University of California

Lick Observatory Technical Reports

No. 50

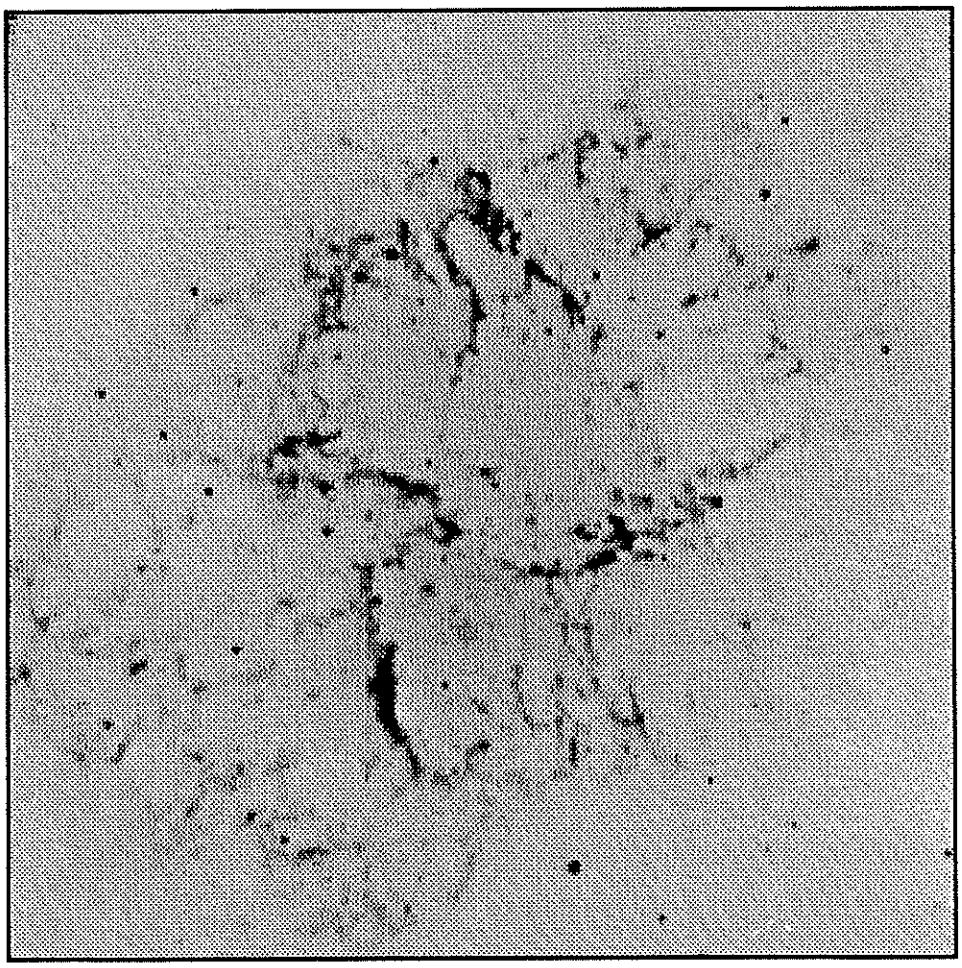
# The VISTA Cookbook

Richard W. Pogge
Robert W. Goodrich
Sylvain Veilleux

Santa Cruz, California

August 1988

# Vista



## *Image Processing Cookbook*

*Lick Observatory*

*University of California, Santa Cruz*

Composed in Times Roman on the Lick Observatory VAX 11/780 computer using Leslie Lamport's LaTeX, and printed on an Imagen 8/300 laser printer. Figures were composed in PostScript and printed on an Apple LaserWriter II-NTX connected to a Sun 4/280.

**On the Cover:** CCD image of the Crab Nebula Supernova Remnant in the light of the [S II] $\lambda\lambda$6716, 6731 emission-lines. Taken with the TI 500×500 CCD and the focal reducing camera on the 1-meter Anna Nickel Telescope at Mt. Hamilton. The image was processed and prepared for display by Rick Pogge using VISTA.

# Table of Contents

# Chapter 1

# Introduction

## 1.1 Overview of VISTA

VISTA is an interactive image reduction and analysis package developed at Lick Observatory. VISTA development was begun by Richard Stover and Tod Lauer in the early 1980's to provide software for the reduction of data obtained with the new CCD cameras coming into use at Mount Hamilton. General agreement among the astronomy departments at the other UC campuses led to the adoption of VISTA as the standard image processing package, and all campuses (at first) installed similar computer hardware on which is was to be run (Digital Equipment Corporation VAX/VMS systems AED 512 color image displays). As a result, the VISTA package from the start was tailored to this equipment. A general overview of the architecture of the VISTA package may be found in an article by Richard Stover in *Instrumentation for Ground-Based Optical Astronomy: Present and Future*, the proceedings of the 1988 Summer Workshop held in Santa Cruz.

Since that time, the VISTA package has taken on a life of its own, spreading to many parts of the world besides the UC system. The principal agents of this spread have been former graduate students, postdocs, and visitors who have taken VISTA with them when they departed. The most recent release, Version 4.0, is nearly complete at the time of this writing. So far, it has been successfully used to VAX/VMS and ISI computers, and a port to SUN Microsystems 3 and 4 computers is slated for the near future (no promises). Non-image graphics (*e.g.*, line drawings, spectra, etc) is done using Lick MONGO, a fully portable incarnation of the old Tonry MONGO which has evolved at Lick Observatory over the years. Lick MONGO supports a variety of graphics display terminals, as well as the most common hardcopy devices, including PostScript laser printers.

VISTA has no "philosophy" *per se.* It is designed to be a small package (small, that is, by image processing standards) that lends itself readily to user modification. As such, it is relatively easy for an astronomer with a modest command of Fortran to write custom commands to take care of special image processing needs. It is not intended to be all encompassing, like the larger MIDAS or IRAF packages. VISTA does have its faults, which any long-time user will be glad to tell you all about, and which you will discover for yourself in time. Recall that it was produced by graduate students and research astronomers, not professional programmers, so it has a nice, homey, slapped together in the garage sort of feel at times (it can also be incomprehensible as hell, too. So it goes).

The general development of routines for the VISTA package has been driven by the needs of individual researchers, principally graduate students working on their Ph.D. dissertations (*i.e.,* driven by that oldest of instincts, self preservation). In addition, it is often these graduate students who are called upon for consultations when problems arise, or to assist visiting scientists with getting started using the VISTA package to reduce their data. Since graduate students are not eternal (at least, we're not *supposed* to be), it was decided among the present hard core of VISTA gurus to distill our experience into a "VISTA Cookbook" before we kick off for the boondocks. This cookbook is to serve as a guide to basic image reduction procedures with VISTA for new users and visitors. The idea was for us not to simply vanish to a postdoc somewhere and leave the people who have come to depend on us at Lick "turning slowly in the wind."

## 1.2 The Cookbook

This cookbook is designed as a resource for VISTA users interested in accomplishing basic image reduction chores. It is divided into 8 chapters, each of which covers a single topic. Within each chapter, the discussion starts at the most basic level, progressing towards more advanced techniques. It is not all encompassing, but that is impossible, as each person's data will ultimately require different treatment depending on how it was taken and what information they wish to derive from it. The intent is to provide new users with a set of basic VISTA tools which can be used to develop individually tailored image reduction and analysis procedures.

The format is laid out so as to guide the user as much as possible. Different typefonts are used to distinguish the running commentary from input required by the computer and the names of external data files. They are:

| Cookbook Fonts | |
|---|---|
| Font | Use |
| `EXTSPEC` | Computer command input (either VISTA or operating system) |
| *HAMPREP.PRO* | Names of external data files |
| ⟨RETURN⟩ | Particular keys on a terminal keyboard |

User input prompts are used throughout the cookbook. These are meant to reproduce, as well as possible, what you should see on the screen, and to remind you that the input is either a VISTA command or an operating system command (DCL or Unix). The prompts are:

| Command Prompts | |
|---|---|
| Prompt | Meaning |
| `GO:` | The default VISTA command prompt. |
| `$` | The default DCL prompt for the VMS operating system. |

It is expected that the cookbook will be used in conjunction with the VISTA Help Manual. The help manual is available in the form of a computer printout, as well as accessible on-line while running VISTA.

A brief synopsis of the cookbook is as follows: We begin with a brief tutorial of basic VISTA commands (Chapter 2) to give new users a place to get started. With the preliminaries out of the way, we begin our discussion of how to use VISTA for a variety of image processing tasks. Chapter 3 is an overview of basic reduction procedures common to most image processing tasks (imaging or spectrophotometric data), such as flat fields, dark frames, etc. In Chapter 4 we describe the basic VISTA procedures for 2-D image reduction. Basic spectral data reduction techniques are described in Chapter 5. An advanced discussion of reduction techniques for data obtained with the Hamilton echelle spectrograph at Mount Hamilton is presented in Chapter 6. Chapter 7 presents a brief tutorial on the use of command procedures in VISTA for automating various reduction tasks. Finally, Chapter 8 is a brief collection of "dirty tricks" that some folks might find useful.

Following the chapters are a series of useful appendices. Appendix A is a compendium of all of the VISTA commands organized by function. A set of examples of VISTA command procedures, ranging from very simple to rather complex, are

given in Appendix B. Appendix C presents a brief discussion of FITS header cards and how they are used in VISTA.

VISTA has been ported to both VMS and Unix (ISI) systems, using a variety of color image display devices. Throughout this cookbook, we shall use examples that are for the most common installation at the time of writing, namely a VAX/VMS computer interfaced with an AED color graphics display. All of the basic image interaction commands have exact equivalences for different computers/display devices (*e.g.,* for a ISI computer with a color monitor running Unix). Variations are described in the Help Manuals for the different flavors of VISTA.

## 1.3 Final Remarks

For the future? It is hoped that with the retirement of the VAX computers at Santa Cruz that VISTA will be ported to Sun 4s running Sun/OS 4.x. The VISTA version 4.0 in progress for the VAX/VMS computers is primarily modifications to facilitate this port. It is clear that VISTA has a fairly large cadre of loyal users, and so will probably be supported in some state or another for the present time.

# Chapter 2

# A Brief VISTA Tutorial

This chapter presents a brief tutorial of basic VISTA commands for the new user. More experienced users can pass this section over, although it may serve as a useful reference to basic procedures when you're so far into it that you start to forget the simple stuff.

The first section of the VISTA Help Manual gives the full details of what is needed to run VISTA, and some of the basic command syntax. Rather than regurgitating the contents of the help manual in altered form, this tutorial is a simple collection of illustrative tasks making use of the most basic VISTA commands. As with all chapters, we will start out simple and progress towards more sophisiticated commands. These are commands which are general to any image processing chore. Before reading this section, read over the first part of the VISTA Help Manual. Make sure you understand all about pixels, rows, column, image buffers, image headers and the like. If this stuff is greek to you. Stop now and read the first part of the manual.

One of the principal assumptions that this and all subsequent sections of the cookbook will make is that your local VISTA custodian has followed the VISTA Installation Guide to the letter. This means that directory and subdirectory names, file names, executable files, logical links, EVERYTHING has the names we have suggested. If your local custodian has his/her own ideas about what to name things, then you'll have to find out what they are and make the necessary accommodation to what follows. We sincerely hope that isn't necessary.

## 2.1 Starting VISTA

First and foremost. VISTA is big. Real big. OK, not that big, but sufficiently large and greedy with computer resources that your system manager will need to configure your account so you can run VISTA comfortably. On large time-sharing systems, many users running VISTA at the same time will make enemies. If you aren't configured correctly, then you'll make even more. The VISTA Installation Guide which came with your VISTA tape has the details of how your system manager can configure your account appropriately to your machine (VAX, ISI, or Sun). Half of the problems with first-time users comes from not first configuring properly.

### 2.1.1 Logical Assignments

The first thing a new user needs is to assign a set of logical name assignments that will be used by VISTA during execution. These point to directories set aside for temporary image storage, or external data files. These assignments are made at the operating system level. This is usually major point of confusion number 1.

If you are using VISTA on a VMS machine, then you would want to make these assigments in your *LOGIN.COM* file. Suppose your username was FRED, then you would want to insert the following lines of DCL into your *LOGIN.COM* file, $-signs and all:

```
$! VISTA Directory and File Assignments
$ DEFINE V_CCDIR [FRED.IMAGES]
$ DEFINE V_SPECDIR [FRED.SPECTRA]
$ DEFINE V_PRODIR [FRED.PROCEDURE]
$ DEFINE V_DATADIR [FRED.VDATA]
$ DEFINE V_STARTUP [FRED.PROCEDURE]STARTUP.PRO
```

The first four lines define the four necessary storage directories for Images, Spectra, VISTA Command Procedures, and auxilliary data files respectively. These must be created ahead of time using the VMS `create/directory` command, otherwise VISTA will spit angry error messages at you for sending it to an non-existant directory. For example, if they did not already exist, you would type in:

```
$ CREATE/DIRECTORY [FRED.IMAGES]
$ CREATE/DIRECTORY [FRED.SPECTRA]
$ CREATE/DIRECTORY [FRED.PROCEDURE]
$ CREATE/DIRECTORY [FRED.VDATA]
```

Unless you totally trash your account somehow, this only needs to be done once. Note that later on you can change the default directories from inside VISTA during execution using the SETDIR command.

The fifth line points to a pre-existing VISTA procedure file that contains VISTA commands to be run at startup time. It is a simple text file (see Appendix B for some simple examples of procedure files). While considered discretionary by VISTA, they are generally so useful, that you should get in the habit of making one and using it. A startup file should contain your favorite command abbreviations (see Appendix B), special variable assignments, or simply just erase the screen for you at the start. The latter is simple enough, define your file *[FRED.PROCEDURE]STARTUP.PRO* to have just two lines:

```
CLEAR
END
```

All procedure files must end with the END command. The first character of a line starts on the first position.

With these logicals in place, we're ready to begin.

### 2.1.2 Running VISTA

We will assume that your local VISTA custodian has followed the suggested installation and had the system manager define the command word "VISTA" to run VISTA for you, no fuss, no muss. If your site is different, then you'll need to find your VISTA custodian and ask them:

1. Where the local VISTA executable lives?
2. What it is called?

On a VMS machine, you would type something like:

```
$ RUN [VISTA.BASE]VISTA
```

The exact directory syntax may vary from site to site.

VISTA should respond by erasing the screen (depends a little on the type of terminal or workstation you are on), and print the welcome message. After the welcome message (informing you that this is indeed VISTA you're running and not

MONGO or some other damn fool thing), you will be shown, page by page, the current VISTA news. If your custodian is the diligent type, then the news will tell you of any changes to VISTA that might affect you, warnings about bugs (there are *always* bugs), and other bits of useful information. Make a habit of reading the news as you begin. When you're done with a page, hit the space bar to see the next page.

### 2.1.3 Terminal Type

One last hurdle to leap before you can actually issue VISTA commands. It is also major point of confusion number 2.

Know what kind of terminal you are on. After the news scrolls past, VISTA will stop and ask you what kind of graphics terminal you are on. You should be on some kind of graphics terminal or fancy workstation. If not, stop and go find one. VISTA really doesn't work too well from a line printer. Also, unless you have special drivers or an investment in a good quality terminal emulator, VISTA won't really work on a PC. If you never want to do graphics, you're OK, but VISTA *is* an *image* processing program, somehow not looking at your data graphically seems a bit pointless.

VISTA will first tell you what the default devices for your installation are. Your VISTA custodian should have set these to the most common device when VISTA was installed on your machine. If you are on one of the default devices, simply hit return and VISTA will continue on.

If you are not on the default device, then find out what type of terminal you *are* on. If you type `?` in response to the `terminal type` prompt, you will be given a menu of the devices that VISTA supports, along with a number representing the "device code." If your terminal is not one of these, then you may be in trouble. VISTA uses Lick MONGO, and Lick MONGO only supports those terminal types that we have in hand at UCSC. (It's awful hard to support terminals we've never heard of.) Most (but not all) terminals on the market these days look and smell like DEC VT100s with Retrographics boards, so you're probably safe if the exact terminal you are on doesn't show in the device menu. Check your owner's manual and proceed with fingers crossed.

If you don't know, then to avoid a potentially major VISTA hassle, ***don't guess.*** Find someone who knows. If you're on a VT100 and you tell VISTA you're a Sun 4 WorkStation, VISTA will take your word for it and make you pay by crashing horribly into your lap. Don't laugh, half of the "VISTA doesn't work" phone calls

seem to be people doing just this sort of thing. VISTA is pretty good, but it doesn't read minds. So it goes.

