JPL Publication 88-32, Rev. 1, Vol. III

548

# Concurrent Image Processing Executive (CIPE)

## Volume III: User's Guide

Mih-seh Kong

```
(NASA-CR-186848)  CONCURRENT IMAGE                    N90-28550
PROCESSING EXECUTIVE (CIPE). VOLUME 3:
USER'S GUIDE Final Report  (JPL)  84 p
                                       CSCL 09B       Unclas
                                               G3/61  0293532
```

March 15, 1990

NASA
National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

TECHNICAL REPORT STANDARD TITLE PAGE

| | | |
|---|---|---|
| 1. Report No. JPL Pub. 88-32 | 2. Government Accession No. | 3. Recipient's Catalog No. |
| 4. Title and Subtitle<br>CONCURRENT IMAGE PROCESSING Executive (CIPE) | | 5. Report Date<br>November 1988 |
| | | 6. Performing Organization Code |
| 7. Author(s) Meemong Lee, Gregory T. Cooper, Steven L. Groom, Alan S. Mazer, Winifred I. Williams | | 8. Performing Organization Report No. |
| 9. Performing Organization Name and Address<br>JET PROPULSION LABORATORY<br>California Institute of Technology<br>4800 Oak Grove Drive<br>Pasadena, California 91109 | | 10. Work Unit No. |
| | | 11. Contract or Grant No.<br>NAS7-918 |
| | | 13. Type of Report and Period Covered<br>Final Report |
| 12. Sponsoring Agency Name and Address<br>NATIONAL AERONAUTICS AND SPACE ADMINISTRATION<br>Washington, D.C. 20546 | | 14. Sponsoring Agency Code<br>RE150 BP-889-20-41-17-62 |
| 15. Supplementary Notes | | |

16. Abstract

This report describes the design and implementation of a Concurrent Image Processing Executive (CIPE), which is intended to become the support system software for a prototype high performance science analysis workstation. The target machine for this software is a JPL/Caltech Mark IIIfp Hypercube hosted by either a MASSCOMP 5600 or a Sun-3, Sun-4 workstation; However, the design will accommodate other concurrent machines of similar architecture, i.e., local memory, multiple-instruction-multiple-data (MIMD) machines. The CIPE system provides both a multimode user interface and an applications programmer interface, and has been designed around four loosely coupled modules: (1) user interface, (2) host-resident executive, (3) hypercube-resident executive, and (4) application functions. The loose coupling between modules allows modification of a particular module without significantly affecting the other modules in the system. In order to enhance hypercube memory utilization and to allow expansion of image processing capabilities, a specialized program management method, incremental loading, was devised. To minimize data transfer between host and hypercube, a data management method which distributes, redistributes, and tracks data set information was implemented. The data management also allows data sharing among application programs. The CIPE software architecture provides a flexible environment for scientific analysis of complex remote sensing image data, such as imaging spectrometry, utilizing state-of-the-art concurrent computation capabilities.

| | | | |
|---|---|---|---|
| 17. Key Words (Selected by Author(s))<br>Computer Programming and Software | | 18. Distribution Statement<br>Unclassified - Unlimited | |
| 19. Security Classif. (of this report)<br>Unclassified | 20. Security Classif. (of this page)<br>Unclassified | 21. No. of Pages<br>107 | 22. Price |

JPL 0184 R 9/83

JPL Publication 88-32, Rev. 1, Vol. III

# Concurrent Image Processing Executive (CIPE)

## Volume III: User's Guide

Mih-seh Kong

March 15, 1990

NASA

National Aeronautics and
Space Administration

**Jet Propulsion Laboratory**
California Institute of Technology
Pasadena, California

## ABSTRACT

CIPE (the Concurrent Image Processing Executive) is both an executive which organizes the parameter inputs for hypercube applications and an environment which provides temporary data workspace and simple real-time function definition facilities for image analysis. CIPE provides two types of user interface. The Command Line Interface (CLI) provides a simple command-driven environment allowing interactive function definition and evaluation of algebraic expressions. The menu interface employs a hierachical screen-oriented menu system where the user is led through a menu tree to any specific application and then given a formatted panel screen for parameter entry.

This document describes how to initialize the system through the 'setup' function, how to read data into CIPE symbols, how to manipulate and display data through the use of 'executive functions', and how to run an application in either user interface mode.

## Table of Contents

## List of Figures

## List of Tables

# INTRODUCTION

CIPE (the Concurrent Image Processing Executive) is both an executive which organizes the parameter inputs for hypercube applications, and an environment which provides temporary data workspace and simple real-time function definition facilities. In the more function-rich command mode, all of these capabilities are available to the user. In the menu mode, the concentration is on the executive with the goal being to guide the user in the definition of parameters for specific applications.

The command mode uses a simple line-at-a-time, command-driven interface. In the menu mode, the user is led through a menu tree to specific applications and then given a formatted panel screen to fill out. Navigation and help are accessed through function keys.

The main purpose of CIPE is to make hypercube processing available in a straightforward way. It was assumed that the concurrent processor is connected to a host computer system which includes the usual broad range of operating system services, file I/O, display device hardware, network connections, etc. In order to develop an image processing executive for a virtual concurrent system environment, system setup procedures, data management schemes among multiple systems, and concurrent system interface methods were implemented. A set of 'executive functions' is provided by CIPE to setup the system configuration and to manipulate data. All image processing applications being developed for the hypercube will also be made part of the CIPE environment.

Due to the data-heavy characteristic of image processing, CIPE minimizes data I/O by keeping data resident and by loading image processing functions incrementally at run time. This is especially crucial with the hypercube since there is a serious data transfer bottleneck between host and hypercube. Image processing functions are managed in CIPE through a function dictionary file which keeps a record of all available application programs. When a user requests a function, CIPE searches for the requested function name in the function table and reads its executable module into the system. A user can also activate his own executable module by specifying the module name. Datasets are managed in CIPE via a symbol table. Each dataset is represented by a CIPE 'symbol' which is a data structure containing the size, datatype, physical data location, and data distribution map if the data is in the hypercube.

This User's Guide is composed of the following five main sections:

*Command Mode* -- A description on maneuvering through the Command Mode; what the terms mean and how the pieces interact.

*Menu Mode* -- How to use the Menu Mode; how the menus are organized and how parameters are fed to applications.

*System Setup* -- A description on setting up the computing system configuration.

*Executive Functions* -- Functions for data manipulation and image display.

*Applications* -- Current list of application functions available through CIPE.

# 1. COMMAND MODE

It is not necessary to understand the workings of the system internals described here to be able to use the system. If you simply want to run a couple of applications, you can skip this section and turn directly to section 2 -- Menu Mode.

When you first start CIPE (by typing 'cipe' at the Unix prompt) it responds with an initialization statement, followed by the CIPE prompt: '>'. For a list of the built-in commands see Table I. For a list of the applications functions and their arguments, see section 5. At the CIPE prompt, if you wish to see a list of the CLI (Command Line Interface) commands and applications functions type 'help'. To obtain help on an individual command or function, type 'help command/function_name. The CLI also allows the user to escape CIPE execution temporarily and run such operating system utilities as editing a file.

CIPE has four main constructs that the user can manipulate:

*SCRIPT* -- a grouping of commands (functions and simple control statements) typed in by the user and used to perform higher level tasks (e.g. image filtering combined with background subtraction and display as one function invocation). Scripts can make use of the 'DEFINE' command to allow argument substitution. In this way scripts are frequently referred to as 'functions' or 'macros'. They may be thought of as 'user defined functions' as opposed to CIPE functions. Scripts also resemble user-defined procedures by allowing commands or functions to be performed repetitively through looping and other conditional statements. An example of a macro file is given in Table II to illustrate how script can be a powerful way of utilizing the CLI mode features such as nested function calls (pipe), nested macro calls, and nested control statments. The macro **lp3-512** in Table II merges two 512 by 512 images while eliminating the seam line on the boundary by using the pyramid image processing technique. Two basic pyramid tools 'reduce' and 'expand; and a subroutine 'concat', (which averages the boundary pixels between two concatenated images) are called repeatedly by the macro.

*WORKSPACE* -- a script (i.e. group of DEFINEd commands of functions) stored on a permanent disk file which can be reloaded or edited. LOADing a workspace causes each of the lines in the file named to be executed. Any DEFINEd functions that have been typed in by the user (or by a previous LOAD for that matter) can be SAVEd to a file. These files can be EDITed at any time (EDIT invokes the vi editor). Files created by EDIT can be LOADed and are not restricted to DEFINEd functions, as is the case for SAVEd files.

*VARIABLE* -- all data accessed by CIPE functions must be stored internally in temporary working space and referred to through variable names. A variable may refer to an integer constant, a three dimensional array of floating point numbers or anything in between. The allowed data types are 8-, 16-, and 32-bit integers (C datatypes unsigned char, short int, and int) and 32-bit floating point (C datatype float). Variables can represent single numbers (e.g. a=6), or 1-, 2-, or 3-dimensional arrays. Variables can be grouped into expressions using arithmetic and logical operators (e.g. b*c+7 or a>c). Such constructs are particularly useful to allow looping and checking inside DEFINEd functions.

## Table I: List of CIPE commands

| syntax | description |
|---|---|
| DEFINE function (formal_args) command_list END | define a function as a collection of commands or functions |
| EDIT workspace | edit a workspace using vi screen editor |
| FOR variable = expr TO expr command_list END | performs same set of commands or functions multiple times |
| FOR variable = expr TO expr STEP expr command_list END | same as last command |
| HELP command/function_name | display a list of CLI commands and existing application functions including user-defined functions and scripts |
| IF expr THEN cmd_list ELSE IF ELSE IF expr THEN cmd_list... ELSE cmd_list ENDIF | |
| LOAD workspace | retrieve a workspace from the disk and execute the commands or functions in the workspace |
| PRINT expr_list | display the specified value(s). Multiple values should be separated with commas. |
| QUIT | exit CIPE section |
| READ variable FROM "filename" | read data from disk file to variable |
| SAVE workspace | save all current user-defined functions |
| SET attribute TO value | specify attribute, existing attributes are: 'coprocessor', 'cube dimension', 'display device id' and 'debug level' |
| SHOW func_name | print the user-defined function in a pretty format |
| SYMBOLS | show a list of all existing symbols |
| TRACES | show the traceable system parameters and their trace status |
| TURN boolean attribute | boolean is either 'on; or 'off', boolean attributes are: 'mouse', 'logging', 'appl trace' |
| WHILE value commands END | executes a set of commands or functions in a WHILE loop |
| WRITE value TO filename | write data to a disk file |
| output_variable = expr | assign the value of expr to output_variable |
| function (expr) | execute a function |
| !unixcommand | perform the specified unix command |

Table II : An Example on Macro File

```
define cat512(x,y)
   cat512=concat(copy(x,{1,1,512,256}),copy(y,{1,257,512,256}),1,1)
   end

define cat256(x,y)
   cat256=concat(copy(x,{1,1,256,128}),copy(y,{1,129,256,128}),1,1)
   end

define cat128(x,y)
   cat128=concat(copy(x,{1,1,128,64}),copy(y,{1,65,128,64}),1,1)
   end

define cat64(x,y)
   cat64=concat(copy(x,{1,1,64,32}),copy(y,{1,33,64,32}),1,1)
   end

define lp(a,ar)
   lp=sub(a,expand(ar,1))
   end

define lp3_512(x,y)
   xr1=reduce(x,1)
   xr2=reduce(xr1,1)
   xr3=reduce(xr2,1)
   yr1=reduce(y,1)
   yr2=reduce(yr1,1)
   yr3=reduce(yr2,1)
   lpx0=lp(x,xr1)
   lpx1=lp(xr1,xr2)
   lpx2=lp(xr2,xr3)
   lpy0=lp(y,yr1)
   lpy1=lp(yr1,yr2)
   lpy2=lp(yr2,yr3)
   lpxy0=cat512(lpx0,lpy0)
   lpxy1=cat256(lpx1,lpy1)
   lpxy2=cat128(lpx2,lpy2)
   xy3=cat64(xr3,yr3)
   txy2=expand(xy3,1)
   xy2=add(txy2,lpxy2)
   txy1=expand(xy2,1)
   xy1=add(txy1,lpxy1)
   txy0 = expand(xy1,1)
   lp3_512 = add(txy0,lpxy0)
   end
```

Variable references are 'symbols' in the CIPE syntax. A user may create symbols by reading a file, by copying an existing symbol (B = A), by assigning a set of values (C = {2,2}), or by activating a function ( D = sqrt(A)). As a symbol is created, its complete set of data attributes are stored in the symbol table. A user may also delete a symbol (delete(A)) or overwrite a symbol.

