

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

ihread(3)

DIAGNOSTICS

When an error is encountered, *gips*() and *egips*() return the error codes described in *ihread*(3). These codes are also available to C callers in the external variable *gipserrno*.

NAME

g_setup, g_map, g_unmap – apply non-local transformation to image coordinates

SYNOPSIS

#**include "/usr/gips/include/gips.h"**

int g_setup()

int g_map(lon, lat, xy)
double lon, lat, xy[2];

int g_unmap(x, y, lonlat)
double lonlat[2];

cc ... –lGI

DESCRIPTION

The *g_setup*() function sets up data structures to transform between input and output coordinates. Before calling *g_setup*(), it is first necessary to invoke *ihread()* to read a GIPS header file, and the header variable named "*mapping*" must be initialized to the following character string:

name=*cccc* **type=***n* **origin=***x,y* **radius=***r* **euler=**φ,θ,φ′ **lat=**θ′,θ″ **xyrot=***sw*

cccc Name of projection—unused but must be present.

n Projection number from 0 through 21. See the *ihgrid*(1) manual entry and the ``Guide to GIPS´´ document for more details.

x,y Pixel address in the output image corresponding to the center of the input image, i.e. longitude θ and latitude φ.

r The mapping scale factor, whose meaning depends of the particular transformation selected. Generally, increasing *r* will magnify the output image.

φ,θ,φ′ The Euler rotation angles to be applied to latitudes and longitudes in the input image before the cartographic mapping itself. The rotation brings input longitude θ and latitude φ to the center of the output image (i.e. output pixel address *x,y*), and rotates the result by φ′. These angles are disregarded for tiepoint transformations (type 0).

θ′,θ″ The special latitude or latitudes that have significance for some of the cartographic transformations, e.g. Lambert conic (types 6 and 7) and Albers (types 11 and 12).

sw An optional field. If the value of *sw* is non-zero, the image is given a final 90° counterclockwise rotation.

After a successful call to *g_setup*(), *g_map*() translates from longitude *lon* and latitude *lat* in the input image to *x*–pixel address *xy*[0] and *y*–pixel address *xy*[1] in the output image. If either *lat* or *lon* is invalid, *g_map*() returns a value of 1. Otherwise, it returns 0.

After a successful call to *g_setup*(), *g_unmap()* translates from pixel address *x,y* in the output image to longitude *lonlat*[0] and latitude *lonlat*[1] in the input image. If the specified output pixel address corresponds to a physical input longitude and latitude, *g_unmap*() returns a value of 0. Otherwise it returns 1. No check is made to determine whether a physical latitude and longitude is actually represented in the input image that is being processed, i.e. the user should test that *lonlat*[0] lies between *xmin* and *xmax*, and that *lonlat*[1] lies between *ymin* and *ymax*.

Latitudes, longitudes, and Euler angles are expressed in degrees, except for tie-point transformations (when the *lon*, *lat*, and *lonlat* variables refer to input pixel coordinates), and user-defined projections (when the definition of *lon*, *lat*, and *lonlat* is left to the user).

FILES

/usr/gips/lib/libGI.a	library containing GIPS routines.
/usr/gips/include/gips.h	symbol and error number definitions

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

``Guide to GIPS´´, Peter G. Ford, MIT Center for Space Research report.

ihgrid(1), gips(3), ihread(3)

DIAGNOSTICS

When an error is encountered in reading GIPS header variables, *g_setup*() return one of the integer codes described in *ihread*(3). The error code is also stored in the external variable *gipserrno*.

NAME

gipserr, gipsint, ihbcl, ihbin, ihbout, iherrm, ihget, ihmerg, ihput, ihread, ihtype, ihwrit – manipulate a GIPS image header

SYNOPSIS

#include <stdio.h>
#include "/usr/gips/include/gips.h"

FILE *ihread(name,profile)
char *name, *profile;

FILE *ihmerg(name)

FILE *ihwrit(name,length)
int length;

ihget(item,value*[,lvalue]***)**
char *item, *value;

ihput(item,value)

ihdel(item)

ihtype(name,value*[,lvalue]***)**
char *name, *value;

ihbin(buffer,file,length)
char *buffer;
FILE *file;

ihbout(buffer,file,length)

ihbcl(file)

gipserr();

gipsint(proc);
void (*proc)();

extern gipserrno;
extern char *gipserrs[];

call iherrm

cc ... –lGI

When called from Fortran routines, omit the *lvalue* argument to *ihget()* or *ihtype()*, and treat all the "*ih*" routines as integer functions. If an error condition is detected, "**call iherrm**" will write a disgnostic message to the standard error stream, *stderr*, and then return to the Fortran caller.