## 2.2 Stopping VISTA

First learn how to start VISTA. Second learn how to stop VISTA. The rest is rock-n-roll. There are two ways to stop VISTA. One neat and the other sloppy but occasionally useful.

To stop VISTA at any time you have the VISTA `GO:` prompt, type the command:

```
GO: QUIT
```

VISTA will stop cold, dispose of all image buffers and variables, and return you to your computer's operating system. Simple.

If VISTA should hang up (or you're just mad at it and want to kill it in a satisfying way), then type a ⟨CTRL⟩ ⟨Y⟩ . VISTA will terminate immediately. Admittedly drastic and a little sloppy, it is good in a pinch when you've sent VISTA into a tight spin and want to bale out.

In one of those rare instances where VISTA locks you out and ignores your pleas to stop, more drastic action needs to be taken. This is a job for your system manager. If VISTA should hang up during reading or writing to tape, this is a more serious (and thankfully very rare) instances of what is known as a "VISTA Death Roll." The cause of a VISTA Death Roll seems to be some combination of crowded system and marginal tape-drive controller. For heaven's sake don't kill your job, especially on a VMS machine. With the usual old VAX tape-drive controllers, you could hang the controller so bad the VAX would need to be rebooted. That sort of thing makes all the theorists running long batch jobs hate you. It also gets in the system manager's face. Theorists are one thing, torqued-off system managers are quite another.

## 2.3 The VISTA Command Prompt

After all the preliminaries are done with, VISTA is ready to accept command input. The screen should show the VISTA command prompt:

```
GO:
```

VISTA commands may be typed in either upper or lower case. VISTA is not case sensitive (although in future Unix versions, file names will most definitely be case sensitive). Each command is terminated by hitting the (RETURN) key. Cases where the command is longer than 75 characters is dealt with later. The following subsections cover sets of simple commands.

Some people like to have VISTA "beep" at them when it wants command input. You can make VISTA do this by typing the command:

```
GO: BELL Y
```

It has its uses, especially to let you know when a particularly long processing routine has finished. Mostly it's a matter of taste.

## 2.4 A Simple VISTA Session

This section describes a simple VISTA session in which you will read images off of tape and perform simple tasks like displaying the images, writing them to disk files, move them around among VISTA buffers, and so forth. For this brief example, we will have the following items in hand:

1. Assume VISTA installed on a VMS computer (like a MicroVAX).
2. VISTA running on a graphics terminal with a nearby AED Color Display terminal.
3. A FITS tape with raw CCD images loaded onto a tape drive which is known to VISTA as UNIT 0. This tape has an unknown number of images on it (Do I have 40 images or 41 images? In all the confusion, I lost count...)

Task 1: What Images are on Tape? VISTA reads tapes which have been written in the FITS format (FITS stands for Flexible Image Transport System). FITS is a standard format for storing astronomical data, originally developed for radio data but adopted for 2-D imaging data. Presumably it is portable anywhere. Lick data, and FITS tapes written with VISTA have proven to be portable to MIDAS, IRAF, and AIPS at many sites, so this shouldn't be a problem. If your tape is not a FITS

format tape, then you are out of luck (for example, some Palomar data tapes, Lick IDS tapes, etc. are not FITS format).

To read a tape, first load it onto the tape drive and, when properly spooled, put the tape drive on-line. Check with your VISTA custodian for the unit numbers that VISTA uses for each of your tape drives (if you have only 1 tape drive, it should be named `UNIT=0`, but check anyway).

To mount a tape on the drive named `UNIT=0`, issue the command:

```
GO: MOUNT UNIT=0
```

And VISTA will either verify the mount, or come back with an error message telling you why it failed.

To find out what images are on a tape, you use the `TDIR` command (for Tape DIRectory). `TDIR` has two modes, wordy and terse. If you want all the gory details, then simply type:

```
GO: TDIR UNIT=0
```

And the headers of all of the FITS images will be translated and the essential details printed onto the screen in a dense format (known as "full" format).

We aren't interested in all that, and since we want a printout we can read at our leisure, we'll do a brief tape directory and redirect the output to an external file which can be printed. This is done with the following sequence of commands:

```
GO: TDIR BRIEF UNIT=0 'MYSTERY TAPE' >TAPE.LIST
GO: $ PRINT TAPE.LIST
```

The first line is a `BRIEF` tape directory of the FITS tape mounted on `UNIT=0`, labels the directory as `'MYSTERY TAPE'` and redirects the output to a file called *TAPE.LIST*. The `>` is the key used to redirect output. Without the redirection "arrow," the tape directory would be printed on the terminal screen. The second line issues a DCL (`$`) command to print the file *TAPE.LIST* on the MicroVAX line-printer. Note that everything after the `$` is not a VISTA command, but a VMS operating system command. The `$` allows you to issue operating system commands without stopping VISTA execution.

Task 2: Read in 2 Images for Inspection Now that we have the tape directory in hand (there were 41 images), we want to read in images 10 and 11 and look at them. Since images 10 and 11 are near the front of the tape, it is best to rewind the tape using the following sequence of commands:

```
GO: DISMOUNT UNIT=0
GO: MOUNT UNIT=0
```

A DISMOUNT followed immediately by a MOUNT is a sloppy but effective rewind. VISTA knows how to read backwards to an image it has passed on a tape, but that can be very slow when it has a long way to go, as in this case.

Read Image 10 into VISTA buffer 1, and Image 11 into VISTA buffer 2 by typing the commands:

```
GO: RT 10 1 UNIT=0
GO: RT 11 2 UNIT=0
```

The tape will move forward to the image, and read it off tape into the requested VISTA buffer. It translates the image header and prints a verbose summary of essential information on the screen.

To get a brief summary of the contents of the VISTA buffers, issue the command:

```
GO: BUF
```

VISTA will print a the image dimensions (in rows and columns), the origin of each image (in rows and columns), the name of the image, and other useful information. A more complete listing of the buffers would be gotten by typing the command:

```
GO: BUF FULL
```

Note that BUF is an abbreviation of the full command name, BUFFER. VISTA recognizes unambiguous abbreviations for all commands (but *NOT* for the keywords making up a command). If you give it an ambiguous abbreviation, it will tell you so. Try it, type:

```
GO: B
```

And VISTA will tell you all of the commands that begin with B and ask you to be more specific.

Task 3: Display the Images on the Color Display Let's have a look at the images. Image display is done using some kind of color display (typically an AED or some fancy color workstation like a Sun 4/110). To display the image in buffer 1, issue the command:

```
GO: TV 1
```

VISTA will automatically translate the image intensities into colors and display the image on the color monitor. Since we have used no command arguments for TV, we will get the default "color" map (Black and White monochrome), and 4 times the image mean intensity (computed automatically by the RT command) will be used to set the scale of the mapping (*i.e.*, 0 will be full black, and 4 times the mean intensity will be full white).

If black and white doesn't do it for you, you can load a color map. The fast an easy way is to use the command:

```
GO: COLOR CF=RAIN
```

Which will load the "RAINbox" color map. (CF stands for "color file," and artifact of Version 1.0). There are other color maps available, see the manual. If you issue COLOR without arguments, then it will begin asking for data that it wants to construct a custom color map. Color maps are arcane, and beyond the scope of this cookbook to go into.

Another way, to avoid typing two commands, is to specify the color map with the TV command. Don't ever do this to change the color map of the image currently on the display, that would be a horrible waste of CPU time. To specify the color map at display time, you would issue the command:

```
GO: TV 1 CF=RAIN
```

Task 4: Plot a Row or Column of an Image Another way to look at your data is to plot along a given row or column. To plot the image intensities along ROW 250 of the image in buffer 2, you would issue the command:

```
GO: PLOT 2 R=250
```

Default X and Y axis limits will be computed, and the plot will appear on your terminal screen. If instead, you see no plot but a stream of apparent gibberish of characters, you haven't specified the right terminal type (see §2.1.3 above).

You can change the plotting limits using keywords. For example, you're only interested in the data between Columns 200 and 350 along Row 250. To restrict the plot to this range, you'd type:

```
GO: PLOT 2 R=250 XS=200 XE=350
```

Now, you only want to see the intensity values between 0 and 255.3, thus you would type:

```
GO: PLOT 2 R=250 XS=200 XE=350 MIN=0.  MAX=255.3
```

Finally, you'd like a hardcopy to post on your door. To do this you would add an additional keyword, HARD as follows:

```
GO: PLOT 2 R=250 XS=200 XE=350 MIN=0.  MAX=255.3 HARD
```

No plot will appear on your screen, and the hardcopy will (hopefully) appear on the plotting device. If it does not, then you may have to consult with your local VISTA custodian to see if something is not set correctly.

Task 5: Save the Images in Disk Files Having inspected the images, you want to save them both in Disk files for easy access later. Disk files are good for temporary storage during reduction and analysis, but they eat up a large amount of space. If you are working on an account with a fixed disk space quota, that quota may have to be increased if you want to store many images temporarily on disk. As with all shared computer resources, you should be frugal.

Let us store the image in buffer 1 in a disk file named *OLLIE.CCD*, and the image in buffer 2 as *STANLEY.CCD*. To do this, you would issue the commands:

```
GO: WD 1 OLLIE
GO: WD 2 STANLEY
```

The .CCD extension is automatically appended to the file name, and the files are written into the image directory you specified at runtime with the V_CCDIR logical name. If you wanted to write the file somewhere else, you would add the directory information onto the front of the file.

To conserve space as much as possible, the image values are converted into integers between −32767 and +32767, and the scaling coefficients stored in the image headers. When the images are read back in from disk, VISTA will translate the image intensities back into their original values. If you want the images to be written in their full REAL*4 precision, you would add the FULL keyword to the WD command.

Task 7: Moving Images around Buffers To move the image in buffer 1 to buffer 3, you would type the command:

```
GO: COPY 3 1
```

In VISTA operations between two buffers, the command syntax is always of the general form:

```
GO: OPERATION <to> <from>
```

or equivalently:

```
GO: OPERATION <destination> <source>
```

This resembles so-called Reverse Polish Notation (RPN) popular among owners of HP hand calculators. For the more algebraically minded, another way to think about the order of buffers is:

```
GO: OPERATION <buffer to change> <buffer left unchanged>
```

OK, back to the task at hand (moving images around, remember?). If you now issue a `BUF` command, you will have 3 images connected, of which the images in buffers 1 and 3 are now identical. You can get rid of the old image in buffer 1 by issuing the command:

```
GO: DISPOSE 1
```

This will erase the image from memory, and free that space for another image.

Using `COPY` to copy an image into a different buffer which is already occupied will have the effect of deleting the existing image, and replacing it with the source image indicated by the `COPY` command.

Task 8: Dismount the Tape and Quit VISTA The first step in finishing a session is to dismount any tapes. To do this, issue the command:

```
GO: DISMOUNT UNIT=0
```

Which will rewind and dismount it. You will need to physically take the tape off-line and remove the tape from the drive. The second step, while not absolutely necessary is illustrative, dispose of the VISTA buffers. You would do this by issuing the command:

```
GO: DISPOSE ALL
```

Finally, stop VISTA by typing the command:

```
GO: QUIT
```

VISTA will terminate, and return you to the operating system prompt after a brief pause.

## 2.5 Final Remarks

VISTA is a system best learned by using. Typical learning times are a few days to many months, depending on the individual, and the degree of sophisitication of the image procession task at hand. Most new users find it very useful to "noodle" around with VISTA, using a dummy image stored on disk, and trying various commands out. VISTA is surprisingly easy to use, and quite flexible once you're accustomed to it. It is very easy for a first-time user to sit down and get an image read in from a raw data tape and displayed in color right away.

# Chapter 3

# Basic Image Processing

This chapter describes image processing techniques that are "basic" in that they are common to almost all image reduction tasks, whether the final end is imaging or spectrophotometry. They represent the basic corrections that must be applied to the "raw" data from the instrument before measurements can be made from the imaging data.

We have tried, in so far is as possible, to keep to what we perceive as the most basic considerations. Clearly, these cannot be free of the bias introduced by our own experience with the Lick Observatory CCDs and corresponding lack of experience with instruments from other sites. If nothing else, the following will give you some feel for the reasons for the corrections, and how to perform them. Fine details will always come in to mess up simple recipes. We will avoid refinements, but will endeavour to remind you of them to keep things in proper perspective.

It must be pointed that "basic" is not synonymous with "simple" in this context. The corrections described below are crucial to all reduction that follows, and great care must be taken. In addition, we feel that all observers should understand the problems and above all the fundamental limitations of each step so that they may approach their data more sensibly. CCD-based instruments are very complicted, and few observers (the authors included) have sufficient depth of knowledge in electronics, computers, and solid-state physics to be able to fully grasp what CCDs are really giving them. This leads, unfortunately, to a tendency to make CCDs "magic black boxes" which produce data which are then passed to 'black box' reduction programs out of which pops the flattened, de-biased, cleaned-up image. While this is fine in principle, it strikes us as dangerous; black boxes rarely if ever cope with anomalies well. Thus, even a rudimentary appreciation of the "causes and cures" of flat-field correction, dark bias subtraction, and other basic reduction techniques, will help the average observer spot and hopefully resolve anomalous

problems that the black boxes cannot.

## 3.1 Flat Fields

### 3.1.1 Basic Principles

By the "response" of a CCD we mean the conversion between the flux of photons falling into each pixel and the number or "count" in the readout computer's memory that constitutes the image "data". If each pixel on a CCD chip responded exactly the same as its neighbors to the light falling into it, then if it were uniformly illuminated, you would observe the same number of counts ($\pm$ noise) in each pixel. In other words, the response would be "flat". Real CCDs don't behave this way. Many don't even get close.

The response of a CCD is never very uniform. Each pixel has its own response function, an in addition, the average response is often correlated with the response of its neighbors. Thus some CCDs have areas of high sensitivity, and regions of low or zero sensitivity. In addition, the response of a given pixel is also a function of the wavelength of the incident photons. The better the quality of the CCD, the more uniform or "flatter" the pixel-to-pixel response across the face of the chip.

To correct for this pixel-to-pixel response variation, an image is taken with the CCD uniformly illuminated. This is called the "Flat-Field Frame". There are a variety of ways to do this. A white screen in front of the telescope on the dome is imaged (out of focus), or the twilight sky are both common techniques. The goal is to get the illumination at the chip surface as uniform as is possible, with the same approximate mixtures of wavelengths that will be in your data. Remember, the response varies both as a function of pixel and *wavelength* across the face of the CCD. Each different instrument setting (different filters, gratings, whatever) requires its own unique set of flat-field frames. If you change the optics in any way, the flat-field will change.

The techniques used to achieve this are as varied as the observers themselves. We are not going to espouse any particular method for taking flat-field frames, but we will tell you what to do with them once you have them. Simply put, flat-field exposures are more surrounded by lore, black magic, and undiluted voodoo than practically anything else in astronomy since the glory days of RCA 1P21 photomultiplier tubes. Our advice is to do what you think is sensible given your final goal (imaging, spectrophotometry, etc), and do it *consistently.* If you can spare

the time (and it is definitely worth your time), test your techniques exhaustively before adopting them and getting seriously entrenched in observing. The flat-field step is the most important factor in your observing. If you blow the flat-field step, you're screwed. Pure and simple.

Once you have solved the uniform illumination problem, the next most important consideration is that you want as much signal as you can squeeze into your flat-field frames without saturating any part of the CCD. The usual practice is to take many flat-field frames (4 or more) from the beginning and end of the night, and then average them in some fashion to create a mean flat-field frame, either for the night or for the entire run. Since a flat-field contains effects of both CCD and the telescope optics, flat fields will often vary from run to run, and hopefully not from night to night. By "averaging", we mean either a simple sum of all flat-field frames, an arithmetic mean, or even such sophisiticated techniques as median filtering of an image stack (see the 2-D image processing chapter for a discussion of this). Like the illumination question, there is also much lore and voodoo surrounding which is "average flat-field" is best. Again, choose the most sensible to your needs, and be consistent.

A flat-field correction is accomplished by generating a flat-field frame using one of the techniques alluded to above. The program image is then divided by the flat-field frame to remove the pixel-to-pixel response variations, resulting in a "flattened" image. What a flat-field frame really contains is the relative pixel-to-pixel response normalized to the average image intensity in the frame. Thus to restore the image "counts", it is necessary to follow division by the flat-field frame by re-scaling with the flat-field normalization.