ATTRIBUTE -- an internal CIPE flag. Attributes tell CIPE such things as which display device to use and whether to keep a log file or not. Attributes come in two types: ones that have values that may be SET to a value, and ones that are Boolean flags and can only be TURNed on or off.

At the command mode prompt, the user must enter either a built-in command or a function call (with arguments). These function calls can be to applications functions, to DEFINEd functions (i.e. SCRIPT described above), or to functions the user created.

The following shows a simple example of a CIPE CLI session. In this example, the user reads in a blurred image and a blurring kernel from disk files, restores the image with function 'ML', and displays the resultant image with the 'draw' function.

```
unix% cipe
CIPE Version 3.2

> set  display device id  to  0
> read  a  from  "/ufs/images/blur.img"
> read  k  from  "/ufs/images/kernel.img"
> b  =  ML (a, k, 4.0, 10)
> draw (b, {1,1})
```

## 2. MENU MODE

Menu mode is a relatively painless way to find and execute a specific application program. Its interface to the user is a set of screens, each consisting of three windows (see Figure 1). The top window (function window) contains the current menu, the bottom window (menu control window) has status and general usage information, and the large middle window (parameter window) will have the parameter fields to be filled out for the application and any messages that are sent out.

To invoke Menu mode from Command mode, enter the command 'menu' at the prompt. In the Main Menu (shown in Figure 1), each of the entries in a menu represents either a sub-menu or a function. The following two simple naming standards are adopted to make these two types of names easily distinguishable: 'all function names are in lower case except for acronyms', and 'the sub-menu names have a "+" suffix. Picking a sub-menu (either by moving the cursor to that item and typing a carriage return or by typing the number associated with the item) causes that menu to take the place of the current menu. Picking a function causes the parameter list (with blanks for the values to be filled in) to appear in the parameter window. Once the values have been entered, the application can be run.

Maneuvering around the menu tree is done by selecting sub-menus, popping up one level in the tree by typing ^P, or jumping up to the top of the tree with ^R. Help on menu items or function parameters is available through ^H.

Some of what look like functions in Menu mode are really groups of built-in CIPE functions (e.g. 'math' functions). However, this is transparent to the user. Also, because the menus are not intrinsic to CIPE but rather are convenient user aides, there is nothing to prevent applications or even whole menu structures from being duplicated at different places in the tree. Since there is no way to jump laterally across the tree this is sometimes advantageous. For this reason, the 'disp+' and 'symbol+' sub-menus appear in every application entry. In addition to the CIPE provided application functions, CIPE also allows the user to execute his own application functions (ref. 'cipedict' in section 3, 'System Setup').

The following is an example showing how to restore a blurred image using the restoration function provided by CIPE. In general, there are three steps in running an application program: (1) set up the environment such as the coprocessor used and its dimension, display device, trace level, etc. (2) read input data to symbols either from disk files or through menu inputs, (3) specify function name and fill in all the parameters needed. Step 1 only needs to be done once per CIPE session; 2 and 3 can be repeated as many times as necessary.

The example will use the 'ML' function to restore an image using the host computer. The blurred image and the kernel matrix are stored in the disk files 'blur.img' and 'kernel.img'. All image files in CIPE need to have a header file specifying the dimension and the datatype of the image. The associated header files for the two files used here are 'blur.hdr' and 'kernel.hdr'.

Once the user has the input image files and appropriate header files, he enters cipe and types 'menu' to get into Menu mode. At the main menu, he uses 'setup' to specify the processor to be used, the trace level, etc. (The detailed descriptions of each field in the 'setup' menu and the allocation of the display device will be described in section 3, 'System Setup'.) Next the 'restore+' sub-menu is selected (Figure 2), and the 'read' function in the 'symbol+' sub-menu is invoked to read the data into CIPE symbols as shown in Figure 3. In the example, data in 'blur.img' is read into the CIPE symbol 'a', with the parameters from the header file 'blur.hdr' automatically inserted as the initial definition of the image area (start_line, start_pixel, number_of_lines, number_of_samples).

Again in the 'restore+' menu, the 'ML' function is selected. The user simply fills out the list of parameters shown in Figure 4, and the program runs with the resultant image going to symbol 'b'.

After the execution of the program, if the user wishes to display the resultant image, he selects sub-menu 'disp+'. In the example, we display the restored image 'b' using 'draw' function (assuming that we have allocated and selected the display device at the beginning of the session). Figure 5 shows the list of parameters to be filled in the 'draw' function.