DESCRIPTION

Ihread reads a General Image header from file *name*, verifying its contents against the expected format stored in file *profile*. If *name* is the string ("-"), input is taken from the standard input stream *stdin*. If *name* or *profile* specifies a relative pathway, the current directory is searched first, then "/*usr*/*gips*/*tables*/". The contents of any previously-read header are lost. *Ihread* returns either the stream-buffer pointer to *name*, or **NULL**, (zero in Fortran), signifying an error while reading.

Ihmerg adds information to that gathered in a previous call to *ihread* by reading a secondary stream. If *name* is the string ("-"), input is taken from the standard input stream *stdin*. If *name* specifies a relative pathway, the current directory is searched first, then "/*usr*/*gips*/*tables*/".

Ihwrit writes an existing General Image header that has been previously input via a successful call to *ihread*. The maximum output line length may be specified via the *length* parameter: at least one item will be written on each line. When *length* is zero, negative, or larger than some implementation limit, the default of 72 characters per line will be used. The output is to the file *name*, unless the latter is the string ("-"), when the standard output stream *stdout* is used. *Ihwrit* returns either the stream-buffer pointer to *name*, or **NULL**, (zero in Fortran), signifying an error while writing.

After successful completion, *ihread* and *ihmerg* leave the input stream positioned such that the next byte to be read is the first data pixel. *Ihwrit* flushes its output buffer so that the next item written should be the first data pixel.

If the *name* parameter of *ihread*, *ihmerg*, or *ihwrit* begins with a "|" character, the remainder of the parameter string is interpreted as UNIX command-shell input. It is passed to "/*bin*/*sh*" and the standard input or output of that sub-shell is assumed to be a legal general image header.

Ihget and *ihput* are used to access header values after a successful call to *ihread*. For both functions, *item* is the name of a header item, and *value* is a pointer to a variable of the appropriate type located within the caller's program. When called from C, *ihget* needs to know the maximum length of character array that may be returned, specified by *lvalue*. *Lvalue* should be omitted when calling *ihget* from Fortran.

Ihdel is used to remove an item from the in-core header, so that it will not be written out by a subsequent *ihwrit* call. However, the definition is not discarded, so that the variable can be "put back" with a later call to *ihmerg* or *ihput*.

Allowed variable names and their data types are defined by entries in the file specified in the *profile* variable of the prior *ihread* call. Each variable is described by the following three fields, separated by blanks and/or tabs, ending with a newline character:

name The name of the variable. It may contain any number of alphameric characters provided the total line length does not exceed 200 bytes.

type A single character code describing the variable:

- **R** A required character-string variable which *must* be present in every header and whose value *must* be identical to the description field that follows this entry in the *profile* file. Such variables identify headers and software version numbers.
- **V** A required character-string variable which *must* be present in every header and whose value *must* be identical to one of the (space-delimited) sub-fields in the description field that follows this entry in the *profile* file. Such variables identify hardware formats.
- **h** A character-string *history* variable that may be *added* more than once to the header. Each addition will be prefixed by a character string representation of the current date and time. Only the most recent added *history* variable may be removed by *ihdel*().
- **s** A character string variable.
- **c** A single character variable.
- **i** An integer variable.
- **d** A double-precision floating-point variable.
- **E** An End-of-header delimiting variable, which is never given an actual value but merely serves to identify the end of the header.

description One or more words of description, usually serving merely to document the variable but, for data-type **R**, actually defining the value that that string variable must have when encountered in a header.

Within the header, each variable is represented by an ascii string "*name* = *value*", where the three subfields may be separated by any number of whitespace characters. If the *value* of a string-variable contains whitespace characters, the *external* value must be enclosed within double quotes, and imbedded double quotes must be preceded by a back-slash. *ihread* will remove these (and only these) back-slashes when reading the header, and *ihwrit* will insert them when writing it out again, so you do not therefore have to pay any special attention to the presence of double quotes in string values passed to *ihput* or returned from *ihget*.

Ihtype returns the type and description fields of a particular header item: it fills *value* with the type character (as shown above), followed by the description string. The maximum length of *value* is specified by *lvalue*. If *lvalue* is set to 0 or 1, only the **type** character will be returned. *Lvalue* must be omitted entirely when calling *ihtype* from Fortran.