### 3.1.2 A Simple Example

In VISTA, the flat-fielding procedure would be as follows: Suppose that the program image to be flattened is in Buffer 1, and the flat field frame is in Buffer 2. The first step is to compute the mean intensity in the flat-field frame to be used as the normalization factor. Then divide the program image by the flat-field, and restore the normalization. The commands you would type are:

```
GO: MN 2
GO: DIV 1 2 FLAT
```

The `FLAT` keyword of the `DIV` command uses the image mean of Buffer 2 (the flat-field frame) calculated with the `MN` command to normalize the flattened image. The

result of the above 2 commands is to leave a flattened program image in Buffer 1. If the flat-field frame in Buffer 2 is to be used to flatten many images, then so long as this flat-field always resides in Buffer 2, the first step of taking the image mean need only be done once in a session. If you are ever unsure of this, issue the `MN` command on the buffer with the flat-field frame. If no image mean exists for the flat-field frame's buffer, the `DIV` command will issue an error and stop.

Flat-field correction should be last step before further reduction of the program image. It must follow removal of effects due to image readout or intrinsic data biases which are (usually) independent of the light falling on the chip. Corrections for these effects are discussed below.

## 3.2 Dark Bias

### 3.2.1 Basic Principles

Light falling into a pixel produces electron-hole pairs in the semiconductor layer proportional to the number of photons striking the pixel. It is these electrons which are accumulated as the "counts" in a pixel. However, some pixels make electron-hole pairs all by themselves, even when it is dark. This "Dark Bias" is primarily thermal in origin (thermal giggling of the atoms in the semiconductor can make electron-hole pairs as well), thus the reason that CCDs are operated at cryogenic temperatures. Since the CCD temperature is held steady throughout a night (presumably), the number of electrons/second produced by a given pixel should be constant in time, and thus the spurious dark bias in a pixel should scale linearly with exposure time. For most pixels, the dark bias is negligible, but some individual pixels (or clusters of pixels) are "hot," producing many electrons/second. A few of these "hot pixels" behave non-linearly, so that the actual dark bias in them scales only roughly with exposure time. The dark bias enters into the image as an additive constant ("bias") which is a function of exposure time. To correct for it, one must somehow subtract the bias from the frame.

The dark bias is a function both of location on the CCD, as the dark count rate varies seemingly arbitrarily from pixel to pixel, and a function of exposre time. Thus it is necessary to somehow create a "Dark Bias Frame" which will be subtracted from the image to remove the dark bias. There are a number of schemes for doing this, depending on the details of the CCD chip that you are using. In particular, it will depend on its "non-signal" bias characteristics. Typically, there are two non-signal bias components to most CCDs we've had experience with. The

first is a time-dependent dark bias, linear with time in all but a handful of very hot pixels. The second is a time-independent bias often called "fixed-pattern noise". This latter is treated in more detail below. Both are additive contributions.

Two basic ways of removing dark bias from an image will be presented. It must be emphasized that these are not exclusive of other methods that might also be employed to this end.

### 3.2.2 Matched Exposure Dark Frames

This method is most useful if you are using only a few exposure times during a run, for example, you are making an imaging survey and always use 15 minute exposures. In this case, you would take 2 or more dark frames of the same exposure time as your program exposures during your run. Since the dark bias is a function of CCD temperature, you should be careful to bracket all CCD temperature changes with dark frames. A "dark frame" is one taken with the shutter closed and (if possible) the dome as dark as possible.

With long exposures, the dark frames you take will be contaminated with ion hits. This is particularly true of thick CCD chips like the GEC P8600 or the unthinned Tektronix CCDs. For this reason, you want to take at least 2 dark frames of the same exposure time (and same CCD temperature). The trick to removing the ion hits is as follows:

Load the first dark frame into Buffer 1, and the second into Buffer 2. Make a copy of the dark frame in Buffer 1 in Buffer 3 using the command:

```
GO: COPY 3 1
```

Then, subtract the dark frame in Buffer 2 from the dark in Buffer 3:

```
GO: SUB 3 2
```

The ion hits on the dark frame in Buffer 2 will make large negative spikes, and the ion hits on the dark frame in Buffer 1 will be left as large positive spikes. We will now make 2 frames: 1 containing only the residual ion hits for the dark frame in Buffer 1, and the other to contain the residual ion hits for the dark frame in Buffer 2. Consider all signal below 10 counts to be worth ignoring. Create the residual frames as follows:

```
GO: COPY 4 3
```

```
GO: CLIP 3 MIN=10.  VMIN=0.
GO: CLIP 4 MAX=-10.  VMAX=0.
```

Buffer 3 now contains the residual ion hits on the dark frame in Buffer 1, and Buffer 4 contains the *negative* of the residual ion hits on the dark frame in Buffer 2. Remove the residual ion hits from each of the original dark frames with the following commands:

```
GO: SUB 1 3
GO: ADD 2 4
```

Note that since the residual counts in Buffer 4 are *negative* we must *add* them to the raw counts in Buffer 2 rather than subtract as we did to the raw counts in Buffer 1. Now, make an average dark bias frame by averaging the dark frames in Buffers 1 and 2:

```
GO: ADD 1 2
GO: DIV 1 CONST=2.0
```

The frame in Buffer 1 is now a "Mean Dark Bias Frame" for images with that exposure time. Change the header of this file to reflect its new status, write it to disk, and clean up the leftover debris:

```
GO: CH 1 'Mean Dark Bias Frame'
GO: WD 1 DARKBIAS.CCD
GO: DISPOSE 2 3 4
```

The actual correction for the image dark bias is quite simple. It must be done after baseline correction (if you are using Lick CCD data) and *before* flat-field correction. Simply subtract the mean dark bias frame from an image frame of the *same* exposure time. This latter point is repeated because dark bias is time-dependent. In practice, this technique vastly improves the appearance of long-exposure images with high dark-bias CCDs. For non-linear hot pixels, this corrects for most of the hot pixel dark bias, but the remaining counts will have to be accepted as beyond correction.

### 3.2.3 Scaled Dark Frames

This method is most useful if you are using many different exposure times during a run. In outline, one takes a "fiducial" dark frame which is long enough that it will contain significant dark bias (you want enough signal to get away from the readout noise). Typically, you would choose a dark frame which has the same exposure time as the shortest exposure time images for which dark bias is important. Unless you have a frightfully dark-noisy CCD chip, dark bias will usually be insignificant on short (less than 1 minute) exposures. Then, this dark frame is multiplied by a constant so that it has the same equivalent exposure time as the image from which the dark bias is to be removed. This constant is the just the program image exposure time divided by the dark frame exposure time. You want this constant to be small (not more than 5 or 6), otherwise the extrapolation error could be very large. Then the scaled dark frame is subtracted from the program image.

In outline, simple; in practice, complicated by small details. Simply scaling the fiducial dark frame will also scale up the constant fixed pattern noise, thus over-correcting the program image by the scaling factor. In addition, ion hits must first be removed from the fiducial dark, otherwise those will be scaled up and cause large negative spikes on the program image. The following seems to be a sensible procedure for Lick CCDs, but there are most likely others that more imaginative folks can come up with.

**Step 1:** While on the mountain, take a number (at least 2) fiducial dark frames, followed *immediately* by 1 second dark frames. The latter 1 second frames will contain only (more or less) the time-independent fixed pattern noise of the CCD. (We're getting ahead of the game, the fine details of how we deal with fixed pattern noise is covered in the next section. We'll use the techniques without explanation here.)

**Step 2:** Back home, prepare a fiducial dark frame as follows: You will repeat this procedure for each fiducial dark frame. The idea is the same as the "Matched Exposure" technique described above, you want to use multiple dark frames to eliminate (as much as possible) contamination of the dark frame by ion hits. Read one of the fiducial dark frames into Buffer 1, and its companion 1 second dark frame into Buffer 2. Remove the fixed pattern noise with the following procedure:

```
GO: MASH 10 2 SP=(SR[2],NR[2]-SR[2]+1) NORM
GO: STRETCH 3 10 SIZE=NR[2] HORIZ
GO: SUB 1 3
GO: DISPOSE 3 10
```

This example assumes explicitly that the fixed pattern noise runs along image columns and is essentially the same row-by-row. This is almost always the case, as fixed pattern noise is introduced during readout, and readout is row-by-row. When this procedure is completed, the image in Buffer 1 will have the fiducial dark frame with the fixed pattern noise removed, and the image in Buffer 2 is the untouched 1 second companion dark frame. Save the fiducial dark in Buffer 1 in some safe Buffer (or in a temporary disk file), and repeat the procedure for all of the fiducial dark frames. Then, prepare a fiducial dark bias frame following the techniques described in the section on "Matched Exposure Dark Frames" above for creating a "mean dark bias frame". This is why you need at least 2 fiducial dark frames and their companion darks to use this method.

Once you have generated a "mean fiducial dark frame", it is used as follows. Suppose the fiducial dark frame is in Buffer 1, and has an exposure time of 10 minutes. The program image frame to be dark bias corrected is in Buffer 2, and has a 45 minute exposure. To correct the program image, you would type the commands:

```
GO: MUL 1 CONST=45./10.
GO: SUB 2 1
GO: DIV 1 CONST=45./10.
```

The first command scales the fiducial dark up to the exposure time of the program image. The second command subtracts the scaled dark bias from the program image, and the third command removes the scale factor from the fiducial dark, restoring it to its original exposure time.

This method does not remove the fixed pattern noise from the program image, like the "matched exposure" method. Usually, the fixed pattern noise is negligible compared to the sky level in program images. Where it is *not* negligible, it must be removed from the program image following the methods described in the section below.

A final note about dark bias correction. Implicit in all of this is the assumption that the dark counts in a given pixel scale linearly with time. This may not be exactly true, but most hot pixels seem to behave linearly on average. Thus, one should avoid large extrapolations between fiducial dark bias frames and the program images to avoid over/under-correction of individual pixels. In addition, recall that any procedure that changes the image introduces noise which must be taken into account later when discussion true uncertainties of measurements made for corrected imaging data. We will not be discussion error propagation in this cookbook, but want to remind you that it is an issue, one often poorly appreciated.

## 3.3 Fixed Pattern Noise

### 3.3.1 Basic Principles

Fixed pattern noise (as it is often called) is any noise which is impressed on the image data during readout of the CCD. It usually takes the form of a background pattern across the chip, repeated from row-to-row (hence the description "fixed"). One common cause is 60 Hz pickup on the data transmission cables from the dome to the data taking computer. On the old GEC P8600 CCD used at the 1-meter Nickel Telescope on Mt. Hamilton, this used to be a real problem, with the amplitude of the fixed pattern noise $\pm$ 5 counts or more.

The exact character of fixed pattern noise changes from device to device. At Lick, by sychronizing the readout clock triggers to the 60 Hz power lines, the pattern is the same from row to row. Before this was done, the pattern shifted in phase as the readout clock triggers beat against the 60 Hz line, causing an awful "herring-bone" pattern. Since the pattern repeated on each row, an average over 576 rows (on the old GEC CCD) meant a very high signal-to-noise representation of the pattern which could be removed.

A common feature of fixed pattern noise is that since it is imposed at readout time, it is fixed amplitude, and independent of exposure time. It then enters as a (hopefully) small additive bias to the image data, and thus may be removed simply by subtracting a suitably defined "fixed pattern frame" from the data.

### 3.3.2 A Simple Example

The following is an example of how you would correct a program image for fixed pattern noise. This example will assume that the fixed pattern repeats row-to-row (how it works at Lick). To remove the fixed pattern noise, along with our program image, there is a short (1 second) dark frame taken immediately after the program image which contains only the fixed pattern noise (plus readout noise, assume dark bias in 1 second is negligible). Because the amplitude of fixed pattern noise is usually close to the readout noise of the system, we want our removal of the fixed pattern to introduce as little noise as possible into the program image.

To make a high signal-to-noise template of the fixed pattern, we will average each row of the image by mashing the fixed pattern image into a 1-D image which contains just the mean fixed pattern. Then this 1-D fixed pattern image will be stretched back out into a full size 2-D image, and subtracted from the program image. For the following, the program image is in Buffer 1 and starts on

(Row,Column)=(0,0). The 1-second dark frame is in Buffer 2 and likewise starts on (0,0). Buffers 3 and 10 will be used as working space. The basic VISTA procedure is as follows:

```
GO: MASH 10 2 SP=(0,NR[2]) NORM
GO: STRETCH 3 10 SIZE=NR[2] HORIZ
GO: SUB 1 3
```

Buffer 1 now contains the program image corrected for fixed pattern noise. Buffer 3 contains a normalized fixed pattern frame, and Buffer 10 contains the template pattern, derived from the average (`NORM` keyword of the `MASH` command) of all of the rows of the 1-second dark frame in Buffer 2.

### 3.3.3 Refinements

One problem is that the pattern is sometimes unstable over a night, and thus for each image you would have to obtain a companion fixed pattern frame (typically defined as a 1 second dark exposure, since the pattern is exposure time independent). Depending on what you wish to achieve, this could pose a real problem, and you must decide if the overhead of a fixed pattern frame (which is readout time dominated) is worth the trouble.

In most cases, fixed pattern noise is small amplitude, a few "counts" peak-to-peak. If your mean background (sky, etc.) is many orders larger than this, the contribution from fixed pattern noise will be negligible and thus may be safely neglected in detail. This is often the case with the TI 500×500 CCD used for imaging on the 1-meter Nickel telescope. It is probably always negligible in full-frame, high-signal flat fields.

However, in cases where the background is small, fixed pattern noise will be a problem. For example, following the "Scaled Dark Frame" method for removing dark bias described above, neglecting the fixed pattern noise on the the fiducial dark frame can lead to real problems with the dark bias subtraction. It is also a consideration for the low background present on Echelle images (see the chapter on Echelle reductions), where it make a non-trivial contribution to inter-order scattered light.

If you are lucky enough to ignore fixed pattern noise in detail, you should not forget that it is there. For example, if you are dividing images well separated in time, large, beating between out-of-phase fixed patterns could produce undesirable

artifacts. Always beware of features that might be artifacts of the interaction of fixed pattern noise and image data. How sensitive your measurements will be to fixed pattern noise depends on what information you are trying to get out of your images.

## 3.4 Baseline Restoration

**Warning:** This procedure is relevant for Lick Observatory CCD images only, and may produce very peculiar results if applied blindly to images from other telescopes.

The last column of CCD images taken at Lick Observatory contains information on changes in the reference voltage during image readout. This reference voltage corresponds to zero signal from the CCD; the "CCD Baseline." During readout, noise is introduced due to digital round-off errors when the signal from the CCD pixels (a voltage) is converted to a digital "count" and stored in the data taking computer's memory. To correct for this, a running average of the baseline counts during readout of a given row is subtracted from each pixel in that row, and stored in the last column of the image. This allows for correction of slow drifts in the baseline voltage during an image readout after the fact. Thus, before further image processing, it is necessary to "restore" the image baseline.

In VISTA, this is done using the BL command. For example, to restore the baseline in a raw image read off a tape into buffer 1, you would type:

```
GO: BL 1
```

VISTA will then print the mean baseline level (in digital counts; DN), and the slope (in units of DN/pixel), giving an idea of the amount of baseline drift during readout. If there are problems with the CCD readout electronics, they will often show up as peculiarities in the baseline data. Large jumps or spike in the baseline value usually signal troubles, thus the baseline serves as a good "quick look" diagnostic of the instrument performance. It is prudent to make a habit of looking at the image baselines throughout the night.

One further point of caution. The baseline column is then replaced with the mean baseline computed, so running baseline on an image a second time could produce very strange results.

# Chapter 4

# 2-D Image Reduction

The goal of this chapter is to outline the basic procedures for reducing raw CCD images. Image "reduction" is distinct from image "analysis," which is actual measurement of some property from the images. Reduction is the set of procedures used to take a raw image from the instrument and prepare it for analysis. Throughout this section, it will be assumed that the reader has already read over the material in the basic techniques section (Chapter 3), particularly regarding flat-fields, dark counts, and baseline restoration.

We shall begin with basic procedures which should be common (in some for or another) to all CCD chips. From there, secondary reduction procedures are described to deal with effects which depend strongly on what instrument has been used, and what measurements you ultimately wish to extract from your data. Finally, a set of advanced techniques which range from somewhat tricky to real "black-belt" level are discussed.

## 4.1 Basic Image Reduction

A few basic corrections must be applied to all images before further reduction can proceed. In what follows, a "raw" image refers to the data values off the tape from the mountain data taking system. The goal of this section is to produce a "reduced" image, containing (to a first approximation) an image of what is in the sky with most of the major instrumental effects removed.

Some basic reduction procedures are strongly dependent on the equipment used. A TI CCD image taken at Lick Observatory will require a few different steps than, for example, an RCA CCD image taken at Cerro Tololo. This cookbook will

describe procedures for reducing CCD images from Lick in particular, although most of the techniques, and the VISTA commands required, should in principle be common to all CCD data.