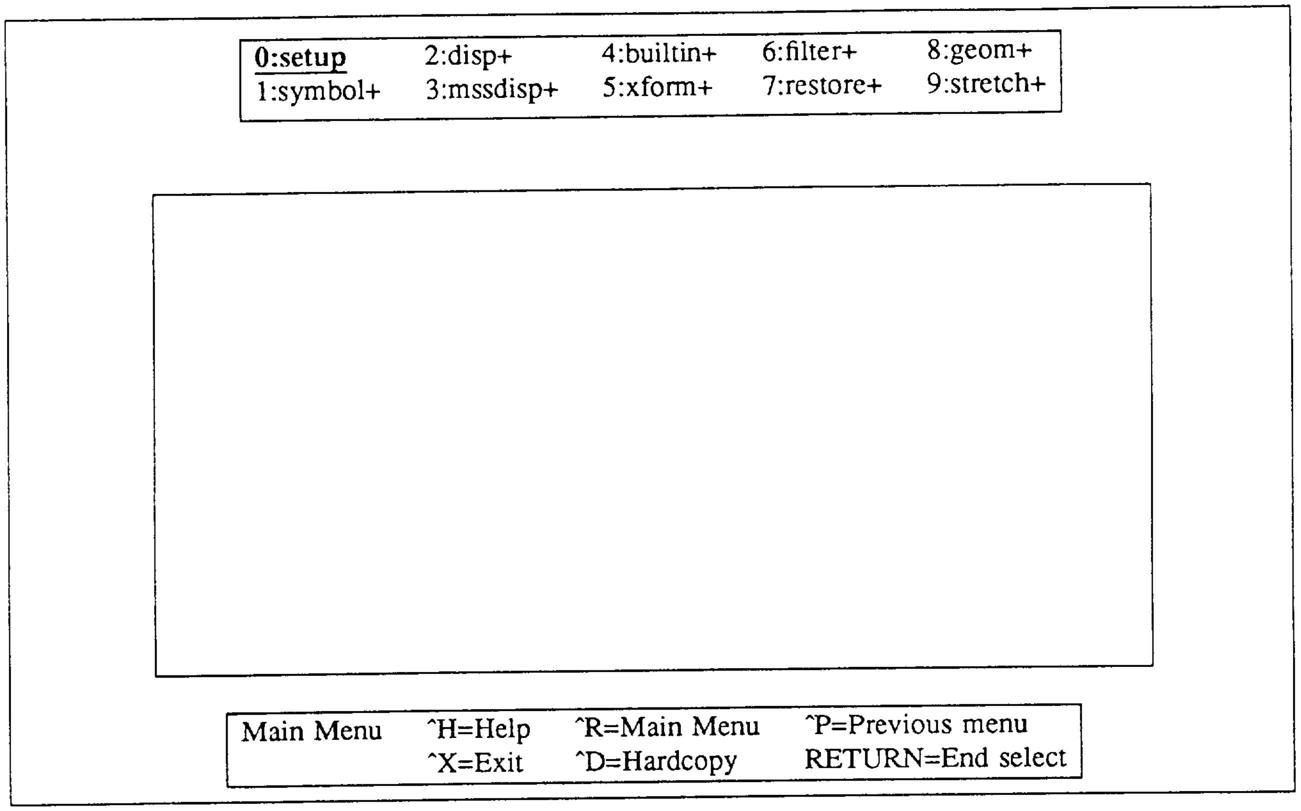


Figure 1: Main menu

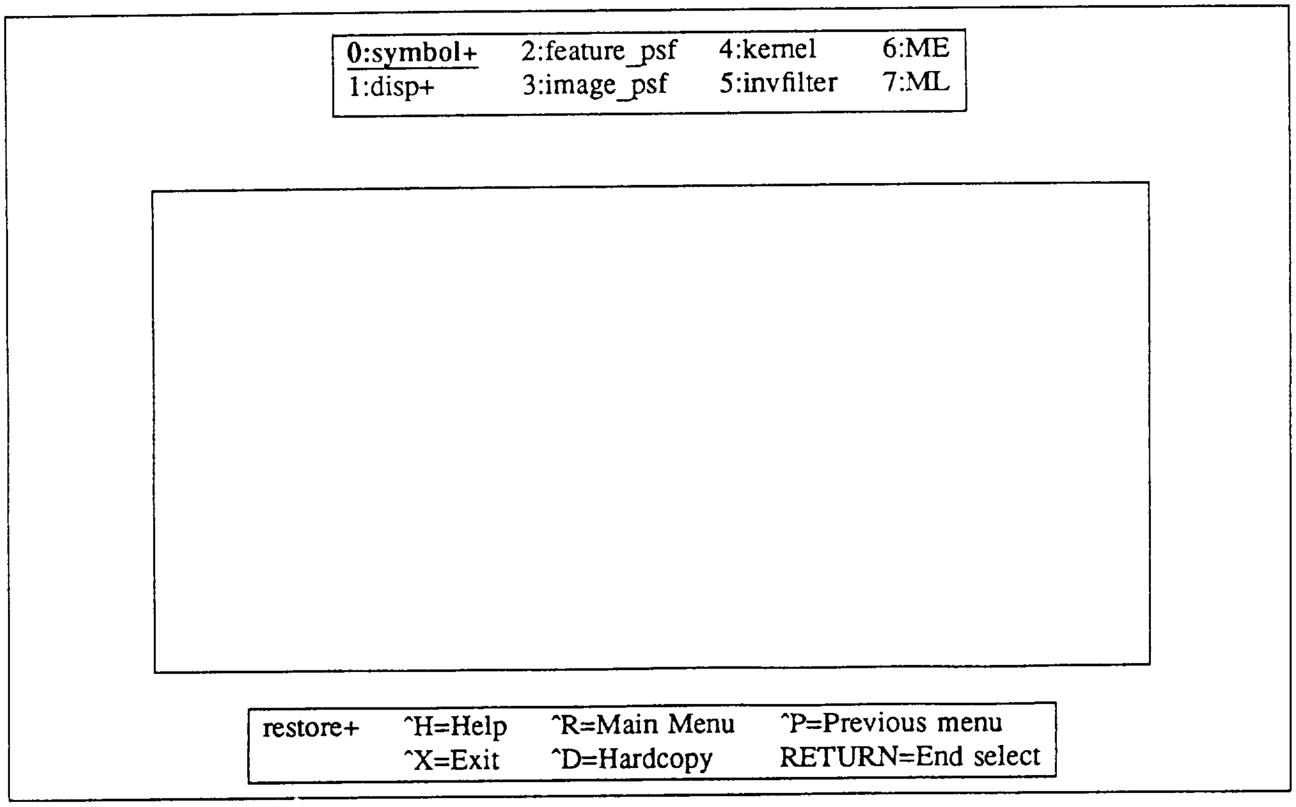
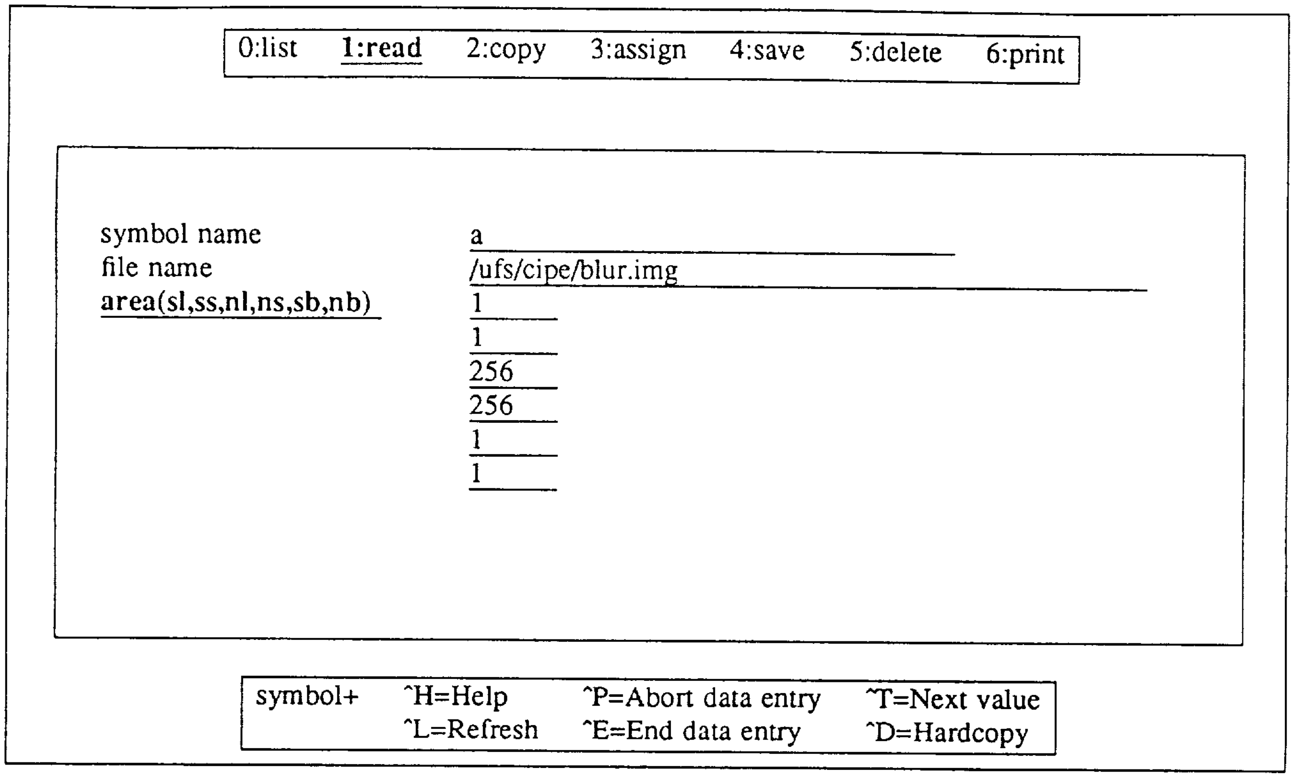
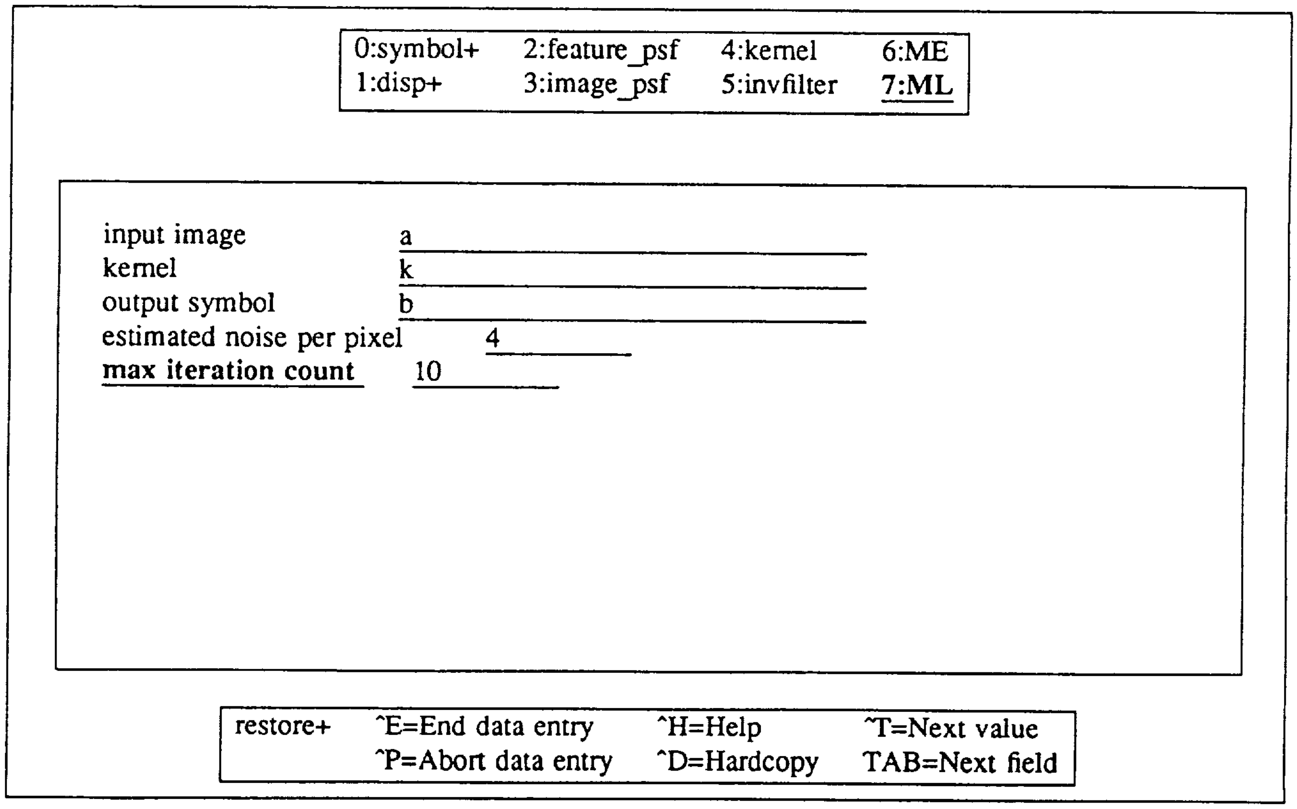


Figure 2: Restore submenu


0:list 1:read 2:copy 3:assign 4:save 5:delete 6:print
symbol+ ^H=Help ^P=Abort data entry ^T=Next value
^L=Refresh ^E=End data entry ^D=Hardcopy

```
symbol name              a
file name                /ufs/cipe/blur.img
area(sl,ss,nl,ns,sb,nb)  1
                         1
                         256
                         256
                         1
                         1
```

Figure 3: Read function in menu mode


0:symbol+ 2:feature_psf 4:kernel 6:ME
1:disp+ 3:image_psf 5:invfilter 7:ML
restore+ ^E=End data entry ^H=Help ^T=Next value
^P=Abort data entry ^D=Hardcopy TAB=Next field

```
input image                  a
kernel                       k
output symbol                b
estimated noise per pixel    4
max iteration count          10
```

Figure 4: Example for executing ML function in menu mode

## 3. SYSTEM SETUP

This section describes the functions that perform interactive system configuration. They include the allocation of the coprocessor along with the selection of various debugging and tracing levels, and the selection and allocation of the display device. In menu mode, these functions are performed through the use of the 'setup' function in the main menu and the functions under 'alloc+' submenu.

### setup function

In menu mode, a list of parameters is presented to the user (Figure 6) to be filled in. In CLI mode, however, it requires a system command to specify each parameter. The following is the list of parameters to be set. The exact syntax (shown in block letters) is required for the command to be accepted in the CLI mode.

#### *coprocessor*

This parameter enables the user to allocate and de-allocate a coprocessor.

> set **coprocessor** **to** *coprocessor_name*

or

> set **coprocessor** **to** *none*

Since the hypercube is a single user system, de-allocating it allows sharing of the cube among multiple users.

#### *cube dimension*

This parameter sets the hypercube dimensions. For a dimension of n, the hypercube provides $2^n$ nodes. This field will be shown on the screen if the coprocessor is specified.