Once a GIPS header has been read or written, rasters (i.e. arrays of pixels corresponding to the same y-value and increasing x-values) may be read or written via calls to *ihbin* and *ihbout*, respectively. The second *(file)* parameter must be taken from the value returned by the last (successful) call to *ihread* or *ihwrit*. The *length* parameter specifies the length of the buffer, in bytes. If the actual length of the raster, as defined by the header variables *xnum* and *dtype*, is greater than *length*, *ihbin* truncates the input raster and *ihbout* fills the remainder of the output raster with pixels given the *dnull* value. Both routines return the number of bytes *actually* read or written. *ihbin* returns 0 on an end-of-file. If any other error occurs, these routines return the *negative* of the GIPS error code (see below).

The *ihbcl* routine closes the GIPS file opened by a call to *ihread, ihmerg*, or *ihwrit*. Thus, the following C and Fortran examples are equivalent:

```
#include "/usr/gips/include/gips.h"
FILE     *f, *ihread();
char     buf[512];
f = ihread("my.image","cprofile");
if (ihbin(buf,f,512) < 0) gipserr();
ihbcl(f);
```

```
integer   i
character*512 buf
i = ihread("my.image","cprofile")
if (ihbin(buf,i,512).lt.0) call iherrm
call ihbcl(i)
```

For Fortran programmers, here is a more lengthy example of a program that copies General Image file "gi.file1" to "gi.file2", adding a simple history entry to the header:

```
      character*4097 buf
      data n /0/

      kin = ihread("gi.test","cprofile")
      if (kin.eq.0) stop 1
      call ihput("history","Copied to ""gi.test2""")
      kout = ihwrit("gi.test2",0)
      if (kout.eq.0) goto 30

10          in = ihbin(buf,kin,4097)
            if (in.le.0.or.in.eq.4097) goto 20
            n = n + 1
            iout = ihbout(buf,kout,4097)
            if (iout.gt.0) goto 10

20    if (in.lt.0.or.iout.lt.0) call iherrm
      if (in.gt.4096) write(0,'(" raster GT 4096 bytes")')

      write (0,'(" copied ",i5," rasters")') n
      call ihbcl(kout)
30    call ihbcl(kin)
      stop
      end
```

FILES

/usr/gips/lib/libGI.a	library containing GIPS routines.
/usr/gips/include/gips.h	error codes, etc

AUTHOR

Peter G. Ford, MIT CSR

SEE ALSO

gcell(3), gips(3)

DIAGNOSTICS

Ihread, *ihmerg*, and *ihwrit* return **NULL** after detecting an error. The remaining routines return integer error codes. All routines save their return code in the external variable **gipserrno**, and C-language routines that include the file "/*usr*/*gips*/*include*/*gips.h*" may generate an error message via the statement **gipserr().** (From Fortran, execute "call iherrm"). The possible non-zero **gipserrno** values and associated error messages are as follows:

1	**GIPSBADARG**	bad GIPS routine argument
2	**GIPSNOHEAD**	no GIPS header in core
3	**GIPSBADOPEN**	open error: *file*
4	**GIPSBADREAD**	read error: *file*
5	**GIPSBADDATA**	bad data read: *file*
6	**GIPSNOVAR**	missing header variable: *item*
7	**GIPSNOVAL**	no value for header item: *item*
8	**GIPSNOEOF**	no end-of-header indicator
9	**GIPSBADVAR**	bad header entry: *item*
10	**GIPSBADVAL**	bad header item value: *item*
11	**GIPSBADTYPE**	bad header variable type: *item*
12	**GIPSTRUNC**	*item* value truncated
13	**GIPSBADLIM**	inconsistent *item* value
14	**GIPSUNSCALE**	unsupported *item* value
15	**GIPSNOSTR**	null *item* value
16	**GIPSNOCORE**	insufficient core
17	**GIPSBADWRITE**	write error: *file*

When a GIPS routine encounters an error, it stores the appropriate error code in *gipserrno*; the corresponding error messages are available in the external variable *gipserrs[gipserrno]*. If the message requires an inserted value (denoted above by *italics*), the insert is saved in *gipserrs[0]* and a *%s* placeholder is included in *gipserrs[gipserrno]*. The error message may therefore be generated within a "C" program by the function call

fprintf(stderr,gipserrs[gipserrno],gipserrs[0]);

Normally, a GIPS routine would wish to abort at this point, and this can most conveniently be achieved by using the *gipserr()* macro, which calls the *error()* routine in *-lGI*. If it is necessary to perform some cleanup task before exiting, *error* can be told to execute a user-specified routine by means of a prior call to *gipsint(proc)*, whose argument *proc* is the name of your cleanup routine. If, at some later stage of your program, this routine is no longer required, you must invoke *gipsint(0)* to signal that fact.