To perform a basic image reduction, you need the following pieces:

1. **Raw CCD Image**, read into a VISTA buffer from tape or disk using either `RT` or `RD`.
2. A pre-prepared **Dark Frame** with an exposure time the same as the raw image, read into a second VISTA buffer from tape or disk (see §2.X).
3. An appropriate **Flat-Field Frame**, prepared as described in §2.X, and read into a third VISTA buffer, and with its mean computed and stored.

In the following example, we start with:

1. A raw CCD image, stored on TAPE as image number 10. The tape is on UNIT 0, and has been mounted (`MOUNT UNIT=0`).
2. A pre-prepared Dark Frame, stored on DISK with the name: `dark.ccd`
3. A pre-prepared Flat-Field Frame, stored on DISK with the name: `flat.ccd`

Read the Dark Frame into buffer 2, the Flat-Field Frame into buffer 3, and compute the mean of the Flat-Field:

```
GO: RD 2 dark.ccd
GO: RD 3 flat.ccd
GO: MN 3 NOBL
```

Buffer 2 now contains the dark frame, and buffer 3 contains the flat-field frame, with its mean intensity computed and stored in the VISTA variable `M3`. The `NOBL` keyword in the `MN` command tells VISTA to ignore the baseline column when computing the image mean.

Read the raw image from the tape into buffer 1 and restore the digital baseline:

```
GO: RT 1 10 UNIT=0 NOMEAN
GO: BL 1
```

Remove the dark counts:

```
GO: SUB 1 2
```

Finally, flat-field correct the image:

```
GO: DIV 1 3 FLAT
```

The corrected and flattened image is now in buffer 1, and is ready for further processing. It is advisable to store this image in some non-fragile way (either on disk, or a working scratch tape) before proceeding. As can be seen, once dark and flat-field frames have been prepared for a night (or a run), the basic reduction of an entire night's data may proceed almost automatically, as little or no image interaction is required at this stage. For a large number of images taken through one or two filters, running VISTA as a batch process can reduce the tedium tremendously.

## 4.2 Secondary Image Reduction

After the basic reduction step, there are a number of secondary steps that may be required before beginning analysis of the images. Which of these steps are taken depends on the analysis you wish to pursue, and the peculiarities of the instrument you are using. In that sense, these procedures are optional. The procedures in this section are listed in approximate order of complexity. The basic format is to present the simplest way to perform each procedure, then adding increasing detail. These are by no means the *only* way to do things, but should serve more as a guide for what the commands do, and some of the general considerations.

### 4.2.1 Windowing

Windowing is image reduction in the most literal sense of the word. It is the process of removing from an image only a small subset for further analysis. Often, this is necessary because vignetting or an aperture plate make it so that some regions contain no data. A number of CCD chips have bad regions around their periphery which are useless and may even be a nuisance later on. In other cases, you may simply wish to break up a large image into small, easily digested subsets.

Windowing in VISTA may be accomplished in one of two ways: either using a combination of the `WINDOW` and `BOX` commands, or the `COPY` and `BOX` commands. In either method, the `BOX` command is used to define the image sub-region to be isolated using either the `WINDOW` or `COPY` commands. The difference is that with `WINDOW`, the regions outside the specified `BOX` are permanently lost, while with `COPY`

the original image may be left untouched, and a new image containing the contents of the specified BOX is created in a different buffer.

For example: You have an 800×800 TI CCD image in buffer 2, of which you wish to use a region 150 columns wide by 250 rows high starting at pixel (R,C) = (127,334). Let's define BOX 4 to surround the region of interest, and throw away all of the image regions lying outside of it. To do this, you would type:

```
GO: BOX 4 SR=127 SC=334 NR=250 NC=150
GO: WINDOW 2 BOX=4
```

The former 800×800 image has now been reduced to a 250×150 (rows×columns) image, and the rest has been removed from memory. It is a very good idea to store the whole image, either in another buffer, or on tape or disk, just in case you screw up. The WINDOW command is PERMANENT!

Suppose instead that you wish to keep the 800×800 image in buffer 2 as is, and copy the region of interest (same as before) into a different buffer, say buffer 9. This way, you could break off pieces of the larger image for individual attention. Remember that smaller images make for faster processing for routines that use the entire image. To do this, you would type:

```
GO: BOX 4 SR=127 SC=334 NR=250 NC=150
GO: COPY 9 2 BOX=4
```

Now buffer 9 contains a 250×150 image which is the desired subset of the larger image, which remains untouched in buffer 2.

One final remark: COPY 2 2 BOX=4 is equivalent to WINDOW 2 BOX=4. If you copy into a buffer a subset of itself (defined by BOX=4), then you will destroy the current image and replace it by the subset. This is essentially what WINDOW is doing.

### 4.2.2 Image Position and Orientation

Often it is desirable to orient the image in a fashion which is logical. For example, an image orientation of North–South running vertically with North at the top, and East–West running horizontally with East at the left, is the conventional orientation of astronomical images. Optics or details of the CCD readout often introduce reflections or rotations which may be removed using simple commands. All of these come under the broad heading of commands effecting the image coordinates.

The simplest image coordinate change is to shift (or translate) the image to a new origin. VISTA provides the SHIFT command for this purpose. SHIFT works very rapidly for shifts by an integral number of pixels in either dimension. For example, an image in buffer 2 has its origin (upper left hand corner) at (R,C)=(345, 23), and you want to translate it so that the new origin is (R,C)=(0, 0). To do this, you would type:

```
GO: SHIFT 2 DR=-345 DC=-23
```

where DR= is the shift in rows, and DC= is the shift in columns. You can find out the origin of an image by issuing the BUFFER command (see § x.x). Another way to do the same thing as above is to use the internal VISTA variables that contain the image origin, thus you could have typed instead:

```
GO: SHIFT 2 DR=-SR[2] DC=-SC[2]
```

The variables SR[i] and SC[i] contain the starting row and column of image i respectively. More complicated applications of SHIFT will be discussed in the Advanced Techniques section below, in the context of the problem of image registration.

The orientation of the image may be changed by either reflection about a given axis (the FLIP command), or rotation about the image center (the ROTATE command). For example, you have an image in buffer 4 which has an orientation which is East–West along the horizontal (columns) axis, with West on the left. To "flip" this orientation so it appears with East on the left, you would issue the command:

```
GO: FLIP 4 COLS
```

This has the effect of reflecting (flipping) the image about its central column. There is an analogous keyword, ROWS, which will reflect the image about the central row.

A more common orientation problem is an image which is turned on one side. For example, the TI 500×500 CCD on the 1-meter Nickel Telescope at Mount Hamilton in its "CCD-D" configuration is mounted such that the images come out with an orientation of North–South running horizontally (columns) with North to the left, and with East–West running vertically (rows) with West at the top. To get it into the conventional "North Up, East Left" orientation, you want to turn it clockwise on its side. Suppose you have such an image in buffer 1, then you would put it into the desired orientation by typing the command:

```
GO: ROTATE 1 RIGHT
```

The RIGHT keyword tells it to rotate the image clockwise by 90°. Similarly, you may restore the old orientation by issuing the command:

```
GO: ROTATE 1 LEFT
```

which will rotate the image *counterclockwise* through 90°. To now turn the image upside down — a rotation by 180° about the image center, you would issue the command:

```
GO: ROTATE 1 UD
```

Note that this is *not* the same as a FLIP 1 ROWS.

For rotations through 90 and 180°, ROTATE works very rapidly (it is only swapping array index pointers). It is possible to rotate through an arbitrary angle, but this internally involves an interpolation and re-binning of the image and thus may be very time consuming, especially for large images. For example, suppose you wish to rotate an image in buffer 3 through an angle of 45°. You would type in:

```
GO: ROTATE 3 PA=45.
```

and wait...

The sign convention for the PA= keyword is: (+) for counterclockwise, (−) for clockwise. It follows the usual astronomical convention of position angles rotation from North counterclockwise towards East.

### 4.2.3 Cosmic Ray Events

CCD detectors are not only great detectors of optical photons, but also of any ionizing radiation energetic enough to knock electrons out of the silicon as it passes through. Cosmic rays, $\gamma$-rays, and X-rays may all be detected by a CCD. In many cases, these high-energy "events" (or "ion hits" as they are sometimes called) create hundreds or thousands of electrons, making the counts in a single pixel jump tremendously. In thick devices, the probability of a passing cosmic ray creating spurious signal is greatly enhanced, so long exposures (30 minutes or greater) with thick chips are often peppered with bright patches. VISTA offers two ways to remove cosmic ray events from an image: ZAP and TVZAP.

The first option, ZAP, is blind. It passes a square filter of a specified width (in rows and columns) over the image. At each pixel, it computes the median image intensity in the specified filter box centered on that pixel. If counts in that pixel deviate from the median by more than a specified threshold value (expressed as a multiple of the standard deviation of the pixels inside the filter), it is set equal to that median value. Since cosmic ray events are usually dramatic deviations from neighboring pixels, they can be removed in this way.

For example, you wish to ZAP an entire image stored in buffer 1, specifying a $5 \times 5$ pixel square filter, and a rejection threshold of 3 standard deviations. You would type:

```
GO: ZAP 1 SIG=3 SIZE=5,5
```

and wait for the routine to finish. The larger the image, the longer it takes. At the end, VISTA will printout how many pixels were "zapped". ZAP is particularly convenient for batch processing (provided it is safe to use on the data) as it require no interaction. It is possible to monitor the progress of ZAP by appending the TTY keyword onto the end of the command line. This will print on the terminal the locations of the pixels zapped. However, for large images, bear in mind that this could significantly increase run time.

There is a significant danger of "zapping" data away, especially if the images are oversampled. An alternative procedure is to *interactively* remove cosmic ray events from the image using TVZAP. The TVZAP command works with the image display cursor. Like ZAP, the user specifies the filter size and the rejection level via keywords. However, TVZAP allows you to change both in mid-stream as required to remove the offending pixels.

For example, to interactively "zap" the image above, you would first display the image using the TV command, then type:

```
GO: TVZAP SIZE=5,5 SIG=3
```

Place the cursor on the offending pixel, and hit the Z key. VISTA will respond by printing out how many pixels in the present filter were "zapped." In some installations (*e.g.,* those for an AED 1024 color display), the "zapped" pixels are updated immediately on the screen, so you can see the effects of the "zap". In others, TVZAP is "blind", and it will be necessary to re-display the image with TV to see the results. An alternative is simply to keep hitting the Z until VISTA responds with `0 pixels zapped`. While not necessarily "better" than ZAP, it is "safer" in some cases. Because it is interactive, it cannot be used for batch processing.

### 4.2.4 Coping with Bad CCD Regions

Many CCDs have bad image columns which appear as dark streaks of "missing data", often with one end very bright. These are columns along which charge cannot be transferred because of a so-called "bulk state trap". The bright pixel which often caps a bad column is the trap where the charge piles up (thus appearing anomalously bright). Sometimes, many adjacent columns may be affected by bulk state traps, so bad columns may be many (4 or more) columns wide. Image data falling across these regions is lost. In addition, there may be regions of reduced or dead sensitivity (either really dead or due to foreign matter on the chip surface) that cannot be removed by flat-fielding, or "hot spots" which are regions of anomalous and/or non-linear dark current which the dark frame cannot entirely remove (see above). Also, on many chips, the edges of the chip are usually really bad, and data there useless.

For the most part, bad regions should be left alone so that there will be no danger of making an erroneous measurement from those regions of the image. However, there are some things that may be done to avoid possible numerical problems, as bad regions can sometimes contain extreme data values which could screw up measurements. The three basic procedures are: masking, clipping, and interpolating.

Masking

A few VISTA commands (`ABX`, `MN`, `INTERP`, `MASH`, `SURFACE`, and `AUTOMARK`) allow the use of an image "mask." A mask is a map of image regions to be ignored by the routines, and which may be stored on disk for later use. There are two ways to generate an image mask. The first is using the `MASK` command, and the second is via the `CLIP` command. Clip will be discussed separately below. With `MASK`, you can specify entire rows or columns, boxes containing entire regions (say a box surrounding a saturated star image), or single pixels to be included in the image mask. For example, to create a mask which flags bad columns 40 thru 42, 65, and 365, the contents of `BOX 1` which surrounds a bad region on the chip, and a bad pixel at (R,C) = (127,359), you would issue the following commands:

```
GO: MASK C=40,42
GO: MASK C=65
GO: MASK C=365
GO: MASK BOX=1
GO: MASK PIX=127,359
```

Note that MASK is *cumulative*, that is, each time it is invoked, something is added to the current mask. Thus it is possible to build up a complicated mask as required.

Since VISTA only allows one mask at a time, you can make many different masks and store them on disk, to be read in when needed. For example, to store the above mask on disk as the file `mymask.msk`, you would type the command:

```
GO: SAVE MASK=mymask
```

Note that by default, the extension `.msk` is appended to the file name. The default directory and extension may be changed by using the SETDIR command (see § X.X). This mask may then be read back in by typing the command:

```
GO: GET MASK=mymask
```

When a mask is read in using GET, the current mask is lost.

You can remove part (or all) of the current mask using the UNMASK command. For example, with the above mask, you later realize that the cosmic ray at pixel (127,359) only appears on one image, so you could just TVZAP it away and ignore it. To remove this region from the current mask, you would type:

```
GO: UNMASK PIX=127,359
```

Finally, to completely eliminate the current mask, simply typing UNMASK without any keywords, and the current mask will be erased.

Interpolation

Interpolation is the process of replacing the data in a bad region (*e.g.,* the "missing" data in a blocked column) with an estimate of what should be there based on interpolation data values in surrounding regions. This will usually produce acceptable results in the case of a bad column crossing a region containing sky or a diffuse region, but erroneous results if a star lands on the bad column. Thus, removing bad regions by interpolation is mostly cosmetic, but it can be valuable if there is danger of biasing a measurement in an image region because of extreme data values in the bad column. This is probably the best way of coping with bad regions for those VISTA routines that do not recognize image masks (see above), or for which clipping (see below) is inappropriate (or too drastic).

Most often, interpolation is used to smooth over bad columns. For example, in an image in buffer 3 there is a column 254 is blocked starting at row 127 and continuing all the way to the bottom of the image at row 799. By using the two

"good" columns on either side of column 254, to interpolate this bad column away requires 2 steps. First define a BOX which contains the bad column, then you issue the INTERP command. You would type:

```
GO: BOX 5 SR=127 SC=254 NC=1 NR=799-127+1
GO: INTERP 3 BOX=5 COL AVE=2
```

The first half defines the BOX 5 to contain the bad column. This box is 1 column wide starting on column 254, and also starting on row 127 and going to row 799. Note that for the NR= we are having VISTA compute the number of rows from 127 to 799. Since this must include row 799, we add 1 to (799−127). The second half uses the INTERP command to tell VISTA to interpolate across the region of the image in buffer 3 contained in BOX=5 such that for each row inside BOX=5, use the data in two pixels either side (AVE=2) of the box.

If there are many bad regions on a given image (many CCD chips are usually chock full of bad columns and such), you can use an image mask (see above) to interpolate across many regions at once. For example: you've got data from Fernwood Observatory's MGA #2 CCD chip which has 127 bad columns. Previously, you've made a mask of these bad columns, which is stored on disk in the file mga2badcol.msk. To use this mask to interpolate across all the bad columns in the image in buffer 1 using 3 pixels either side of the offending columns you would type:

```
GO: GET MASK=mga2badcol
GO: INTERP 1 COL AVE=3 MASK
```

And all of the bad columns will be interpolated out.

Clipping

Masking is asking some VISTA commands to simply ignore regions. Interpolation is smoothing them over by using surrounding data. Clipping is major surgery. Clipping is the process of setting all pixel values in a specified region equal to a constant value if they are above (or below) some fiducial value. For example, if the basic reduction (dark and flat-field correction) causes bad regions to have extreme values, it is possible to clip all these pixels to some neutral value (say, zero). This is done in VISTA with the CLIP command. Needless to say, clipping is a rather drastic procedure, so it should only be done with great care.

For example, after dark and flat-field correction, there are regions of the an image in buffer 3 which contain large negative numbers. Since negative counts are "unphysical" for this image, you wish to have VISTA ignore all negative pixels. One way is to set all negative pixels equal to zero. To do this you would type:

```
GO: CLIP 3 MIN=0.  VMIN=0.
```

This sets all pixels with values less than `MIN=0.` to have the value `VMIN=0.`. Some VISTA commands (*e.g.*, `MN`, and `SURFACE`) have keywords that will explicitly ignore pixels with zero value, so this may be useful.