> **set cube dimension to** n

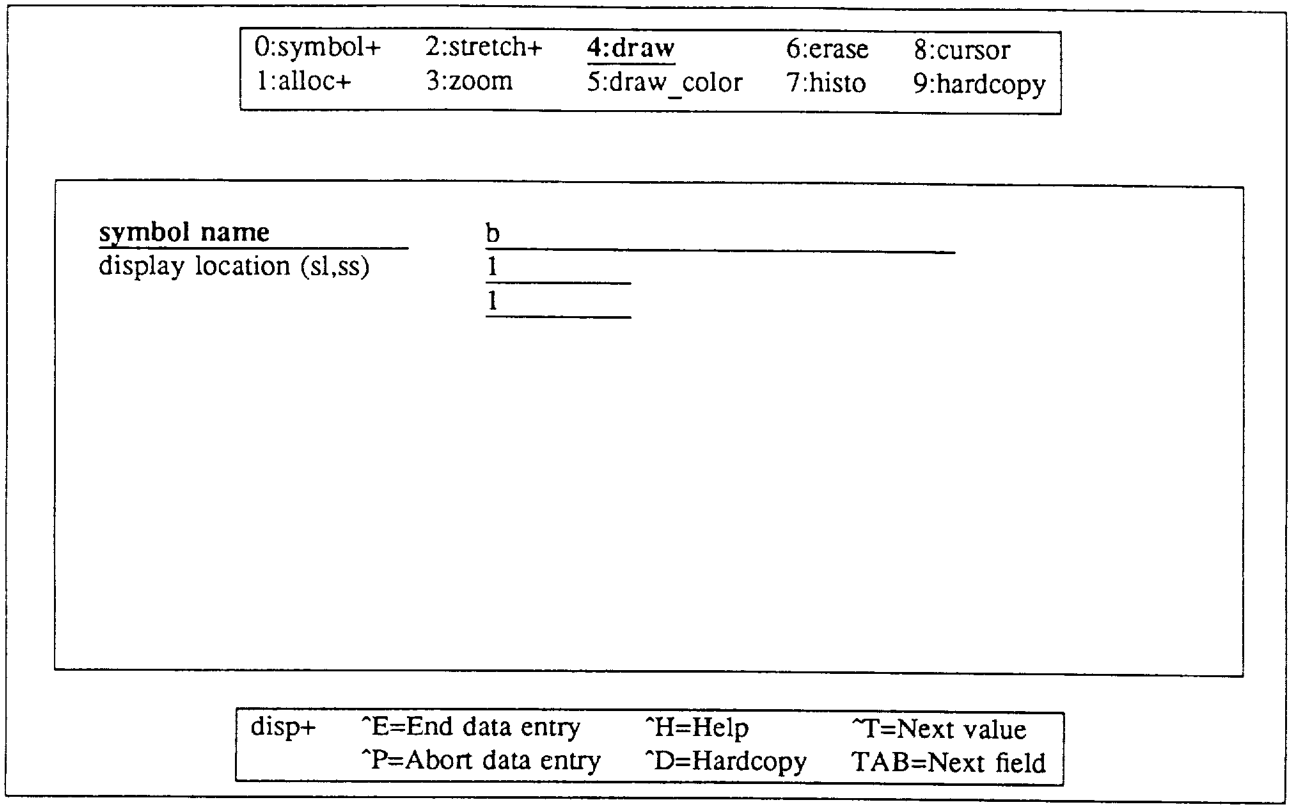
#### *logging*

This option allows the user to have a log file created which records what the user has done during the CIPE session. This file is created in the user's home directory under the name 'session.log.pid' where pid is the process identification number of the session.

> **turn on logging**

or

> **turn off logging**

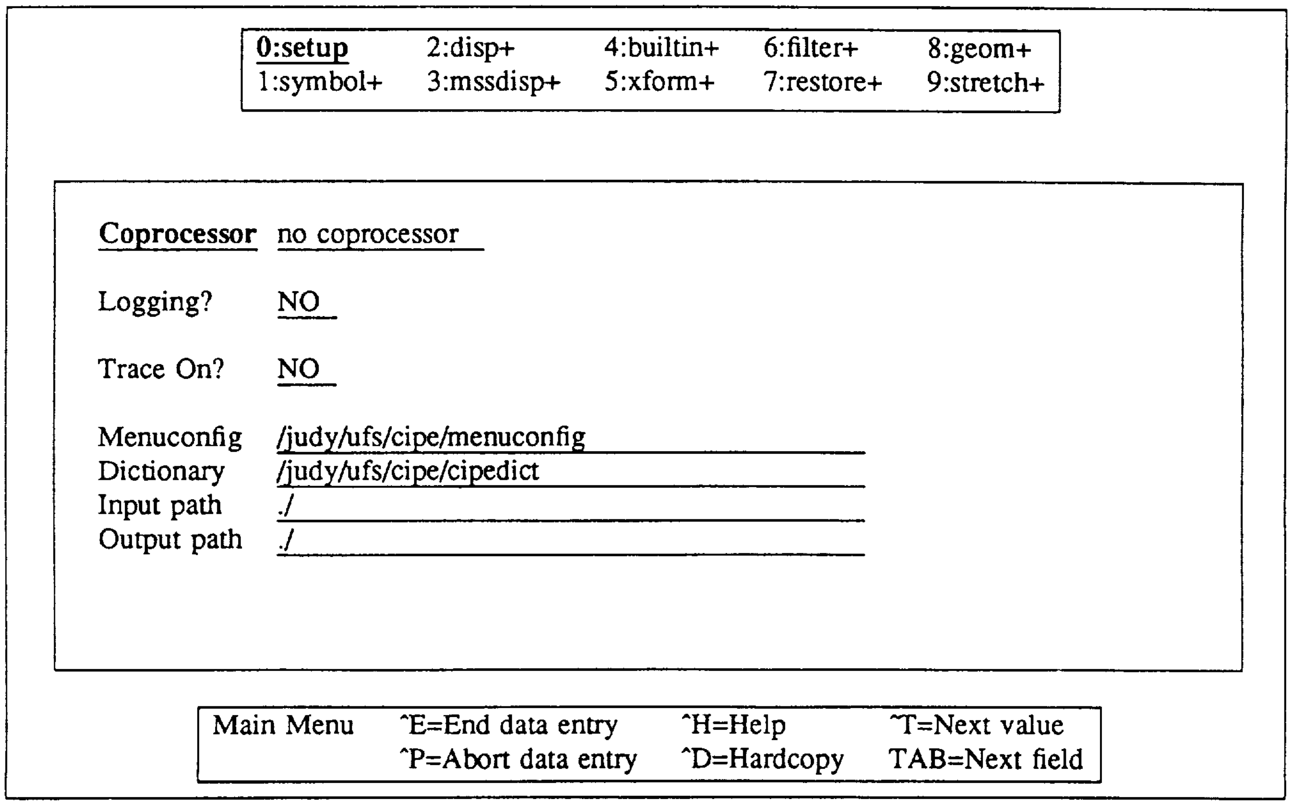


```
0:symbol+   2:stretch+   4:draw        6:erase   8:cursor
1:alloc+    3:zoom       5:draw_color  7:histo   9:hardcopy

symbol name                 b
display location (sl,ss)    1
                            1

disp+   ^E=End data entry     ^H=Help       ^T=Next value
        ^P=Abort data entry   ^D=Hardcopy   TAB=Next field
```

Figure 5: Example for displaying output symbol in menu mode



```
0:setup     2:disp+      4:builtin+   6:filter+    8:geom+
1:symbol+   3:mssdisp+   5:xform+     7:restore+   9:stretch+

Coprocessor  no coprocessor

Logging?     NO

Trace On?    NO

Menuconfig   /judy/ufs/cipe/menuconfig
Dictionary   /judy/ufs/cipe/cipedict
Input path   ./
Output path  ./

Main Menu   ^E=End data entry     ^H=Help       ^T=Next value
            ^P=Abort data entry   ^D=Hardcopy   TAB=Next field
```

Figure 6: Setup function in menu mode

*trace*

This trace turns on the debugging statements from the application program:

> \> **turn on appl_trace**

or

> \> **turn off appl_trace**

*debug level*

A set of parameters are provided which can be set for tracing the specific executions of functions. They include command line interpreter function related traces (**parse_trace, lex_trace, codegen_trace, functab_trace**), a symbol table manipulation related trace (**symtab_trace**), hypercube executive traces (**cube_command, cube_data, cube_symbol**), and a general CIPE execution trace (**exec_trace**). A parameter called 'debug level' is provided to control the details of the trace; however, these traces are meant for the programmer rather than a typical user.

> \> **turn on** name (e.g. executive) **trace**

or

> \> **turn off** name (e.g. executive) **trace**

In menu mode, the user may specify only appl_trace which selects the level of debugging messages provided in the application programs.

*menuconfig*

Figure 7 shows the entire menu tree as defined by the standard CIPE 'menuconfig' file (Appendix A). CIPE allows the user to activate a different menu tree by supplying an alternative menu configuration file in the menu mode.

*cipedict*

All CIPE application functions have their function names and path names defined in a file called 'cipedict' (Appendix B). When a user wishes to execute an application program he created, he has three options.

(1) The whole path_name and function_name can be specified. This is done by typing the path_name plus function_name in CLI mode or by using menu entry 'my_func' under 'bltin+' sub-menu in Menu mode.

(2) The new function can be added to the function list using the CIPE provided function 'add_func'. This can be done by typing add_func ("function_name", "path_name", "help_message") in CLI mode or by using Menu entry 'add_func' under 'bltin+' sub-menu.

(3) User creates his own 'cipedict' file and adds an entry for his function.

```
           ├ setup                          ├ list
           |                                ├ read
           |                                ├ copy
           |                                ├ assign
           |               ├ symbol+  -     ├ save
           |               |                ├ delete
           |               |                ├ print
           |               |
           |               ├ alloc+ -       ├ alloc
           |               |                ├ select
           |               |                ├ dealloc
           |               |                ├ disp_list
           ├ disp+  -      |
           |               ├ stretch+ -     ├ linear
           |               |                ├ table
           |               ├ zoom
           |               ├ draw
           |               ├ draw_color
           |               ├ erase
           |               ├ histo
           |               ├ cursor
           |               ├ hardcopy
           |
           |               ├ mssdraw
           ├ mssdisp+ -    ├ mssplot
           |               ├ erase
           |               ├ zoom
           |
           |               ├ add_func
           |               ├ myfunc
           |               ├ typecast
           |               ├ pattern
           ├ builtin+ -    ├ math
           |               ├ stat           ├ matop
           |               ├ matrix+ -      ├ cmatop
main+ -    |                                ├ constop
           |               ├ rfft2
           ├ xform+   -    ├ cfft2
           |               ├ powerspec
           |
           |               ├ kernel
           ├ filter+  -    ├ spfilter
           |               ├ freqfilter
           |               ├ medfilter
           |               ├ reseau
           |
           |               ├ kernel
           ├ restore+ -    ├ invfilter
           |               ├ ML
           |               ├ ME
           |               ├ feature_psf
           |               ├ image_psf
           |
           |               ├ rotate
           ├ geom+     -   ├ scale
           |               ├ surfit
           |               ├ tiept
           |               ├ gentie         ├ reduce
           |               ├ pyramid+ -     ├ expand
           |                                ├ concat
           ├ stretch+ -    ├ perc
```

Figure 7: Menu tree

*input path and output path*

These two parameters allow the user to specify the default directories for all input and output files. These path names can be overwritten at run time simply by specifying the whole absolute path name. In the CLI mode, the path names are specified by two separate commands as in the following examples:

```
   > set  input path   to  "/ufs/mydir/"
or
   > set  output path   to  "/ufs/mydir/"
```

**Alloc submenu**

There are four functions in the 'alloc+' submenu: 'alloc', 'disp_list', 'select', and 'dealloc'. 'alloc' and 'dealloc' are to allocate and de-allocate display unit. Since multiple device allocation is allowed, 'disp_list' function allows the user to display a list of allocated display units and 'select' function allows the user to select one of the units he allocated.