Another way is to put the negative regions into the current mask (see above). This would be done by typing:

```
GO: CLIP 3 MIN=0.  VMIN=0.  MASK
```

Thus, all negative pixels are "clipped" and set to zero (0.) as above, and their positions are loaded into the current mask. It is possible to make a mask using `CLIP`, but not actually change the data values by typing instead:

```
GO: CLIP 3 MIN=0.  VMIN=0.  MASKONLY
```

In this case, the negative pixels are left untouched, but their locations are loaded into the current image mask.

`CLIP` may also be used to suppress extremely large values. For example, you know that pixels with more than 30,000 counts in the image in buffer 4 are due to saturation, so you wish to ignore everything above 30,000. To set all pixels with more than 30,000 counts to a fiducial value of (say) 500., then you would type:

```
GO: CLIP 4 MAX=30000.  VMAX=500.
```

Similarly, these pixels may also be put in the current image mask using the `MASK` and `MASKONLY` keywords as above. Note that you can clip pixels above or below the thresholds set by the `MIN=` and `MAX=` keywords simultaneously. Thus, in the above example, to clip all negative pixels to zero (0.) at the same time, you would have typed:

```
GO: CLIP 4 MAX=30000.  VMAX=500.  MIN=0.  VMIN=0.
```

which is a more efficient use of computer time as you scan over the image only once.

Clipping an entire image may be dangerous, as you might accidentally punch out valid data points. In addition, if the offending pixels are confined to a small region, it is very wasteful of computer time to have `CLIP` scan over an entire image (for example, if the bad spot is only a 10×10 pixel region on a 500×500 CCD

image, the times savings is roughly 2500, although additional execution overhead will not make it quite that fast). You can restrict the limits of the CLIP to only those pixels within a given BOX. For example: a saturated star image centered on pixel (250,303) of an image in buffer 2 blows away a region roughly 15 pixels across. You want to clip out all pixels in this region greater than 20,000 counts and set them to zero (0.). You first create a box isolating the offending star (call it BOX 1) and then clip the contents of the box. To do this type:

```
GO: BOX 1 CR=250 CC=303 NR=15 NC=15
GO: CLIP 2 BOX=1 MAX=20000.  VMAX=0.
```

When the image in buffer 2 is redisplayed, the central pixels of the saturated star image should have all zeros in it. The combination of CLIP and BOX for handling bad regions of an image is rather powerful and finds a fair amount of use in most image processing applications.

### 4.2.5 Sky Subtraction

In some sense, the subject of sky subtraction belongs under the heading of "image analysis" rather than "image reduction", as it is often a crucially important step in such problems as stellar photometry (both artificial aperture and profile fitting techniques) and surface photometry of extended objects. However, in other applications, it may not be as critical a step. How you handle sky subtraction very much depends on the quality of your images, the instrumental peculiarities, and your ultimate analysis goals. What shall be presented here are some of the basic considerations and techniques, and how to use VISTA for this end.

The field of view of most CCD cameras is such that after correction for the flat-field, there should be a constant background level due to the brightness of the night sky. Thus, to remove the sky, one simply estimates the sky level in some way, then subtracts that value from every pixel in the image. There are, however, some complications that arise. Deviations from constancy are due primarily to scattered light in the camera — particularly if there are bright, saturated stars in the field, and optical effects like vignetting in a focal reducing camera (due to the field being imaged overfilling the collimating lens). It is also more difficult to estimate the sky level if the image is of an extended object (say an emission nebula or galaxy) which fills most of the field.

Another complication is that the sky may not be "flat" after all, and so simply subtracting a constant background will not produce acceptable results. In these

cases, it may be more appropriate to fit a two-dimensional surface (either a plane or curved polynomial surface) to the sky and subtract that from the image. These techniques are very involved, and dependent on the type of data and the instrument, and so will not be discussed here. Instead, we shall assume that the sky background is well described by a constant value everywhere on the CCD. Even if there are optical problems which cause deviation from flatness, let us assume that it is possible to window down the image (see § 4.2.1 above) to a region where it is flat.

In a frame full of stars, or of an extended object that fills a fraction of the field, most of the pixels contain sky (or close to sky). To estimate the sky level, there are basically two schools of thought. One is that the sky value should be the most common value in the image, which is defined as the *mode* of the image intensity distribution. Another is that the image *mean* in regions away from the object should be used, although occasionally the *median* is used by some workers instead of the mean.

The former is useful if it is difficult to isolate regions which are "sky" and which are "object(s)", for example, an image of a star field. The latter is useful only in those cases where "sky" and "object" may be readily isolated, say, an image of a single star. A representative histogram of the distribution of single-pixel intensities in an image is shown in Figure 4.1. The mode is simply the peak of this distribution (indicated by the arrow). The arithmetic mean and the median are also indicated. Note that both the mean and median are affected by pixels in the high-intensity tail of the distribution, thus a large number of stars on the image, or inability to isolate "sky" and "object(s)" will lead to an over-estimate of the sky value. For most applications, deriving a modal sky value is the most appropriate estimate of the sky level to use.

VISTA computes the mode of an image with the `SKY` command. For example, suppose you wish to estimate the modal sky value of an image in buffer 2. To compute the sky level of the entire image, you would type:

```
GO: SKY 2
```

The mode and its uncertainty (defined as the full-width at half-maximum of the image intensity distribution) will be printed on the terminal, and stored in the VISTA variables `SKY` and `SKYSIG` respectively. These variables allow you to conveniently use this data for sky subtraction. For example, typing the commands:

```
GO: SKY 2
GO: SUB 2 CONST=SKY
```

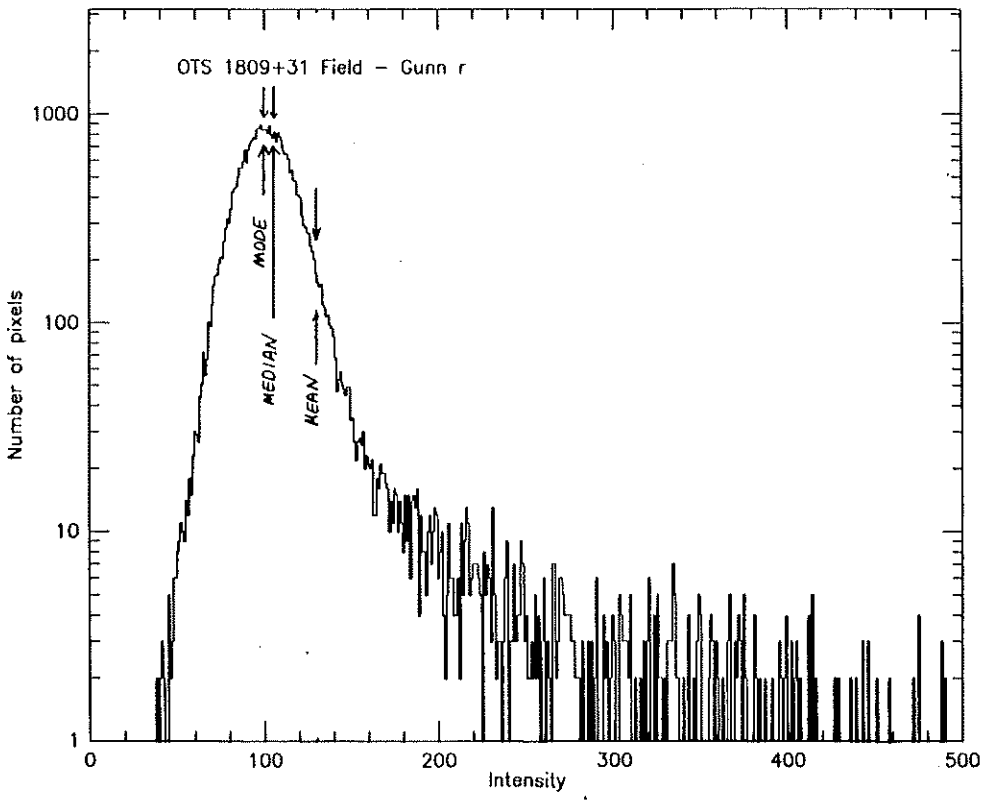



**Figure 4.1**: Representative image intensity distribution for a 500×500 CCD image of a star field. The x-axis plots the intensity value in digital counts, and the y-axis plots the number of pixels with that intensity. The Mode, Median, and Mean of the distribution are labelled with arrows.

will compute the value of the sky level in the image in buffer 2, then subtract that value from the image, leaving the sky-subtracted image in buffer 2.

Consider now the case where there is an extended object, say a galaxy, in the center of the CCD image, and sky and foreground stars in the periphery. There are two ways to estimate the sky level in this image without including the regions containing the extended galaxy image. The first is to simply compute the modal sky value by restricting the regions of interest to the four corners of the image. You would do this by typing:

```
GO: SKY 2 CORNERS
```

The `SKY` command will then print the coordinates of the four corner regions it has used, and the sky value and uncertainty in each. It will then automatically select the *smallest* sky value of the four, and put those into the variables `SKY` and `SKYSIG`. If you do not wish to accept VISTA's estimate of the sky from each corner,

you may enter your own choice from among the four. The second way is to choose a region of the image that is relatively free of stars and extended image, and define your own box in which to do the modal sky calculation. For example, suppose in our image there is a region 55 columns wide by 245 rows high, with the upper left hand corner at pixel (R,C)=(76,345) which contains a "clean" piece of sky. To compute the mode within the region, you would first define a box (say `BOX 2`), and type:

```
GO: BOX 2 SR=76 SC=345 NR=245 NC=55
GO: SKY 2 BOX=2
```

which would load its estimate of the modal sky level and its uncertainty into `SKY` and `SKYSIG`. Note that if you choose a small region, your calculation of the mode will become less certain.

Another way of getting a good estimate of the sky is to proceed semi-iteratively. This procedure is especially useful in crowded fields with many stars or extended objects. It makes more advanced use of the VISTA variables `SKY` and `SKYSIG` loaded by the `SKY` command. In outline, assume you have an image in buffer 2:

1. Choose a region (or regions) which contains mostly sky, and is relatively free of "objects" and define a `BOX` to outline the region(s). Call it `BOX 4`.
2. Compute the modal sky and its uncertainty within the `BOX`, using the command: `SKY 2 BOX=4`. The values of `SKY` and `SKYSIG` are now loaded.
3. Define a "rejection threshold" to be $3\sigma$ above the sky computed modal sky value. Load this into the VISTA variable `THRESH` using the command: `THRESH=3.0*SKYSIG`.
4. Recompute the sky level by using the `CLIP=` keyword of the `SKY` command as follows: `SKY 2 BOX=4 CLIP=THRESH`. The `CLIP=` keyword tells the `SKY` command to ignore all pixels greater than `THRESH`. Note that it *does not* actually change any pixel values in the image.
5. Improved (?) values of `SKY` and `SKYSIG` are now available for use.

While nice, this little procedure is not foolproof. You may need to experiment a bit to find an appropriate threshold value, and with the effects of multiple iterations.

## 4.3 Advanced Techniques

This section presents examples of more advanced image reduction techniques developed over the years for special applications.

### 4.3.1 Image Compression

It may be desirable, in some applications, to compress an image by enlarging the size of the pixels. For example, a 500×500 CCD image with an initial pixel scale of 0.″18/pixel may be compressed into a 250×250 image with a pixel scale of 0.″36/pixel by adding adjacent pixels in both rows and columns. The process of compression is usually referred to as "binning." Usually, this is done to try to improve dynamic range, especially in an oversampled image, at a cost of increased readout noise. VISTA uses the `BIN` command for image compression.

Consider the above example of wishing to compress a 500×500 image by a factor of 2 in both rows and columns, so as to produce a 250×250 image. If this image is in buffer 2, you would issue the command:

```
GO: BIN 2 BIN=2
```

The old image in buffer 2 is destroyed and replaced by the old image compressed by a factor of 2 (`BIN=` keyword). Each pixel will contain the *sum* of the adjacent pixels, and the origin of the compressed image will be set to (0,0), regardless of its origin before compression. Note that only *integral* compression factors are allowed.

Instead of the sum, it is possible to compress an image so that each pixel contains the *mean* of the adjacent pixels. This is done using the `NORM` keyword, thus:

```
GO: BIN 2 BIN=2 NORM
```

will do the same as above, only now the average instead of the sum is in each pixel.

By default, `BIN` will begin compressing an image starting at the image origin. You may specify the pixel on which the compression begins using the `SR=` and `SC=` keywords. Thus, to compress the example image above starting on pixel (1,1) rather than (0,0) you would type:

```
GO: BIN 2 BIN=2 SR=1 SC=1
```

Note that all pixels before (SR,SC) are deleted from the image during compression, and the compressed image origin will be set to (0,0).

Finally, it is possible to compress an image in one dimension only, or with a different compression factor for each axis. For example to shorten the number of pixels in each image row by a factor of 3, but leave the number of rows the same, you would type:

```
GO: BIN 2 BINR=2
```

Note that the original 500×500 (rows×columns) image is compressed into a 500×166 image (*i.e.*, each row is now 500/3=166 pixels long — `BIN` rounds to the nearest integer). Similarly, to shorten each *column* by a factor of 3, and leave the image with the same number of pixels per row, you would have typed: `BIN 2 BINC=2`, which would result in a 166×500 image of 166 rows of 500 pixels each.

### 4.3.2 Image Smoothing

Smoothing (or "Convolving") is an image enhancement technique whose application must be made with circumspection. It is a process of cutting through noise on an image at the expense of instrumental resolution. It is most justified when you want to make large-scale trends visible without the accompanying complication of short spatial-scale (*i.e.*, pixel-to-pixel) variations (as Press *et al.* say in *Numerical Recipes*: "to guide the eye through a forest of data points all with large error bars"). For example, smoothing an image might make a faint extended feature more "visible" by smearing out (literally) the large pixel-to-pixel noise variations in the image. Applied without care, smoothing can often *create* artificial features that are not there.

Smoothing, however, also has definite applications other than purely aesthetic. It is possible, with a carefully chosen smoothing function, to degrade the image resolution (*i.e.*, seeing) of one image to that of another to allow a more fair comparison. Another useful application is for removal of large spatial-scale features from an image (*e.g.*, fixed pattern noise) without introducing additional noise (*i.e.*, small spatial-scale variations) to the image. For example, in broad-band filter images made with thinned, back-illuminated CCD chips, one often encounters "fizeau fringes" due to monochromatic light from the night sky that is absent from the flat-fields (and variable with sky position), and thus not removed by the flat-fielding process. Using images of "blank" regions of the night sky, it is possible to create a smooth "fringe field" frame to remove (or at least mitigate) the effects of fringing.

A smoothed fringe field will retain the fringe pattern (a variation over many pixels), and suppress pixel-to-pixel noise.

VISTA offers three options for image smoothing: Gaussian Convolution, Boxcar Filtering, and Median Filtering. Both Gaussian and Boxcar smoothing are done using the SMOOTH command. In this command, the width of the filter in pixels is specified (either as a FWHM for the Gaussian, or full width for a square, "boxcar", filter). Image convolution is done in the real (image) domain, rather than in the fourier domain, and may be quite slow for large filters widths. The SMOOTH command does two 1-D convolutions, rather than a true 2-D convolution, so filters are *square* rather than round (or elliptical). This is a good first approximation for most applications.

For example, you wish to artificially degrade the seeing in an image in buffer 2 by convolving it with a gaussian filter with FWHM of 1.75 pixels. To do this, you would issue the command:

```
GO: SMOOTH 2 FW=1.75
```

Large images can take long times (>5 minutes) to filter completely. To smooth each image row independent of adjacent rows, you would type:

```
GO: SMOOTH 2 FWC=1.75
```

Note that to smooth each row separately you set the width of the filter in *columns*, as each row is composed of pixels in adjacent columns. Similarly, to smooth each image column independently, you would type:

```
GO: SMOOTH 2 FWR=1.75
```

To switch from a Gaussian to a Boxcar filter, you would append the BOXCAR keyword to the above commands, hence, to smooth the image in buffer 2 with a boxcar filter 1.75×1.75 pixels square you would type:

```
GO: SMOOTH 2 FW=1.75 BOXCAR
```

In all of these cases, the original image is destroyed and replaced with the results of the convolution. Smoothing is irreversible, so care should be taken in its application.

Median filtering is done using the ZAP command in an extreme fashion. To median filter an image, use the ZAP command with the rejection threshold set to 0., thus all pixels are set to the median pixel value within the filter. For example, to median filter an image in buffer 3 with a 5×5 pixel square median filter, you would type:

```
GO: ZAP 3 SIG=0.  SIZE=5,5
```

In most cases, the median is close or indistinguishable from the mean in the filter, and so boxcar filtering (using the SMOOTH command) is probably more appropriate, and is certainly faster. However a median filter is less sensitive than a boxcar filter in cases where there are hot pixels which deviate significantly from the other pixels within the filter.

### 4.3.3 Editing Image Values

There is a facility within VISTA for changing image data at the individual pixel level. It is possible to make a given pixel any value desired, using the SETVAL function in VISTA. For example, to set the pixel at (125,435) in the image in buffer 4 to have the value 653.45, you would issue the command:

```
GO: X=SETVAL[4,125,435,653.45]
```

To provide a "bail-out", the VISTA variable X now contains the value of the pixel at (125,435) *before* it was changed. You can read an individual pixel non-destructively using the GETVAL function, thus:

```
GO: Y=GETVAL[4,35,134]
```

will put the value of pixel (35,134) in the image in buffer 4 into the VISTA variable Y.

### 4.3.4 Image Registration

Image registration is the procedure of aligning features in two (or more) images. It is a two step process. The first step is to determine the coordinates of features (*e.g.,* star images) common to both images, and the second step is to shift one of the images so that they line up. While at first sight fairly simple, it can become a rather involved process. The procedures outlined below have been found to be good general starting points, but the quality of the registration depends to some degree on the general quality and peculiarities of your images, and what you wish to use registration for. For the purposes of adding two images together, these techniques should be more than adequate. For subtraction or division of registered images, you become immediately more sensitive to the accuracy of the registration, and greater care needs to be taken before adopting a particular procedure.

There are two basic ways within VISTA to determine the shift between two images. The first, and most simple, is to measure the centroids of star images common to both, and determine the mean shift in rows and columns between the image centroids. The most convenient way to do this is with the interactive command MARKSTAR. For example, you have two images in buffers 1 and 2, with 5 bright (but not saturated) stars in common. You want to shift the image in buffer 2 so it lines up with the image in buffer 1. To do this you would follow this procedure:

```
GO: TV 2 <span, zero, and color map as desired>
GO: MARKSTAR NEW RAD=1
```

*(Mark stars with cursor, typing* C *to mark the stars. When done, type* E *on image display device to exit from the interactive cursor mode).*

```
GO: MARKSTAR AUTO
```

*(This extra step serves to refine the centroids, marking your old stars automatically by boxing them on the image display. Simply type* E *on image display device to exit from the interactive cursor mode.)*

```
GO: TV 1 <span, zero, and color map as desired>
GO: MARKSTAR AUTO
```

*(If the image registration is not too far off, this will mark the stars from the image in buffer 2 automatically. If it has found them satisfactorily, type* E *on the image display device to exit the interactive cursor mode.)*

The last MARKSTAR AUTO command will printout the mean incremental shift in rows and columns between the images in buffers 1 and 2. Let us suppose that the MARKSTAR AUTO command has returned with DELR=0.24 and DELC=−1.25 pixels. You would then issue the command to shift the image in buffer 2 into alignment with the image in buffer 1 by typing:

```
GO: SHIFT 2 DR=0.24 DC=-1.25
```

After a pause, depending on the sizes of the images, the image in buffer 2 will be shifted so that the five stars should be lined up on both images (*i.e.,* have the same centroid).

If the initial shift between the two images is large, it is best to estimate the approximate whole-pixel shift between them using the interactive cursor (the ITV command), and shift the image in buffer 2 by an integral pixel amount that will

get them close to each other, then refine the registration with the procedure outlined above. Since shifting by an integral pixel amount is very fast (it involves no interpolations as in fractional pixel shifts), the smaller the difference to start with, the more accurate the centroids calculated by the peak-finding algorithms in MARKSTAR.

A second way to estimate the relative shift between two images is to use a two-dimensional cross-correlation. For images which share field stars in common, tests have shown that 2-D cross-correlation is no more accurate than the centroid technique outlined above. Cross-correlation is more involved, and much less economical of CPU time. If however, you feel that it might be more appropriate (and it might be in some applications), here goes.

You wish to align the image in buffer 2 so that it lines up with the image in buffer 1. First, use the interactive cursor (ITV command) to determine the rough shift in whole-number pixels between the images. Using the SHIFT command, bring the image in buffer 2 into rough alignment with the image in buffer 1.

Second, choose a sub-region of the image which contains features in common on both images (say a star or group of stars). Define a BOX to contain this region. Call it BOX 2 for the present example.

Next, determine the sky level within this box in each of images 1 and 2 (see § 4.2.5 above), and subtract the sky from each image as follows:

```
GO: SKY 1 BOX=2
GO: SUB 1 CONST=SKY
GO: SKY 2 BOX=2
GO: SUB 2 CONST=SKY
```

Next, cross-correlate buffer 2 against buffer 1, and put the resulting cross-correlation function in buffer 3. Confine the cross-correlation to the contents of BOX 2 and a radius of 3 pixels.

```
GO: CROSS 3 1 2 RAD=3 BOX=2
```

Compute the position of the peak of the cross-correlation function in buffer 3. The position of the peak in (rows,columns) is the relative shift between buffers 1 and 2. Compute the shift using the SURFACE command by assuming that the cross-correlation function is well described by a parabolic surface, and derive the position of the peak of this function (*i.e.,* the point of zero slope). Once this shift is computed, perform the shift. To do this, you would issue the following commands:

```
GO: SURFACE 3 LOAD
GO: DENOM=4*COEFFC2*COEFFR2-COEFFRC*COEFFRC
GO: DELR=MIDR+(COEFFC*COEEFRC-2*COEEFR*COEEFC)/DENOM
GO: DELC=MIDC+(COEFFR*COEEFRC-2*COEEFR*COEEFC)/DENOM
GO: SHIFT 2 DR=DELR DC=DELC
```

The VISTA variables COEFFC, COEFFR, COEFFRC, COEFFR2, COEFFC2, MIDR, and MIDC are the coefficients of the 2-D polynomial surface fit loaded by the SURFACE command. See the VISTA Help Manual for details. These are used in the above procedure to compute the relative shift, DELR and DELC.

As noted before, the procedure is fairly involved, and for most applications encountered by the author, gives the same result that would be obtained using the simple centroid technique described first. The principle limitation of image registration is the degree of matching between the two images, and it is fundamentally limited by the degree of sampling. An undersampled image (taken under good seeing or with a large pixel scale) cannot be accurately registered to more than a few percent of a pixel to begin with. In addition, seeing variations (due to the atmosphere, or, more commonly, guiding errors) can also limit the registration accuracy.

The SHIFT command offers two interpolation schemes: simple bi-linear interpolation and sinc function interpolation. In the author's experience (which is pretty extensive), bi-linear interpolation gives perfectly acceptable results for a lower investment of CPU time compared to sinc interpolation. There is no measurable difference between the mean residual registration errors noted after using either interpolation scheme for the shift in most applications. This result, however, should be taken with caution, and could well depend on the type of image data you have. Thus, you should test both interpolation schemes before proceeding.

### 4.3.5 Image Stacking and Combining

Image stacking (sometimes called "co-adding") is a technique whereby many exposures of a single image are added together to simulate the effects of a longer exposure. This is very useful if you wish to preserve flux information from objects in your images that might be saturated on a longer exposure. Stacking serves to increase the effective dynamic range of your imaging data at a cost of increased readout noise (how expensive depends on how noisy your CCD is; for most CCDs it may be negligible).

To stack a set of images, the first step is to bring them into alignment (see "Image Registration" above). Then add them together using the ADD command. Nothing could be simpler. The reason that this is in the "black-belt" techniques section is that it involves the step of image registration. However, if you have taken great care to keep guiding as steady as possible while taking the data (and not changed the filter at all), then accurate registration should not be necessary, and image stacking is then trivial. In a sense (modulo an integral factor), you are constructing the *mean* of the "stack" of CCD images.

Another technique for combining multiple images into a single image is to construct the *median* of an image stack. This is useful for rejection of cosmic ray and other transient events among the images as computing a median effectively rejects values which deviate strongly from the most common value of a given individual pixel among the images in the stack. The more images in the stack, the more effective constructing the median image will be at transient noise rejection. However, more images also means much more computational overhead, so there is a trade-off. Some observers compute the median of a stack of flat-field frames to represent the best flat-field for an observing run.

To construct a median image of a stack of images, you would first read the images into VISTA buffers, leaving enough room in virtual memory for the median image. For example, you wish to construct the median of a stack of 6 images. Read each of these 6 source images into buffers 1 thru 6, and then issue the command:

```
GO: MEDIAN 7 1 2 3 4 5 6 TTY
```

The median image will be computed and stored in buffer 7. The TTY keyword is an option which serves to report the progress of the median calculation. You must have *at least* 3 images to perform a median computation. In addition, if the images are not in good registration (see above), then the derived median image could contain unwanted artifacts (a nice way of saying its screwed up).

Using MEDIAN as a way of rejecting cosmic rays is especially useful for long-exposure images taken with thick CCD chips (like the GEC P-8600 or thick RCA and Tektronix CCDs). However, you must have planned your observations around applying this technique in order to have enough images to compute a good median.

### 4.3.6 Image Mosaics

In addition to combining images vertically by stacking them together (see above), larger images may be created by laying images next to each other, side-by-side, like

tiles in a mosaic. The simplest "image mosaic" is made by laying together two (or more) images that do not overlap. This is most often done for sets of images taken in different filters so they may be displayed together.

To create a simple mosaic of two images (it may be easily extended to more as need be), first create an appropriate "substrate" image, then add the individual images, properly aligned, to create the mosaic.

For example, consider two images, in buffer 1 and 2, which are both 223×233 in size. You wish to combine them so they lie side-by-side, with the image in buffer 1 on the left. In buffer 3, you will create an empty image large enough to hold both images side by side with a 1-pixel border around and between them. Then, you will have to shift the individual images in buffer 1 and 2 so that they land in the appropriate parts of the mosaic image, and then add them to the mosaic. To do this, you would issue the following commands (I'm purposely multiplying the steps so you can follow the logic):

```
GO: CREATE 3 NR=NR[1]+2 NC=2*NC[1]+3
GO: CH 3 'Two Image Composite'
GO: SHIFT 1 DR=-SR[1] DC=-SC[1]
GO: SHIFT 2 DR=-SR[2] DC=-SC[2]
GO: SHIFT 1 DR=1 DC=1
GO: SHIFT 2 DR=1 DC=NC[2]+2
GO: ADD 3 1
GO: ADD 3 2
```

The mosaic is now in buffer 3.

A more difficult procedure is to produce true image mosaics by aligning images where their fields overlap to create a larger overall image. This is done where a CCD camera with a small field of view is used to image a larger region, and is a common technique with Infrared imaging arrays which, at present, are limited in size. Creating a multi-image mosaic involves determining the relative shift between common features on the component images (see "Image Registration" above), and then merging the components, averaging the regions of overlap to attempt to achieve as uniform a background as is possible. I will not give a detailed example, but outline the steps and relevant command procedures to follow.

1. Choose one image which is to remain fixed during the mosaicing procedure. For convenience, let this be the image in buffer 1. We will proceed to build up the mosaic in buffer 1.

2. Determine the sky level of the image in buffer 1, and subtract it from the image (see "Sky Subtraction", § 4.2.5)
3. Load the next component of the mosaic into buffer 2.
4. Following the "Centroid Method" for image registration described above (§ 4.3.4), determine the relative shift between common objects (*e.g.*, stars) in the images in bufferS 1 and 2.
5. `SHIFT` the image in buffer 2 so it is aligned with the image in buffer 1.
6. Determine the sky level of the image in buffer 2, and subtract it from the image.
7. Merge the image in buffer 2 with the mosaic (buffer 1) by issuing the command: `MERGE 1 2`
8. Iterate back to Step 3 and continue until the mosaic is complete.

It should be possible, in this way, to build up an image mosaic consisting of a large number of components. Each piece changes the size of the image in buffer 1. To avoid potential problems with FORTRAN, it is necessary to `SHIFT` the image in buffer 1 so that its starting row or column is not at a negative pixel value. Some VISTA routines don't mind a negative starting pixel value, but others may crash horribly.

## 4.4 Final Remarks

Despite the tone of the above, it is important to point out that there are no hard and fast rules for how to reduce or analyze imaging data with VISTA (or any other program for that matter). Ultimately, you will have to decide what information you wish to derive from your images. How you should go about that in detail depends almost entirely upon the context in which you are working. It is hoped that the procedures outlined above will give you some idea of how to make use of the tools VISTA provides, as well as suggesting other image processing possibilities to you.

# Chapter 5

# Spectral Data Reduction

## 5.1 Basic Techniques

This section describes the basic techniques used to reduce long-slit spectroscopy like that obtained by the Cassegrain spectrographs and the non-echelle coudé spectrographs. Only the most basic procedures are considered here, with the following section serving to cover more advanced reduction techniques and details which will not generally affect the average user. It is recommended, however, that even casual users at some point read the advanced section so that they can be sure that they are not missing some technique which is important for them.

The main command of interest in spectral reduction is the `MASH` command, which takes a two-dimensional long-slit image and creates a one-dimensional spectrum from a section of it. Before this is done, of course, the usual treatment of baselining and flat-fielding the two-dimensional image must be applied (described in the beginning of this manual). The form of the `MASH` command is:

```
GO: MASH dest src SP=s1,s2 BK=b1,b2 COLS SKY=s REFLAT
```

There are some other keywords which might prove useful — see the manual for details. The `SP=s1,s2` keyword specifies the columns (or rows) which define the object spectrum. More than one `SP=` keyword can be used to co-add several regions of spectrum, but in general only one region is specified at a time. The `BK=b1,b2` keyword specifies the background to be scaled and removed from the spectrum. Usually this contains sky which must be removed but even in coudé spectra, where there may not be a large sky contribution, there is often a constant background across most of the CCD which should be removed. More than one background region can and often is specified, usually on either side of the spectrum region.

The MASH command is somewhat clever about the SP= and BK= windows. If a given column (or row) is specified in *both* SP= and BK= keywords then the spectrum designation is given priority and it is used only as a spectrum column (or row). This allows the user to use a form like :

```
GO: MASH 10 1 SP=314,324 BK=284,354 REFLAT SKY=11 COLS
```

where the background specified in the single BK= keyword will actually define two background regions, columns 284 – 313 and 325 – 354, straddling the spectrum.

Before we describe the other keywords a word is in order about how best to locate the spectrum and background regions. A simple PLOT command using the RS= keyword (or the CS= keyword if the spectrum runs along rows) will usually do the trick. From this the columns at which the spectrum become visible above the background can be determined, and the regions where the background is relatively flat can similarly be found. Note that depending on the application different criteria for the spectrum rows might be used. Some users prefer to define the spectrum "window" when the crosscut through the object's spectrum has risen to 10% of its peak value, but the individual user will have to determine what is best for his or her needs.

Now back to the other MASH keywords. The COLS keyword tells MASH that the spectrum lies along columns; otherwise MASH assumes that the spectrum lies along rows on the CCD. The REFLAT keyword is optional. It tells MASH to use a parabolic fit to each background row (or column) instead of simply taking an average. This usually improves the subtraction of bright, narrow night sky lines, but occasionally REFLAT will actually make things worse. This can happen, for example, if two narrow but widely separated background windows are being used. In such a case it is clear that a parabola is not well-constrained in the region between the backgrounds, where presumably it is most critical to the spectrum subtraction. Also note that REFLAT modifies the original image by dividing each row (or column) by the parabolic fit to that row's background pixels. The SKY= keyword is used to place the *background* spectrum into a buffer. This can then be used by the SKYLINE command to remove first order wavelength shifts induced by spectrograph flexure.