*alloc*

The 'alloc' function allocates the display device. The host name, the display device type, and the size of the display window are to be specified by the user as shown in the example in Figure 8. In this specific example, there are two ivas display units of size 1024x1024 (they are refered to as ivas 0 and 1). In addition to the two ivas, the user may create as many Sun windows as he pleases by specifying the window size; these windows will be assigned a unit number automatically by the system. In the CLI mode, the syntax for 'alloc' is just like any other function:

```
   > alloc (host_name, display_device_type, window_size)
```

*select*

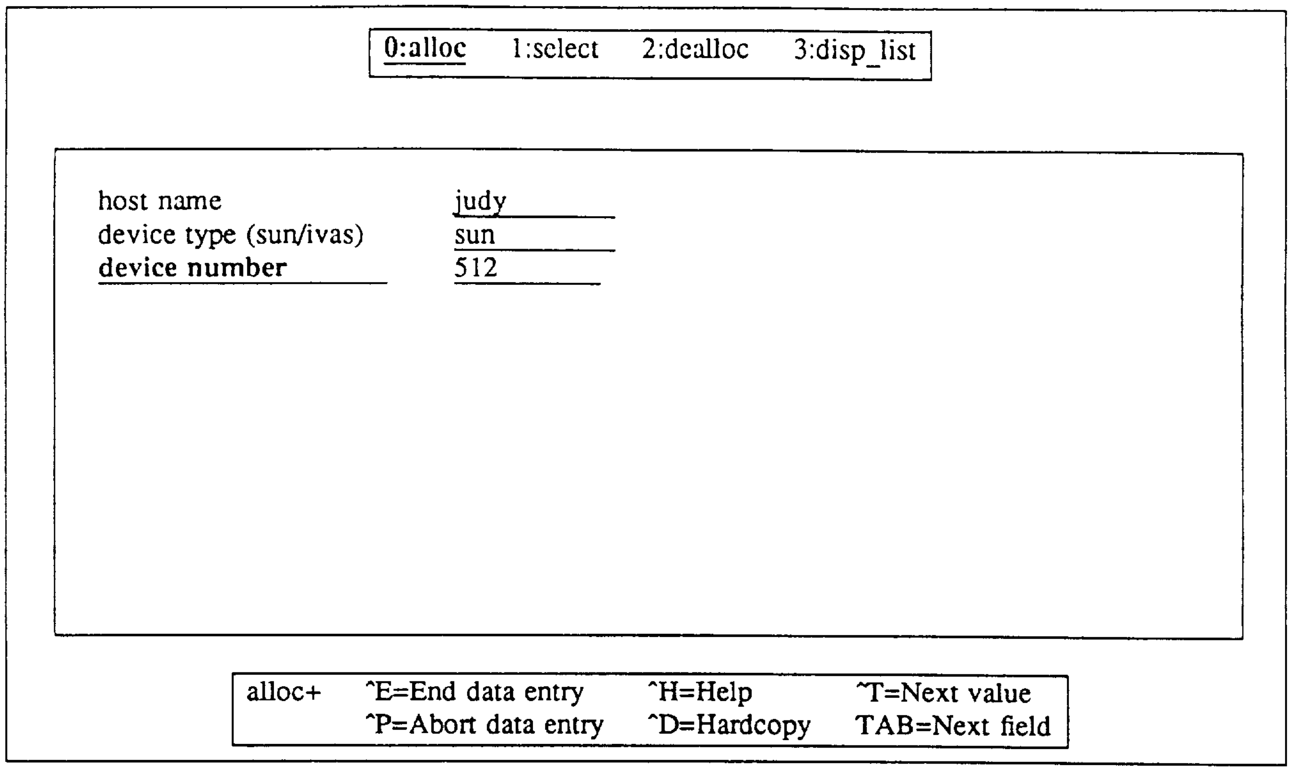
The 'select' function selects the display device to be used from all the ones the user has allocated. In menu mode, the combination of 'host_name /device_type /unit_number' is shown on the CIPE parameter window as shown in Figure 9; the user simply uses ^T to select the one he desires. In the CLI mode, the syntax is as follows:

```
   > select (display_unit_number)
```

*dealloc*

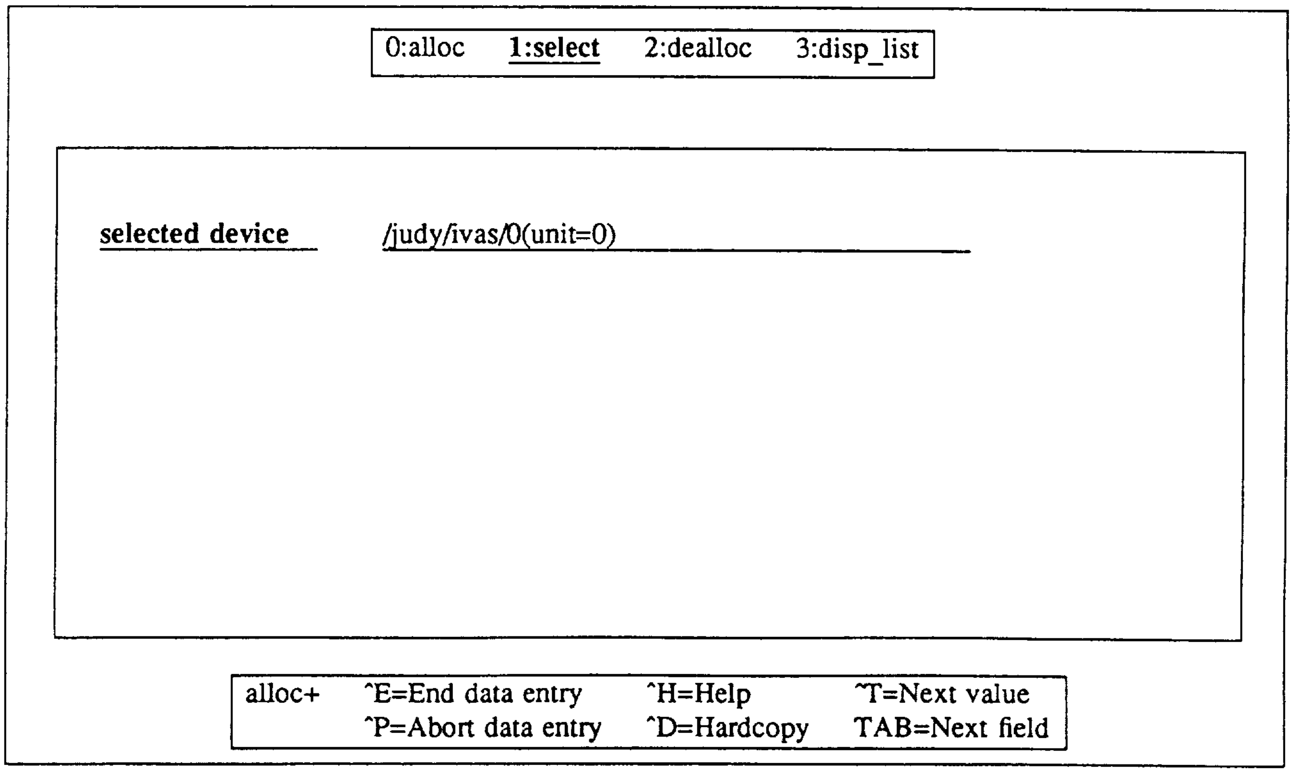
The 'dealloc' function de-allocates the display unit the user has selected.

```
   > dealloc
```



host name judy
device type (sun/ivas) sun
device number 512

Figure 8: Display device allocation in menu mode



selected device /judy/ivas/0(unit=0)

Figure 9: Select display unit in menu mode

*disp_list*

The 'disp_list' function prints a list of display units user has allocated. This function is particularly useful in the CLI mode because the user might not know the display unit numbers the system assigns to all the display windows.

```
> disp_list
```

# 4. EXECUTIVE FUNCTIONS

The executive functions include data management and image display functions. In menu mode, the data management functions are all in the 'symbol+' submenu, and the display functions are in the 'disp+' submenu. The individual functions are listed in alphabetical order as follows:

Data management

| | | |
|---|---|---|
| *assign* | -- | creates a CIPE symbol by assigning a value to it |
| *copy* | -- | copies data from one CIPE symbol to another |
| *delete* | -- | deletes a CIPE symbol |
| *list* | -- | lists all existing CIPE symbols |
| *print* | -- | prints out values of a CIPE symbol |
| *read* | -- | reads data from a disk file to a CIPE symbol |
| *save* | -- | saves a CIPE symbol to a disk file |

Image display

| | | |
|---|---|---|
| *alloc* | -- | allocates a display device |
| *cursor* | -- | reads the cursor position and displays the coordinate on the display screen |
| *dealloc* | -- | de-allocates a display device |
| *disp_list* | -- | prints a list of allocated display units |
| *draw* | -- | displays an image on the display screen |
| *draw_color* | -- | displays a color image on the display screen |
| *erase* | -- | erases the display screen |
| *hardcopy* | -- | makes a hard copy of the display screen |
| *histo* | -- | generates a histogram of the displayed image |
| *select* | -- | selects a display device previously allocated |
| *stretch* | -- | scales the intensity of a displayed image |
| *zoom* | -- | scales the size of a displayed image |

## Name

*add_func* -- dynamically add a function entry

## Synopsis

*add_func* ("function_name", "path_name", "help_message")

## Description

*add_func* adds an entry to a function table of CIPE; the added function can be accessed just like the CIPE application functions during the CIPE section.

## Menu mode path

Main --> builtin+ --> add_func

**Name**

*alloc* -- allocate a display unit

**Synopsis**

*alloc (host_name, display_device_type, window_size)*

**Description**

*alloc* allocates a display device. The host computer name is required for CIPE to activate the appropriate device driver and display software. In the CLI mode, the burdens of finding out the exact host name and the device type name are on the user. In menu mode, the available options are displayed on the screen.

**Menu mode path**

Main --> disp+ --> alloc+ --> alloc or

Application Function --> disp+ --> alloc+ --> alloc

## Name

*assign* -- create a CIPE symbol by assigning values to it directly

## Synopsis

*symbol_nam* = {*data*}

## Description

*assign* is similar to the mathematical assignment statement. In the CLI mode, this function assigns a single value or a small one dimensional array to a CIPE symbol. For higher dimensional data (such as an image) or a large data array, the *'read'* function should be used instead.

In menu mode, the user first specifies the symbol name and the dimension of the data array. A new menu page then appears with blank spaces for the individual data values.

## Menu mode path

Main --> symbol+ --> assign or

Application Function --> symbol+ --> assign

**Name**

*copy* -- copy all or partial data from one symbol to another

**Synopsis**

output = *copy* (input, {sl, ss, nl, ns}) for copying partial data,

output = input for copying the whole data array.

**Description**

*copy* duplicates all or part of the data from an input symbol to an output symbol. The area of the data to be copied is specified by {sl, ss, nl, ns} where

| | |
|---|---|
| sl : | start line |
| ss : | start sample |
| nl : | number of lines |
| ns : | number of samples |

If the input symbol is a vector rather than an image, sl and nl are still required, both of which should be 1.

**Menu mode path**

Main --> symbol+ --> copy or

Application function --> symbol+ --> copy

**Name**

*cursor* -- read the cursor position on the display screen

**Description**

This function is not available in the CLI mode; in menu mode, it reads the cursor position and prints the coordinates on the CIPE parameter window

**Menu mode path**

Main --> disp+ --> cursor or

Application function --> disp+ --> cursor

**Name**

*dealloc* -- de-allocate a selected display unit

**Synopsis**

*dealloc ()*

**Description**

*dealloc* de-allocates a selected display unit.