### 5.1.1 Wavelength Calibration

Another important step in the reduction is the wavelength calibration of the spectrum. Very high-precision calibration may rely on rather "exotic" techniques such as absorption cells in the light path, Edsel-Butler interference bands, etc. These

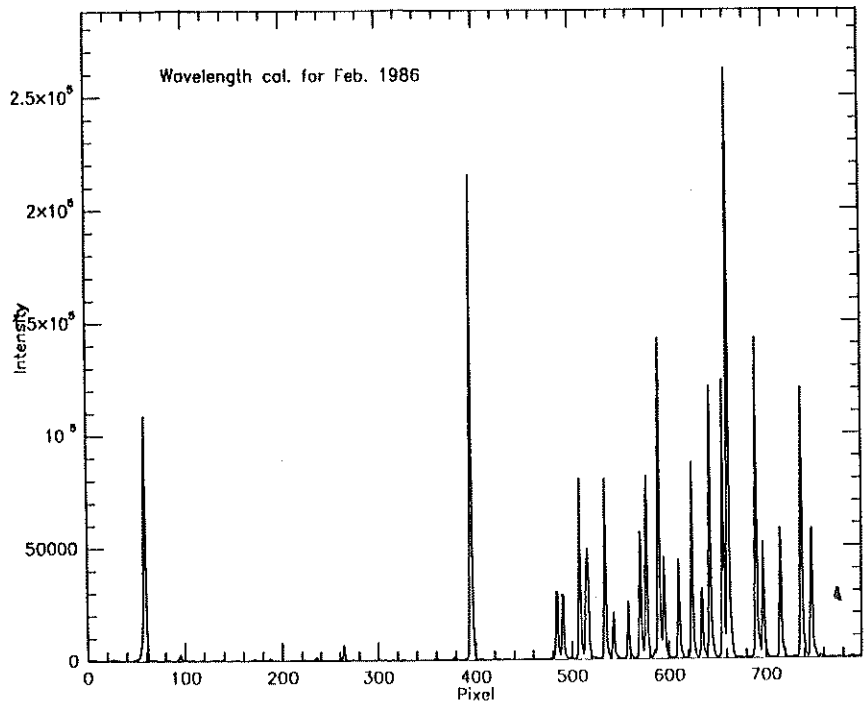



**Figure 5.1**: An example of a wavelength calibration image.

will not be discussed here, nor in the advanced techniques section, as they are the realms of radial velocity experts and others. Ordinarily, the user will have one or more exposures of wavelength calibration lamps, usually containing some gas or gases like mercury, helium, argon, neon, iron, thorium, and so on. These images must be reduced to spectra using the MASH command, although in general there is no need to flat-field the image, and no background specification is used in the MASH command. An example of a wavelength calibration spectrum is shown in Figure 5.1.

With one or more of these calibration spectra available we are ready to try to identify the lines. It is assumed that an appropriate wavelength list has already been written into a disk file (see the manual for how to do this if a list does not already exist for you), and that you have a rough idea as to the dispersion and central wavelength of the spectra. Then use the LINEID command:

```
GO: LINEID 10 FILE=wavelist CEN=5760 DISP=4 TTY INT
```

where "wavelist" is the wavelength line list file, 5760 Å is the approximate central wavelength in this hypothetical case, and 4 Å is the dispersion. The routine will

try to identify the lines in the spectrum using the specified central wavelength ($\pm$100 pixels) and the specified dispersion ($\pm$5%). The TTY keyword prints out the final line identifications, while the optional INT keyword allows the user to edit the line list, removing lines that the program misidentifies and adding lines that it misses. The line identification is usually quite automatic and doesn't require the interactive line list editing, but difficult cases can occur and can require substantial user intervention. Note that if two or more sets of different wavelength calibration spectra are to be used there are two ways to combine the fits. The first is simply to add the spectra together and do a LINEID on the summed calibration spectrum. The other is to use multiple LINEIDs and after the first one use the ADD keyword to each new LINEID command to append the new line list to the previous one.

Now that LINEID has provided a list of identified lines with their expected wavelengths and observed pixel values the WSCALE command fits a polynomial to these points to provide the actual wavelength calibration. The form is:

```
GO: WSCALE 10 ORD=3 INT
```

where ORD=3 specifies a polynomial of order 3 (i.e. a cubic) and INT allows the interactive manipulation of line weights. The particular circumstances of a set of wavelength calibration spectra will determine the order of the fit which the user wants to use, although the program limits this to less than 8. When in doubt, start at a higher order (for instance 5) and see if the highest order terms in the fit are significant, using the uncertainties in each parameter which are calculated and printed in WSCALE. If these terms are not significant then you may lower the order of the fit and try another WSCALE. On the other hand, for cases in which there is a paucity of calibration lines in one or more regions of the spectrum it may be more prudent to use a lower-order fit so that where the polynomial is poorly constrained the high-order terms do not send it into large oscillations. The user will have to make his or her own decision in the end.

WSCALE contains a (rather conservative) automatic line-rejection feature, but it is usual to use the INT keyword to remove lines that are not fit well. The WSCALE command will print out a list of the lines, their weights in the fits and the residuals from the polynomial fit. Then it will ask for a line to be changed (specified by an index number so that you don't have to type in the whole wavelength) and then the new weight to give that line. A weight of zero will completely remove the line from the fit. Entering "$-1$" for the line index to change will either return the routine to make a new fit with the altered line weights, or if no line weights were changed then it will assume that you are finished and will calculate the reverse fit (i.e. pixel as a function of wavelength instead of vice versa). Note that it is actually the *reverse* fit

which is used by the ALIGN command (discussed below) which re-bins the data onto a linear or logarithmic wavelength scale. Once a satisfactory fit has been attained the wavelength calibration spectrum must be saved, as the coefficients of the fit are stored in the header of the spectrum that was specified in the WSCALE command. To copy these coefficients from the calibration spectrum to another uncalibrated spectrum use the COPW command (e.g. COPW 12 10).

### 5.1.2 Linearizing the Wavelength Scale

Quite often the user wishes to re-bin his or her data onto a linear (or logarithmic in some cases) wavelength scale. This is done via the ALIGN command. There are a number of options in ALIGN which will not be covered here; the user should look over the help manual for details. It should be mentioned here that ALIGN also converts the data from counts to counts per second per Angstrom, unless the NOFLUX keyword is specified. Always keep this in mind, because often seeming "bugs" which change the scale of spectra can be traced to this feature of ALIGN. Also note that, as mentioned above, ALIGN uses the *reverse* wavelength fit (i.e. pixel as a function of wavelength).

More immediately relevant options include the LGI keyword. By default interpolation is done with a sinc interpolation scheme which more or less preserves the frequencies in the data. But many users find that the sinc technique leads to oscillations often called "ringing," common to many Fourier-based routines. To avoid this the LGI keyword invokes a 4-point, flux-conserving Lagrangian interpolation. The other available keywords in ALIGN allow the user to remove (or put in) a velocity or redshift, a shift in pixel space, and other special features.

If the SKY= keyword in MASH is being used to find a first-order correction to the wavelength calibration by using the night-sky lines then the spectrum buffer containing the sky should be ALIGN'ed and the SKYLINE command used:

```
GO: ALIGN 12 LGI
GO: SKYLINE 10 12
```

assuming that the sky is contained in buffer 12 and the spectrum to be adjusted is in buffer 10. SKYLINE looks for a few night-sky lines, mostly in the yellow part of the spectrum, and compares their observed wavelengths with their expected wavelengths. A mean shift from these night-sky lines is calculated and the starting wavelength of the object's spectrum is adjusted.

### 5.1.3 Flux Calibration and Extinction Correction

Once a wavelength scale has been associated with a spectrum (whether or not it has been ALIGN'ed) the EXTINCT command can be used to correct for the atmosphere's extinction. The extinction curve appropriate for Mount Hamilton is the default, but other extinction curves (e.g. for CTIO) may be available. This command then converts observed flux (or counts) to flux (or counts) above the Earth's atmosphere.

At the Cassegrain focus users are very often concerned with determining an absolute flux calibration of their spectra. This is usually done by observing a flux standard and MASH'ing, ALIGN'ing, and correcting its spectrum for atmospheric correction as usual. Then the command FLUXSTAR can be used to create what is called a "flux curve," which, when multiplied by an extracted, extinction-corrected spectrum of an object, will yield a flux-calibrated spectrum. The FLUXSTAR command is of the form:

```
GO: FLUXSTAR 14 FEIGE56 SYSC
```

where 14 is the buffer containing the flux standard, FEIGE56 is the file name containing the monochromatic magnitudes of the flux points (see the help manual for how to set one of these up if necessary), and SYSC tells the program to create a particular type of flux curve. There are three types of flux curves which can be created. The SYSB keyword (also the default if no option is specified) takes the flux points and calculates the correction (known flux divided by measured counts) at each point within the spectrum. A spline is then fit through the flux points and the resulting smooth curve is the correction curve (called the "flux curve"). Note that the spectrum buffer which is specified in the FLUXSTAR command is replaced by the flux curve, so you may want to copy the standard star spectrum to another buffer before using this command. Now any extracted spectrum can be multiplied by this curve to obtain the "fluxed" spectrum of the object. Some of the strong atmospheric absorption bands in the near-infrared (the A- and B-bands) are saturated (optically thick) and hence they are not removed by sky subtraction like most other night-sky features. To first order they can be removed using the SYSC option, which clips out those parts of the spectrum in the standard star containing the A- and B-bands and splices them into the spline flux curve. An example of this is shown in Figure 5.2. This technique works to a certain extent but the strengths of these bands varies with the $O_2$ content along the line-of-sight, and this can change rapidly from point to point in the sky and from one moment to the next.

More sophisticated techniques are discussed in the advanced section. The SYSA option is a little-used option which essentially uses the entire standard star spectrum, inverts it, and then uses the SYSB spline fit to normalize it. Hence some of

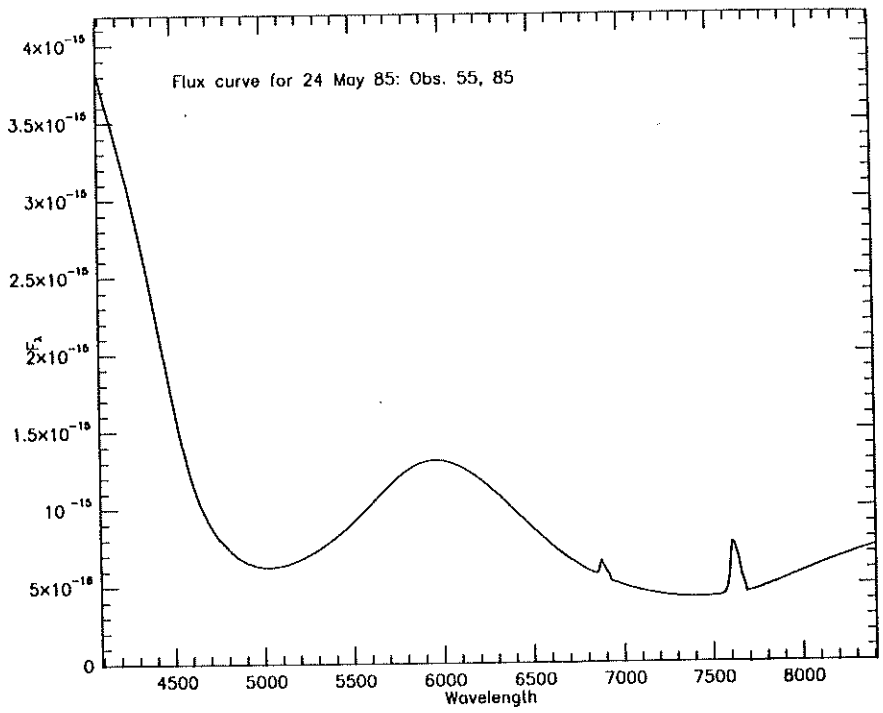



**Figure 5.2**: Example of Flux Curve.

the features in the standard star spectrum are retained (inverted) in the flux curve. The usefulness of this option is very limited and it will not be discussed further. The choice of flux stars and techniques for taking the data is discussed further in the next section.

## 5.2 Advanced Techniques

### 5.2.1 Flat-Field Notes

The treatment of flat-fields also requires some discussion. There are two types of effects which the flat-fielding procedure is expected to remove. The first, most important effect is the pixel-to-pixel variation in sensitivity. The second type of effect is more global, lower frequency variation in sensitivity which might be due to actual quantum efficiency variations, blaze functions, interference fringes, vignetting, etc. It should be stressed that the pixel-to-pixel variation is usually the most important effect to account for with a flat-field, because there is usually some other technique

such as using a flux curve which will remove the lower frequency patterns. In this sense it is sometimes desirable to actually *remove* the low frequency variations in the flat-field and retain only the pixel-to-pixel effects. This technique is called flat-field normalization, and is most often done *along* the dispersion direction. A MASH may be used to obtain the shape of the flat-field along the dispersion direction, which may then be smoothed or fit with a spline or polynomial. This smoothed spectrum is then STRETCH'ed into an image and the original flat-field is divided by this smoothed version.

The EXTRACT command discussed below actually requires a normalized flat field, because that is the most straightforward way of keeping track of the actual number of electrons which have fallen into any given pixel. Procedures which calculate the quantum efficiency of the spectrograph also often use a normalized flat. Another use is when filters have been used in the flat-field (usually to suppress the red end of the spectrum and get a more color-balanced spectral shape) which have rather high frequency structure in their spectral shape. However, *care must be exercised*! There are real low-frequency effects which you probably *don't* want to remove, such as interference fringes in the near-IR and certain efficiency effects in gratings. For instance, the grating efficiency as a function of wavelength is dependent on the *polarization* of the light striking the grating, and often this efficiency has frequencies in it which are too high to be properly removed by the flux curve. This is also an appropriate place to mention that when trying to retain the highest accuracy with grating spectrographs, it is probably a good idea to avoid using some standard stars like Hiltner 102 which are also highly polarized. The flux measured will depend to a small extent on the interaction between the polarized starlight and the grating's intrinsic polarization, and hence on the spectrograph's orientation.

### 5.2.2 MASH alternatives: SPECTROID and EXTRACT

While MASH is the most commonly used method of changing a 2-D long-slit image into a one-dimensional spectrum, there are a couple of alternatives. The most useful of these is the SPECTROID command, which performs two functions. First it will calculate the centroid of the spectrum, using a spectrum window and a set of background window specifications. If the NOMASH keyword is used then the command will return this centroid as a function of column number as the spectrum. (Note that SPECTROID only works when the spectrum runs across rows, so the ROTATE command may have to be used to place the image into the proper orientation first.) This centroid can be valuable by itself or it can be used, smoothed or as is, as a "model" for a fainter spectrum on another image. An example of a centroid is shown in Figure 5.3. Users have thought up many uses for this command.

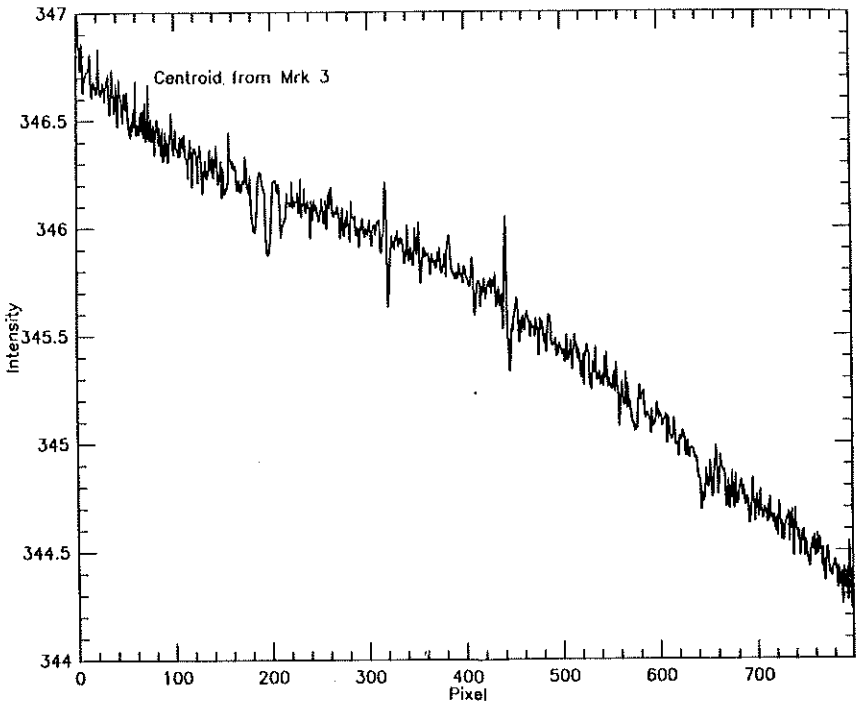



**Figure 5.3**: An example of a centroid derived from the SPECTROID command with the NOMASH keyword.

If the NOMASH keyword is *not* used then the command will continue, and using the centroid as a function of column which the program has already calculated it will extract the spectrum as specified in the command line. In contrast with MASH, SPECTROID interpolates for fractional pixels. It is useful for many things, most obviously for extracting spectra which are tilted or curved across the CCD. It is the main utility used in the echelle reduction because of this capability. It is also useful for mapping distortions. (Use a night-sky line or a wavelength calibration line as a "spectrum" to get an idea of the curvature of these perpendicular to the dispersion direction.) It can calculate the distribution of a particular emission line along the length of a slit, and it can even be used to calculate higher order moments of the light distribution as a function of column number, using the MOMENTS keyword.

Another command of perhaps more limited usefulness is the EXTRACT command. This is an "optimal extraction" routine patterned after that by Keith Horne (see *Publ. A.S.P.*, **98**, 609) and is used for extracting very low signal-to-noise spectra which are dominated by shot noise from the sky. It works best for continuum sources rather than pure emission-line sources. EXTRACT has a number of useful features, and uses the gain and readout noise of the CCD to estimate the expected

noise for each pixel in the original image. Because of this it requires a normalized flat-field (discussed above) so that the number of counts will be approximately conserved during the flat-fielding. EXTRACT has the ability to automatically reject points from the operation, if they are farther away from the parametrized fit than is expected from the calculated uncertainty in each pixel. This feature includes rejection of points from the spectrum itself, something which is not easy to do with the ZAP command in many cases.