**Menu mode path**

Main --> disp+ --> alloc+ --> dealloc or

Application function --> disp+ --> alloc+ --> dealloc

**Name**

*delete* -- delete a symbol and free the data area

**Synopsis**

*delete* (symbol_name)

**Description**

*delete* deletes an existing symbol and frees the data area for other usage.

**Menu mode path**

Main --> symbol+ --> delete or

Application function --> symbol+ --> delete

**Name**

*disp_list* -- display a list of allocated display units

**Synopsis**

*disp_list ()*

**Description**

*disp_list* prints a list of allocated display windows, including for each window the host_name, device_type_name, window_size, and an associated unit number which the system assigned to each display window. (e.g. /judy/sun/256/unit=4)

**Menu mode path**

Main --> symbol+ --> disp_list or

Application function --> symbol+ --> disp_list

## Name

*draw* -- display an image on a display device

## Synopsis

*draw* (symbol_name, {start_line, start_sample})

## Description

*draw* displays an image on the selected display device, starting with pixel {start_line, start_sample}. Before using the 'draw' function, a display device must be allocated.

## Menu mode path

Main --> display+ --> draw or

Application function --> display+ --> draw

**Name**

*draw_color* -- display a color image on a display screen

**Synopsis**

*draw_color* (red_image, green_image, blue_image, {start_line, start_sample})

**Description**

*draw_color* displays a color image on a selected display unit, starting with pixel {start_line, start_sample}. Before using the 'draw_color' function, a color display device must be allocated.

**Menu mode path**

Main --> display+ --> draw_color or

Application function --> display+ --> draw_color

## Name

*erase* -- erase all or part of an image from a display screen

## Synopsis

*erase* ("i/o/a", {sx, sy, nx, ny})

## Description

*erase* erases the portion of display screen specified by coordinates {sl, ss, nl, ns} where

| | |
|---|---|
| sl : | start line |
| ss : | start sample |
| nl : | number of lines |
| ns : | number of samples |

i/o/a indicates image/overlay/all display frame buffers.

## Menu mode path

Main --> display+ --> erase or

Application function --> display+ --> erase

## Name

*list* -- show the list of all existing CIPE symbols user has created

## Synopsis

*symbols ()*

## Description

Typing '*symbols*' in CLI mode or choosing the '*list*' function in menu mode prints out a list of existing symbols

## Menu mode path

Main --> symbol+ --> list or

Application function --> symbol+ --> list

## Name

*mssdraw* -- display multi-spectral images

## Synopsis

*mssdraw* (image_symbol_name, band, {start_line, start_pixel})

## Description

*mssdraw* displays a multi-spectral image with specified band number.

## Menu mode path

Main --> mssdisp+ --> mssdraw

**Name**

*print* -- print the contents of a symbol

**Synopsis**

*print* (symbol_name, {sl, ss, nl, ns})

**Description**

In CLI mode, only one dimensional data can be printed out, while in menu mode, a two dimensional data area can be specified and printed out on the CIPE parameter window.

**Menu mode path**

Main --> symbol+ --> print or

Application function --> symbol+ --> print

## Name

*read* -- read data from a disk file to a CIPE symbol

## Synopsis

*read* symbol_name *from* "file_name"

## Description

The disk file to be read by the CIPE function must have a corresponding CIPE header file with the name 'filename.hdr'. CIPE header file contains the following:

```
CIPE
offset = 8
number of lines = 128
number of samples = 128
number of bands = 1
type = byte
```

## Menu mode path

Main --> symbol+ --> read or

Application function --> symbol+ --> read

**Name**

*save* -- write data from a CIPE symbol to a disk file

**Synopsis**

*save* (symbol_name, "file_name", {sl, ss, nl, ns})

**Description**

This function is the reverse of the '*read*' function. It allows all or part of the image to be saved to a disk file; the associated header file will also be created by CIPE with the file name "data_file_name.hdr".

**Menu mode path**

Main --> symbol+ --> save or

Application function --> symbol+ --> save

**Name**

*select* -- select a display unit previously allocated

**Synopsis**

*select* (device_unit_number)

**Description**

Since multiple devices might be allocated, user uses this function to select one of the allocated display windows for displaying the image

**Menu mode path**

Main --> disp+ --> alloc+ --> select or

Application function --> disp+ --> alloc+ --> select

**Name**

*stretch* -- perform intensity scaling on a displayed image

**Synopsis**

*stretch* (min, max)

**Description**

In CLI mode, '*stretch*' scales the intensity values of a displayed grey-scale image between the specified min and max values. In the Menu mode, both grey-scale and color images can be manipulated (the user specifies the number of colors to be scaled and the minimum and maximum intensities for each color). In either case, this function only affects the intensity values on the displayed screen and not the data.

**Menu mode path**

Main --> disp+ --> stretch or

Application function --> disp+ --> stretch

## Name

*zoom* -- zoom a displayed image

## Synopsis

*zoom* ("i/o/a", zoom_factor, {sl,ss})

## Description

*zoom* scales the size of an image on the selected display screen; it does not affect the data.

## Option

i/o/a where

| | |
|---|---|
| i : | image plane |
| o : | overlay plane |
| a : | all planes |

## Menu mode path

Main --> disp+ --> zoom or

Application function --> disp+ --> zoom

## 5. APPLICATIONS

The individual applications are listed here in alphabetical order. They fall into the following six general groups:

Filtering

| | | |
|---|---|---|
| *spfilter* | -- | convolves an image with a given kernel |
| *freqfilter* | -- | convolves an image with a given kernel in frequency domain |
| *medfilter* | -- | performs median filtering |
| *kernel* | -- | generates kernel matrix in spatial domain |
| *reseau* | -- | removes reseau marks from an image |

Fourier Transform

| | | |
|---|---|---|
| *rfft2* | -- | performs two dimensional FFT with a real input array |
| *cfft2* | -- | performs two dimensional FFT with a complex input array |
| *powerspec* | -- | calculates power spectrum of FFT results |
| *Power* | -- | takes FFT of a real image and displays its power spectrum |

Restoration

| | | |
|---|---|---|
| *ML* | -- | restores a blurred image using Maximum Likelihood restoration algorithm |
| *ME* | -- | restores a blurred image using Maximum Entropy restoration algorithm |
| *invfilter* | -- | restores a blurred image using recursive inverse convolution filtering method |
| *feature_psf* | -- | creates a kernel using user specified line segments |
| *image_psf* | -- | creates a kernel using user specified sub_image area |

Geom

|  |  |  |
|---|---|---|
| *rotate* | -- | rotates an image |
| *scale* | -- | scales the size of an image |
| *concat* | -- | concatenates two images |
| *tiept* | -- | resamples an image using a given tie point file |
| *surfit* | -- | resamples an image using a least squares fit through irregular tie points |
| *gentie* | -- | generates regular grid tie point file using a least squares fit through a set of irregular tie points |
| *reduce* | -- | performs Gaussian pyramid operation on an image |
| *expand* | -- | performs expand pyramid operation on an image |
| *merge* | -- | mosaic two images using 'reduce' and 'expand' pyramid operations |

Histogram Manipulations

|  |  |  |
|---|---|---|
| *perc* | -- | linear scaling of an image histogram |

Built-in Utilities

|  |  |  |
|---|---|---|
| *typecast* | -- | typecasting functions: char, int, and float |
| *math* | -- | basic mathematical functions: sqrt, log, log10, square, abs, and minus |
| *Matrix* | -- | basic matrix operations: addition, subtraction, multiplication, division, and the calculation of complex conjugate |
| *constop* | -- | basic matrix operation between a matrix and a constant |
| *stat* | -- | basic statistical functions: min, max, median, std, var, and mode |
| *pattern* | -- | generates images with simple patterns such as bar, checkered, etc. |

## Name

*cfft2* -- perform FFT with complex input variables

## Synopsis

(output_real, output_imaginary) = *cfft2* (input_real, input_imaginary, mode)

## Description

*cfft2* performs a two dimensional FFT on complex variable input data. Two input data arrays are required, for real and imaginary data respectively. Two output arrays are generated containing the real and imaginary parts of the result. (This function is not available in the CLI mode at this time.)

## Option

| | |
|---|---|
| mode = -1 : | forward FFT |
| mode = 1 : | inverse FFT |

## Menu mode path

Main --> xform+ --> cfft2

## Name

*cmatop* -- math functions involving two complex matrices: *cmp_add, cmp_sub, cmp_mult, cmp_div, cmpc_mult* -- add, subtract, multiply, divide and calculate the complex conjugate of two complex input symbols.

## Synopsis

**(out_real, out_imagi)** = *cmp_add* **(s1_real, s1_imagi, s2_real, s2_imagi)**
**(out_real, out_imagi)** = *cmp_sub* **(s1_real, s1_imagi, s2_real, s2_imagi)**
**(out_real, out_imagi)** = *cmp_mult* **(s1_real, s1_imagi, s2_real, s2_imagi)**
**(out_real, out_imagi)** = *cmp_div* **(s1_real, s1_imagi, s2_real, s2_imagi)**
**(out_real, out_imagi)** = *cmpc_mult* **(s1_real, s1_imagi, s2_real, s2_imagi)**

## Description

These five functions perform basic mathematical functions between two complex matrices. In the Menu mode, these functions are denoted by the operation symbols: '+', '-', '*', '/' for addition, subtraction, multiplication, division, and '#' for the calculation of the complex conjugate. (This function is not available in the CLI mode at this time.)

## Menu mode path

Main --> builtin+ --> matrix+ --> cmatop

**Name**

*concat* -- concatenate two images of compatible sizes together

**Synopsis**

output = *concat* (input1, input2, ivert, iave)

**Description**

*concat* concatenates two images of compatible sizes into a single image. There are two parameters to be specified in addition to the two input images required for concatenation. If two images are to be concatenated horizontally, the parameter 'ivert' is to be set to 0; otherwise ivert is 1. If 'iave' is set to 1, averaging of the boundary will be performed.

**Menu mode path**

Main --> geom+ --> concat

## Name

*constop* -- math functions involving a matrix and a constant: *cadd, csub, cmult, cdiv* -- add, subtract, multiply, and divide the content of an input symbol by a given constant.

## Synopsis

sout = *cadd* (s, c)
sout = *csub* (s, c)
sout = *cmult* (s, c)
sout = *cdiv* (s, c)

## Description

These four functions add, subtract, multiply or divide every element in the input symbol by a given constant.

## Menu mode path

Main --> builtin+ --> matrix+ --> constop

## Name

*expand* -- perform expand pyramid operation on an image

## Synopsis

output = *expand* (input, pyramid_level)

## Description

*expand* is one of the pyramid operations; it expands an image of N x N to 2N x 2N by interpolating sample values between the given pixels.

## Menu mode path

Main --> geom+ --> pyramid+ --> expand

## Name

*feature_psf* -- create a kernel using user specified features

## Synopsis

This function requires the interactive graphic inputs, it is not available in the CLI mode

## Description

*feature_psf* allows user to specify a feature by entering multiple line segments using an interactive graphic input device; these segments are saved and generated into a kernel for the purpose of deblurring an image.

## Menu mode path

Main --> restore+ --> feature_psf

## Name

*freqfilter* -- perform convolution / deconvolution in frequency domain

## Synopsis

output = *freqfilter* (input, kernel, mode)

## Description

*freqfilter* performs a convolution in the frequency domain using FFT; this is meant to be used when the kernel size is large (12 or larger).

## Option

|  |  |  |  |  |
|---|---|---|---|---|
| mode | = | 1 | : | inverse FFT |
| mode | = | -1 | : | forward FFT |

## Menu mode path

Main --> filter+ --> freqfilter

## Name

*gentie* -- generate a tie point file consists of regular grid tie points

## Synopsis

*gentie* ("input_tie_file", order_of_fit, {nptx, npty, gapx, gapy}, "output_tie_file")

## Description

*gentie* performs a least squares fit through a set of irregularly spaced tie points, and generates regular grid tie points based on the fitted coefficients. Besides the input and output tie point file names, the user may also specify the default parameters 'order_of_fit' and 'tie_parameter' ({nptx, npty, gapx, gapy}) where

| | |
|---|---|
| nptx : | number of tie points in the x direction |
| npty : | number of tie points in the y direction |
| gapx : | distance between two tie points in the x direction |
| gapy : | distance between two tie points in the y direction |

## Menu mode path

Main --> geom+ --> gentie

## Name

*image_psf* -- create a kernel using user specified sub_image area

## Synopsis

This function requires the interactive graphic inputs; it is not available in the CLI mode

## Description

*image_psf* allows user to specify an image by entering the coordinates of the upper left corner and the lower right corner using an interactive graphic input device. This sub_image area is made into a kernel for the purpose of deblurring an image.

## Menu mode path

Main --> restore+ --> image_psf

## Name

*invfilter* -- restore a blurred image using a constrained inverse filter algorithm

## Synopsis

output = *invfilter* (input, kernel, noise_level(float), number_of_iteration, {lamda(float), del_lamda(float)})

## Description

*invfilter* uses a constrained inverse FFT iteratively to restore a blurred image. The user needs to supply the blurring kernel, the estimated noise level of the input image, and the maximum number of iterations desired. The parameters 'lamda' (the Lagrange multiplier) and 'del_lamda' are optional if run in menu mode. Noise_level, lamda, and del_lamda are floating numbers.

## Menu mode path

Main --> restore+ --> invfilter

## Name

*kernel* -- generate a convolution kernel for image filtering or deblurring

## Synopsis

output = *kernel* ("psf_type", {operand1, <operand2>})

## Description

*kernel* generates the following types of convolution kernels:

| psf_type | psf_attr | resultant psf |
|---|---|---|
| "box" | {nl, ns} | a rectangular kernel containing ones |
| "gauss" | {sigmax, sigmay} | a rectangular kernel based on a Gaussian point spread function |
| "vector" | {magnitude, angle} | a rectangular kernel containing a vector |
| "circle" | {radius} | a square kernel based on a circular point spread function |

## Menu mode path

Main --> filter+ --> kernel

## Name

math operations involving single argument: *sqrt, log, log10, square, abs, negate* -- these are similar to the math functions in the C math library.

## Synopsis

output = *sqrt* (input)
output = *log* (input)
output = *log10* (input)
output = *square* (input)
output = *abs* (input)
output = *negate* (input)

## Description

All these basic math library functions require float datatype as input. If the input is an array, the operation is done on each element.

## Menu mode path

Main --> builtin+ --> math

**Name**

*matop* -- add, subtract, multiply, and divide the contents of two input symbols

**Synopsis**

output = *add* (s1, s2)
output = *sub* (s1, s2)
output = *mult* (s1, s2)
output = *div* (s1, s2)

**Description**

These four functions add, subtract, multiply or divide the contents of two input symbols. If the inputs are two images, the results are pixel by pixel operations. The two input symbols must be of the same dimension.

**Menu mode path**

Main --> builtin+ --> matrix+ --> matop or

Main --> restore+ --> matop

## Name

*ME* -- restore a blurred image using a maximum entropy algorithm

## Synopsis

output = *ME* (input, kernel, noise_level, d_lamda, number_of_iteration)

## Description

*ME* uses a maximum entropy algorithm to restore a blurred image. The user needs to supply the blurring kernel, estimated noise level of the input image, and the maximum number of iterations desired as well as the input image. User may also specify the step size d_lamda; however, the default value of 0.25 will be used if it is not specified. Both noise_level and d_lamda should be specified as floating numbers in the CLI mode to avoid misinterpretation.

## See also

*kernel* for the generation of various kernel types and sizes

## Menu mode path

Main --> restore+ -->ME

**Name**

*medfilter* -- perform median filtering over an image

**Synopsis**

output_image = *medfilter* (input_image, window_size, tolerance)

**Description**

*medfilter* performs median filtering with given window size.

**Menu mode path**

Main --> filter+ --> medfilter

## Name

*merge* -- mosaic two images using pyramid operations

## Synopsis

output = *merge* (input1, input2, sample1, sample2, pyramid_level)

## Description

*merge* utilizes pyramid operations 'reduce' and 'expand' to merge two images so that the boundary blends without an apparent seam line. The implementation allows the merge of two images to be along only a vertical line; also the length of each image has to be power of 2. Besides the two input images, the user needs to specify the vertical lines to be jointed on each image and the pyramid level to be used.

## Menu mode path

Main --> geom+ --> pyramid+ --> merge

## Name

*ML* -- restores a blurred image using maximum likelihood algorithm

## Synopsis

output_image = *ML* (input_image, kernel, noise_level, number_of_iteration)

## Description

*ML* uses maximum likelihood algorithm by Lucy and Richarson to restore a blurred image. User needs to supply the blurring kernel, estimated noise level (a floating point number) of the input image, and the maximum number of iterations desired as well as the input image.

## See also

*kernel* for the generation of various kernel types and sizes

## Menu mode path

Main --> restore+ -->ML

## Name

*pattern* -- generate image patterns

## Synopsis

output = *pattern* ("pattern_type", {image_width, image_length}, {min_inten, max_inten}, {pattern_width, pattern_length}/{variance})

## Description

*pattern* generates the following image patterns:

| pattern_type | pattern_attr | resultant pattern |
|---|---|---|
| "bar" | {width, length} | vertical bar stripes |
| "checker" | {width, length} | checkered pattern |
| "plane" | {width, length} | uniform intensity rectangle |
| "randu" | | uniform random noise |
| "randg" | {variance} | Gaussian random noise |

## Menu mode path

Main --> builtin+ --> pattern

## Name

*perc* -- scale an image histogram linearly according to the specified upper and lower percentage cutoff

## Synopsis

output = *perc_stretch* (input, lower_perc, upper_perc)

## Menu mode path

Main --> Stretch --> perc

**Name**

*Power* -- perform FFT on a real image and output its power spectrum

**Synopsis**

output = *Power* (input)

**Description**

*Power* performs FFT on a real array and calculates its power spectrum

**Menu mode path**

Main --> restore+ --> Power

**Name**

*powerspec* -- calculate the power spectrum of FFT results

**Synopsis**

output = *powerspec* (real_fft_result, imaginary_fft_result)

**Description**

*powerspec* calculates the power spectrum of a complex array

**Menu mode path**

Main --> xform+ --> powerspec

## Name

*reduce* -- perform Gaussian pyramid on an image

## Synopsis

output = *reduce* (input, pyramid_level)

## Description

*reduce* is one of the basic pyramid tools called 'Gaussian pyramid'. At each level, the image is blurred and sub-sampled; thus the resolution of the image is reduced in half at each level. Image sizes should be power of two for pyramid algorithms.

## Menu mode path

Main --> geom+ --> pyramid+ --> reduce

**Name**

*reseau* -- remove reseau marks from an image

**Synopsis**

output = *prep* (input, "reseau_location_filename")

**Description**

*reseau* uses correlation to detect the reseau marks from an image and removes them

**Menu mode path**

Main --> filter+ --> reseau

## Name

*rfft2* -- perform two dimensional FFT with a real input image

## Synopsis

(output_real, output_imaginary) = *rfft2* (input_real, mode)

## Description

*rfft2* performs a two dimensional FFT on a real input array; the output is a complex array.

## Option

mode = 1 : inverse FFT
mode = -1 : forward FFT

## Menu mode path

Main --> xform+ --> rfft2

## Name

*rotate* -- rotate an image n degrees counterclockwise

## Synopsis

output_image = *rotate* (input_image, angle_in _degree)

## Description

*rotate* rotates an image counterclockwise; the angle should be specified in degrees ( a floating point number).

## Menu mode path

Main --> geom+ --> rotate

## Name

*scale* -- scale the size of an image

## Synopsis

output_image = *scale* (input_image, x_scale_factor, y_scale_factor)

## Description

*scale* scales the size of an image; the scale_factors should be specified in floating point numbers.

## Menu mode path

Main --> geom+ --> scale

## Name

*spfilter* -- convolve an image with a kernel

## Synopsis

output_image = *spfilter* (input_image, kernel)

## Description

*spfilter* convolves an image with a given kernel in the spatial domain. This function is meant for use with a small kernel size. When the kernel size is larger than 12 it is more efficient to use the *freqfilter* function.

## Menu mode path

Main --> filter+ --> spfilter

**Name**

statistical functions: ***min, max, median, std, var, mode*** -- evaluate minimum, maximum, median, standard deviation, variance, and mode of an input array.

**Synopsis**

output = *min* (input)
output = *max* (input)
output = *median* (input)
output = *std* (input)
output = *var* (input)
output = *mode* (input)

**Description**

*min, max, median, std,* and *var* evaluate the minimum, maximum, median, standard deviation, or variance of an input array. 'mode' returns the intensity value of the peak in the array's histogram.

**Menu mode path**

Main --> builtin+ --> stat

## Name

*surfit* -- perform a least squares fit through a set of tie points and warp the image

## Synopsis

output = *surfit* (input, "tiept_file", order_of_fit)

## Description

*surfit* performs a least squares fit through a set of irregularly spaced tie points, then resamples the image based on the fitted coefficients.

## Menu mode path

Main --> geom+ --> surfit

## Name

*tiept* -- resample an image based on a set of tie points

## Synopsis

output = *tiept* (input, "tiept_file")

## Description

*tiept* resamples an image based on a set of regularly spaced tie points. Bi-linear interpolation is used while resampling.

## Menu mode path

Main --> geom+ --> tiept

**Name**

typecast functions: *char, int, float* -- convert datatype of input symbol to character, integer, or float datatype

**Synopsis**

output = *char* (input)
output = *int* (input)
output = *float* (input)

**Description**

These three functions convert the data in the input symbol to unsigned char, integer, or float datatypes respectively.

**Menu mode path**

Main --> builtin+ --> typecast

## Appendix A -- CIPE menu configuration file

```
MENU mainmenu
setup/setup
symbol+/Symbol
disp+/Display
mssdisp+/Mssdisp
builtin+/Builtin
xform+/Xform
filter+/Filter
restore+/Restore
geom+/Geom
stretch+/Hstretch
END

MENU Builtin
symbol+/Symbol
disp+/Display
add_func/appl
myfunc/appl
typecast/appl
pattern/appl
math/appl
stat/appl
matrix+/Matrix
END

MENU Matrix
matop/menu_bltin
cmatop/appl
constop/appl
END

MENU Xform
symbol+/Symbol
disp+/Display
rfft2/appl
cfft2/appl
powerspec/appl
END

MENU Filter
symbol+/Symbol
disp+/Display
kernel/appl
spfilter/appl
freqfilter/appl
medfilter/appl
reseau/appl
END
```

## Appendix A -- CIPE menu configuration file

```
MENU Geom
symbol+/Symbol
disp+/Display
pyramid+/Pyramid
surfit/appl
gentie/appl
tiept/appl
rotate/appl
scale/appl
concat/appl
END

MENU Pyramid
reduce/appl
expand/appl
merge/appl
END

MENU Restore
symbol+/Symbol
disp+/Display
feature_psf/appl
image_psf/appl
kernel/appl
invfilter/appl
ME/appl
ML/appl
END

MENU Hstretch
symbol+/Symbol
disp+/Display
percent/appl
END

MENU Mssdisp
symbol+/Symbol
stretch+/Stretch
mssdraw/mssdisp
mssplot/mssdisp
erase/display
zoom/display
END

MENU Display
symbol+/Symbol
alloc+/Alloc
stretch+/Stretch
zoom/display
draw/display
draw_color/display
erase/display
histo/display
cursor/display
hardcopy/display
END
```

## Appendix A -- CIPE menu configuration file

```
MENU Stretch
linear/display
table/display
END

MENU Alloc
alloc/display
select/display
dealloc/display
disp_list/display
END

MENU Symbol
list/list_symbol
read/read_image
copy/copy_symbol
assign/assign_data
save/save_image
delete/delete
print/print_data
END
```

## Appendix B -- CIPE function dictionary file

```
function add_func
pathname "bltin_function"
help "add_func (function_name, pathname, help_msg) "
!
! cube
function cube_reset
pathname "bltin_function"
help "cube_reset"
!
! symbol stuff
function copy
pathname "bltin_function"
help "output = copy (input, {start_line, start_sample, number_of_line, number_of_sample})"
function delete
pathname "bltin_function"
help "delete (input)"
!
!builtin 2arg
function matop
pathname "bltin_function"
help "output =  matop (operation, input1, input2)"
function add
pathname "appl/bltin/cp/matop"
help "output = add (input1, input2) "
function sub
pathname "appl/bltin/cp/matop"
help "output = sub (input1, input2) "
function mult
pathname "appl/bltin/cp/matop"
help "output = mult (input1, input2) "
function div
pathname "appl/bltin/cp/matop"
help "output = div (input1, input2) "
!
!typecast
function typecast
pathname "appl/bltin/cp/bltintype"
help "output = typecast (output_data_type, input) "
function char
pathname "appl/bltin/cp/bltintype"
help "output = char (input) "
function int
pathname "appl/bltin/cp/bltintype"
help "output = int (input) "
function float
pathname "appl/bltin/cp/bltintype"
help "output = float (input) "
!
! math functions
function math
pathname "appl/bltin/cp/bltinmath"
help "output = math (operation, input) "
function sqrt
pathname "appl/bltin/cp/bltinmath"
help "output = sqrt (input) "
function log
pathname "appl/bltin/cp/bltinmath"
help "output = log (input) "
function log10
pathname "appl/bltin/cp/bltinmath"
help "output = log10 (input) "
```

# Appendix B -- CIPE function dictionary file

```
function square
pathname "appl/bltin/cp/bltinmath"
help "output = square (input) "
function abs
pathname "appl/bltin/cp/bltinmath"
help "output = abs (input) "
function negate
pathname "appl/bltin/cp/bltinmath"
help "output = negate (input) "
!
!statistics functions
function stat
pathname "appl/bltin/cp/bltinstat"
help "output = stat (operation, input) "
function min
pathname "appl/bltin/cp/bltinstat"
help "output = min (input)"
function max
pathname "appl/bltin/cp/bltinstat"
help "output = max (input)"
function mean
pathname "appl/bltin/cp/bltinstat"
help "output = mean (input)"
function median
pathname "appl/bltin/cp/bltinstat"
help "output = median (input)"
function mode
pathname "appl/bltin/cp/bltinstat"
help "output = mode (input)"
function std
pathname "appl/bltin/cp/bltinstat"
help "output = std (input)"
function var
pathname "appl/bltin/cp/bltinstat"
help "output = var (input)"
!
! complex 2arg matrix operation
function cmatop
pathname "appl/bltin/cp/cmatop"
help "{out_real, out_imagi} = cmatop (operation, input1_real, input1_imagi, input2_real, input2_imagi)
-- not implemented in CLI mode yet"
function cmpadd
pathname "appl/bltin/cp/cmatop"
help " {out_real, out_imagi} = cmpadd (input1_real, input1_imagi, input2_real, input2_imagi)
-- not implemented in CLI mode yet"
function cmpsub
pathname "appl/bltin/cp/cmatop"
help " {out_real, out_imagi} = cmpsub (input1_real, input1_imagi, input2_real, input2_imagi)
-- not implemented in CLI mode yet"
function cmpmult
pathname "appl/bltin/cp/cmatop"
help " {out_real, out_imagi} = cmpmult (input1_real, input1_imagi, input2_real, input2_imagi)
-- not implemented in CLI mode yet"
function cmpdiv
pathname "appl/bltin/cp/cmatop"
help " {out_real, out_imagi} = cmpdiv (input1_real, input1_imagi, input2_real, input2_imagi)
-- not implemented in CLI mode yet"
```

## Appendix B -- CIPE function dictionary file

```
!
! matrix arithmetic operation with a constant
function constop
pathname "appl/bltin/cp/constop"
help "output =  constop (operation, input1, input2)"
function cadd
pathname "appl/bltin/cp/constop"
help "output =  cadd (input1, input2)"
function csub
pathname "appl/bltin/cp/constop"
help "output =  csub (input1, input2)"
function cmult
pathname "appl/bltin/cp/constop"
help "output =  cmult (input1, input2)"
function cdiv
pathname "appl/bltin/cp/constop"
help "output =  cdiv (input1, input2)"
!
! display utilities
function alloc
pathname "display"
help "alloc (host_name, device_type, window_size) "
function select
pathname "display"
help "select (unit_number) "
function dealloc
pathname "display"
help "dealloc -- no argument needed"
function disp_list
pathname "display"
help "disp_list -- no argument needed"
function draw
pathname "display"
help "draw (input, {start_line, start_sample}) "
function draw_color
pathname "display"
help "draw_color (input_red, input_green, input_blue, {start_line, start_sample}) "
function erase
pathname "display"
help "erase (i/o/a, {start_line, start_sample, number_of_line, number_of_sample}) "
function lstretch
pathname "display"
help "lstretch (min, max)"
function zoom
pathname "display"
help "zoom (i/o/a, zoom_factor, {start_line, start_sample}) "
! multi spectral data display
function mssdisp
pathname "disp/mssdisp"
help "mssdisp (input, band, {start_line, start_sample}) "
!
! pattern generator
function pattern
pathname "appl/bltin/host/pattern"
help "output = pattern (pattern_type, pattern_size{length,width},
inten{dark,light},size{length,width}) -- consult menu mode for
the param of specific pattern"
```

## Appendix B -- CIPE function dictionary file

```
!
! spatial filter
function spfilter
pathname "appl/filter/cp/spfilter"
help "output = spfilter (input_image, input_kernel)"
function medfilter
pathname "appl/filter/cp/medfilter"
help "output = medfilter (input_image, {nlw(3), nsw(3)}, thresh(0))"
!
! frequency filter
function freqfilter
pathname "appl/filter/cp/freqfilter"
help "output = freqfilter (input_image, input_psf, mode)"
!
! preprocessing
function reseau
pathname "appl/filter/cp/prep"
help "output = prep (input, reseau_file) -- this program requires hypercube"
!
! kernel generator
function kernel
pathname "appl/filter/host/kernel"
help "output = kernel (psf_type, {operand1, <operand2>})"
!
! power spectrum
function Power
pathname "appl/xform/cp/Power"
help "output = Power (input)"
function powerspec
pathname "appl/xform/cp/powerspec"
help "output = powerspec (real_fft_result, imagi_fft_result, fold(y/n))"
!
! complex input fft2
function cfft2
pathname "appl/xform/cp/cfft2"
help "(output_real, output_imagi) = cfft2 (input_real, input_imagi, mode)
-- not implemented in CLI mode yet"
!
! real input fft2
function rfft2
pathname "appl/xform/cp/rfft2"
help "(output_real, output_imagi) = rfft2 (input, mode)
-- not implemented
in CLI mode yet"
!
! restoration using inverse filter
function invfilter
pathname "appl/restore/cp/invfilter"
help " output = invfilter (input_image, input_psf, noise_level(float), niter, {lambda(float), del_lamda(float)})"
!
! restoration using maximum likelihood constraint
function ML
pathname "appl/restore/cp/ML"
help " output = ML (input_image, input_psf, noise_level(float), niter)"
!
! restoration using maximum entropy constraint
function ME
pathname "appl/restore/cp/ME"
help " output = ME (input_image, input_psf, noise_level(float), d_lamda(float), niter)"
```

# Appendix B -- CIPE function dictionary file

```
!
!psf
function feature_psf
pathname "appl/restore/host/gen_psf"
help "it needs interactive graphic device -- not available in CLI mode"
function image_psf
pathname "appl/restore/host/gen_psf"
help "it needs interactive graphic device -- not available in CLI mode"
!
! pyramid related functions
! pyramid reduce
function reduce
pathname "appl/geom/cp/reduce"
help " output = reduce (input, pyramid_level)"
! pyramid expand
function expand
pathname "appl/geom/cp/expand"
help " output = expand (input, pyramid_level)"
function merge
pathname "appl/geom/cp/merge"
help " output = merge (input1, input2, sample1, sample2, pyramid_level)"
! concatenate two images
function concat
pathname "appl/geom/host/concat"
help " output = concat (input1, input2, istat(0 for horiz, 1 for vertical),
iave(i if averaging))"
function rotate
pathname "appl/geom/cp/rotate"
help " output = rotate (input, angle(float), clip_option)"
function scale
pathname "appl/geom/cp/scale"
help " output = scale (input, x_scale_factor(float), y_scale_factor(float))"
function surfit
pathname "appl/geom/cp/surfit"
help "output = surfit (input, tiept_file, order_of_fit)"
function gentie
pathname "appl/geom/cp/gentie"
help "output_tiept_file = gentie (input_tie_file, order_of_fit, tiept_param{nptx,npty,gapx,gapy})
-- use menu mode"
function tiept
pathname "appl/geom/cp/tiept"
help "output = tiept (input, tiept_file)"
function data_dist
pathname "appl/diag/cp/data_dist"
help "output = data_dist (input, dist_type)"
function percent
pathname "appl/stretch/cp/perc_stretch"
help "output = perc_stretch (input, lower_perc, upper_perc)"
function sar
pathname "appl/geom/cp/sar"
help "output = sar(input, zres/xyres)"
```