The algorithm which EXTRACT uses is beyond the scope of this cookbook, but in brief it parametrizes the point spread function (PSF) using polynomials. It also parametrizes the background with polynomials, and should a point fall more than a certain tolerance away from its respective fit (using the calculated uncertainty) then it is rejected and the fit is repeated. Because of the way in which the parametrization is done there are limitations to the use of EXTRACT. In particular highly tilted spectra cannot be correctly extracted with this command, although highly creative use of the SHIFT command with a model may allow the user to "fake it." Also, it is known that the polynomial parametrization of the PSF is not very accurate with high flux levels, and hence any spectrum which is well-exposed should be created with MASH or SPECTROID. The value of EXTRACT in improving the signal-to-noise in a spectrum is only manifest in *very* faint spectra, those less than 5% of the night sky, and hence it is not used very often.

### 5.2.3 Spatial Distributions and 2-D Distortion Corrections

One of the nice features of long-slit spectroscopy is the ability to map out the intensity of a particular line or other feature as a function of position along the slit. To do this simply treat the line as a spectrum and the slit direction as the dispersion direction. Then MASH or SPECTROID can be used to extract the spatial distribution of the feature.

The SPECTROID command can also be used in its NOMASH mode to map out the center of a line along the slit, which may be useful, for example, in showing the velocity of a feature as a function of position along the slit. It can also be used to map the distortions of the instrument along both the dispersion and the slit directions. But no one but Jesus knows how to do this.

### 5.2.4 "Re-fluxing" and "Atmospheric Correctors"

The normal observing procedures usually do not allow for a particularly accurate absolute flux calibration because of slit losses and differential atmospheric refrac-

tion. The former effect is by far the larger, and does not adversely affect the *relative* flux calibration, and hence line ratios. Normally observers are not overly concerned with the absolute flux levels, as long as the relative levels are accurate. However, there are observing techniques which allow quite accurate absolute fluxing of objects. In general the observer must take a spectrum of the object using a "normal" slit width to retain spectral resolution. But then a shorter, "wide-slit" observation can be taken in which the slit width is increased to a size which will encompass the total flux from the object. The division of the smoothed wide-slit spectrum by the smoothed normal spectrum yields a wavelength-dependent slit loss correction. This technique is often called "re-fluxing." In this case the flux standards taken throughout the night must also be taken with a wide slit to assure that all of *their* light has entered the slit.

As mentioned above there are certain night sky absorption features which are not easily removed from the spectrum with normal sky subtraction techniques. In particular, the water vapor content of the atmosphere changes dramatically from hour to hour along one line of sight through the atmosphere, as well as from one line of sight to another. Hence, atmospheric correction of these water absorption features is often done using a star which has few lines in the spectral region of interest and which is as nearby in the sky as possible to the object to be corrected. The spectrum of this "atmospheric standard" must be taken close in time to the object's spectrum to assure accurate results.

# Chapter 6

# Hamilton Echelle Reduction

This chapter describes the techniques used to reduce data obtained with the Hamilton echelle spectrograph at Lick. As the reduction of Hamilton data is significantly more complex, we shall follow in this chapter a somewhat different approach than in the previous chapters. A detailed discussion of all the steps of Hamilton data reduction would indeed be extremely tedious so we will discuss instead the procedures used to reduce these data. These procedures are well-suited to the batch processing of echelle data; but also give the user some flexibility in following some of the procedures in an interactive mode. The idea was to have highly automated procedures which require as few inputs from the user as possible, but still allow some control over reduction parameters.

In addition to these procedures two VISTA commands have been created to help reduce Hamilton data. The first one called `EWAVE` is used for the wavelength calibration; it is discussed in §6.5. The second command is called `EXTSPEC` and is used to extract a given order from a wavelength-calibrated image where each row corresponds to a different order. This command preserves the wavelength information of the original image. It will be discussed in §6.7.

The first section will cover what the Hamilton user needs to obtain at the telescope to correctly reduce the data. Then we will show how to use the two top-level procedures that prepare and extract the raw data. Next we will discuss the techniques used to remove interference fringes in the data and how to proceed for the wavelength calibration. Finally, the flux calibration procedure will be described along with considerations on how to combine the orders of an Hamilton frame.

These routines, in principle, could be used for reduction of any echelle spectrograph data. Echelle spectrographs all have unique properties (for example, how the echelle images are affected by interorder scattered light), thus a general discussion

of echelle reduction with VISTA is well beyond the scope of this cookbook. However, it does provide a good starting point for users interested in such an application of VISTA.

## 6.1 Input Data

First a word about what you need to get at the telescope in order to reduce your Hamilton data. Using the "Dome Quartz" lamp you need to take open-decker flats. These flats will remove the pixel-to-pixel variations in sensitivity of the detector. If your data cover the spectral range above about 6000Å you also need to take short-decker (1-2 '' ) flats in order to remove the color- and position-dependent fringing effects (see §6.4). It is recommended to obtain both these flat-field exposures using broad-band filters to get a more even illumination and thus minimize the number of exposures required to obtain good counts everywhere on the chip. (e.g. use a BG 14 or BG 38 for $4500 \lesssim \lambda \lesssim 8000$Å and a UG 5 for $3500 \lesssim \lambda \lesssim 5000$Å). For wavelength calibration purposes you need to take exposures of the Th-Ar lamp spectrum. Once again a combination of exposures using broad-band filters can be used to try to reduce the intensity of the strong Argon lines above about 7000Å and some Thorium lines.

It is recommended to take a short (1–2 seconds) dark exposure. It will be used to remove the fixed pattern from the data (see Chapter 3 for a brief discussion of the fixed pattern). Note, however, that the acquisition of a short dark is just a further precaution to correctly remove the fixed pattern since the method used to subtract the interorder light in the procedures discussed below should remove this fixed pattern from the data. Some users also like to take a long dark exposure ($\gtrsim$ 15 minutes) to define an overall dark rate and/or help remove the hot columns on the "old" TI 800×800 CCD (for this last purpose, the long dark has to be of the same exposure time as the image from which the observer wish to remove the hot columns and simple modifications of the procedures described below are necessary). This determination of the dark rate is only approximative since it is known to depend on a number of factors difficult to control (e.g. whether the light in the coudé room have been on recently, whether there are any glowing oscilloscope screens in the coudé room, whether an exposure of a bright star or a calibration lamp has been taken recently, etc). Finally, if you wish to flux your Hamilton data, you need to take a well-exposed spectrum of one of the standard stars out of the list of Goodrich and Veilleux (1988).

## 6.2 Preparation of the Data

Before starting, you need to tell VISTA where the Hamilton reduction procedures reside on your machine. On most VISTA installations, these will be in the default VISTA procedure directory. For the example below, we shall use the installation as it exists on the Lick VAX 11/780, thus the procedures files are in the directory DRA0:[CCDEV.RELEASE3.HAM]. We make the following logical definition:

```
$ DEFINE V$PRODIR DRA0:[CCDEV.RELEASE3.HAM].
```

There are two top-level procedures used to extract the Hamilton data. They are *HAMPREP* and *HAMREDUCE*. The first prepares things which you will need for most of the rest of your data reduction, like a fixed-pattern spectrum, an estimate of the general dark count rate on the CCD, a flat-field sum, and the positions of the orders across the CCD. The second, *HAMREDUCE*, assumes that all of these have been prepared and is used to reduce subsequent images. Another very important procedure which both of these call is the *SETPARAMS* procedure. This provides all of the default values for things such as the sizes of the spectrum and background extraction windows, the names of the files for fixed pattern, flat, etc., flags to turn various features on and off, a flag to determine whether you are using the tape drive or files already transferred to disk, etc. It is advised that you have a hardcopy of this file so that you can see what the available parameters are in case you ever need to change them. Almost all of the other procedures have a statement at the top which will not allow them to be run unless *SETPARAMS* has already been run, but if you are only using *HAMPREP* and *HAMREDUCE* don't worry, the first thing these procedures do is call *SETPARAMS*.

The next thing that these procedures do is call a routine called *CHANGEPARAMS*. Currently the only thing that this does is to pause to allow you to alter the default values which *SETPARAMS* has set up. Someone might prefer to put into *CHANGEPARAMS* a set of questions which will ask the user for various parameters to be changed but it seems that this is not really necessary. After *CHANGEPARAMS* has paused and you have made any changes type `CONTINUE` and the procedure will resume.

In *HAMPREP* you will be asked a number of questions. We will assume throughout the rest of this section that you are reading data from tape — if you are reading data from disk `SET TAPE=0` and the program will ask for the disk file name instead of the tape file number. Also, be advised that the disk files should be transferred directly from tape to disk, without doing a baseline correction or anything (without using full precision) — all of this is done within the procedures. The first thing you

will be asked is the file number of an observation containing a short dark exposure. This will be used to construct the fixed-pattern spectrum. If you don't want to correct for the fixed pattern then `SET USEFP=0` in *CHANGEPARAMS*.

Next the file number of a long dark will be requested. This will be used to define an overall dark rate. As we mentioned above the dark rate is somewhat difficult to determine accurately and if you want to skip this part then `SET USEDARK=0`. Otherwise, the dark rate in units of DN/sec will be determined and printed out. The next time *SETPARAMS* is run the value of the VISTA variable `DARK` will be reset to 0, so during *CHANGEPARAMS* you will have to reset it to the value determined previously. Because a constant dark count across the chip is also removed by the background subtraction this step is often skipped to save some reduction time.

Next come the flat-fields. *HAMPREP* will ask for the number of flat-fields, then for the list of file numbers. These will all be averaged together and written to disk. Finally, the routine will ask for the tape number of the "standard" star. This doesn't necessarily mean a "flux standard" in the usual Cassegrain sense, but merely a well-exposed spectrum from which the order positions can be determined accurately. Since it is best to have a featureless spectrum for this (although not at all critical) it is suggested that some hot, bright star be used. Next you will be asked for the position (near the left side of the CCD) of an order, and then the number of this order. These numbers can easily be recorded on the mountain while you are taking the data — otherwise you might want to read an image into VISTA before starting *HAMPREP* and determine these numbers from the AED image display or do all this while in *HAMPREP* by typing `-1`. If you are off by a little on the order number the routine probably won't get lost, but you need to rectify this situation before you can do the wavelength calibration, and preferably immediately after running *HAMPREP*.

*HAMPREP* will then iterate to find and fit polynomials to each order on the CCD. After this is done it will ask for the name of the output file for these data. The order positions will be written out to a file with the extension .ORD and the spectrum of the object will be written out using the extension .CCD. In these files the x-axis is in pixels and the y-axis corresponds to the order number (pixel–order space).

## 6.3 Extraction of the Data

Now that all of the preliminary images and whatnot have been formed you can run *HAMREDUCE* to start reducing your data. *SETPARAMS* and *CHANGEPARAMS* will

be run. Type `CONTINUE` when you are finished resetting the values of the variables you wanted to change. Next the file number to be reduced will be requested. This image will be "prepared" by removing the fixed pattern, a mean dark count, interpolating over the bad columns and flat-fielding. Note that the mask used for the interpolation over the bad columns is by default the map for the "old" TI CCD; if you are using the NSF #4 CCD you need to set the string `BADMAP` to `NEWINTERP.MSK` (instead of the default name `INTERP.MSK`) when *CHANGEPARAMS* is run. Next the following question will be asked:

```
Do you want to locate the orders from this image
(type ''0'') or use the order positions determined
from the standard star (''1'') ?
```

If you answer "1" the name of the file containing the order positions will be requested. If you answer "0" you will be asked for the position of an order near the left side of the chip and the order number. The order fits will be re-done using this image (which hence must also be pretty well-exposed). The usual choice here will probably be "1", but the option exists. The procedure will first determine the background intensities between each order and will then proceed to extract the intensities in each order. The extraction windows for the background and the spectrum are by default 3 and 10 pixels, respectively. These can be changed by setting the variables `BKW` and `SPW` to the values you desire while still in the procedure *CHANGEPARAMS*. When `BALANCE` is not set to zero the procedure will make a first order correction to the background intensities for the presence of irregularities in the open-decker flat-field exposure. By default the background intensities will also be median filtered and smoothed by a 1-D. gaussian with FWHM of `BSMOOTH=21` columns before being subtracted from the data. If you wish to skip this part of the reduction (this may be the case if your backgrounds are strongly affected by the presence of some strong hot columns) simply `SET BSMOOTH=0`. Finally, the name of the output file will be requested and the extracted spectrum contained in BUFFER 4 will be written to disk. At this point BUFFER 5 contains the treated background. Rerun *HAMREDUCE* for each of your other images.

It is recommended to run these procedures in BATCH mode since *HAMPREP* takes more than 20 cpu minutes to complete while *HAMREDUCE* requires about 10 cpu minutes per object (for a VAX 11/780 running under VMS version 4.3 with FPA). These numbers are only indicative since they obviously depend on the number of orders in the raw image and which options you use. To make it easier, a sample DCL command file (with comment statements added) to show what numbers you have to provide to run *HAMPREP* and *HAMREDUCE* is shown below.

Directories containing the necessary files are defined for the VISTA installation on the Lick VAX/VMS 11/780. Actual directory names and batch command syntax will vary depending on your local computer/operating system, and the details of your particular VISTA installation.

```
$ DEFINE V$PRODIR DRA0:[CCDEV.RELEASE3.HAM]
$ DEFINE V$CCDIR DRA1:[SCRATCH.USERNAME.ECHELLE]
$ VISTA :== R DRA0:[CCDEV.RELEASE3.BASE]VISTA
$ VISTA                          ! A bunch of blank lines are
                                 ! put here because VISTA asks
                                 ! for some input at the
                                 ! beginning, and we need to
                                 ! get past it before typing
                                 ! commands in.
 RP HAMPREP
 GO
 STRING FLATFILE 'FLATSUM.CCD'   ! File name of the summed
                                 ! flat-field.
 CONTINUE
 4                               ! File number for short dark.
 5                               ! File number for long dark.
 3                               ! Number of flat-fields.
 10                              ! First flat-field file.
 11                              ! Second "     "     ".
 12                              ! Third  "     "     ".
 18                              ! Standard star file.
 196                             ! Row position of an order.
 87                              ! Number of that order.
 STRD                            ! Output name for the order
                                 ! positions.
 SET OLDDARK=DARK                ! This keeps track of the dark
                                 ! count for later.
 RP HAMREDUCE
 GO
 SET DARK=OLDDARK                ! Reset the dark rate.
 STRING FLATFILE 'FLATSUM.CCD'   ! File name of the summed
                                 ! flat-field.
 CONTINUE
```

```
23                              ! File number to reduce.
1                               ! Use orders positions from
                                ! standard star.
STRD.ORD                        ! Name of file with order
                                ! positions.
MYOBS23                         ! Name of file for extracted
                                ! spectrum.
QUIT                            ! Don't forget this!!!
                                ! Otherwise the .LOG file
                                ! will keep growing until it
                                ! fills up your disk quota!!
$ EXIT
```

## 6.4 Fringe Removal

The fringes caused by interference in the CCD layers start to become apparent above about 6000Å. In usual Cassegrain spectroscopy the flat-field exposures correct for both the pixel-to-pixel sensitivity variations *and* the fringing effects. This is not the case for the open-decker flats obtained with the Hamilton spectrograph because of the considerable order overlap. They are excellent for removing the pixel-to-pixel sensitivity variations (which are only weakly dependent on the color of the incident light) but are not good at correcting the color-sensitive fringing effects. To do so you need to take a short-decker flat exposure in which the fringe pattern is identical to the pattern in the images of your objects.

The order structure of the short-decker flats prevents you from doing a straightforward division of the object frame by the short-decker flat (as was done for the open-decker flats). There are many schemes to solve this problem. The one we recommend here is to use *HAMREDUCE* and reduce the short-decker flats exactly the same way as you reduced the object from which you want to remove the fringes (*i.e.* same long-decker flat-field, short and long dark, extraction windows of the spectrum and the background, etc). The next and final step is to simply divide the object's extracted data by the extracted short-decker flat.

There are two potential problems with this method. First, the positions of the orders in the short-decker flat are not exactly the same as those of the object (due to a slightly different illumination). The shift is generally less than one pixel and is of no consequences if the extraction windows are large enough. Ideally, this method also requires that the profiles of the orders perpendicular to the dispersion are the

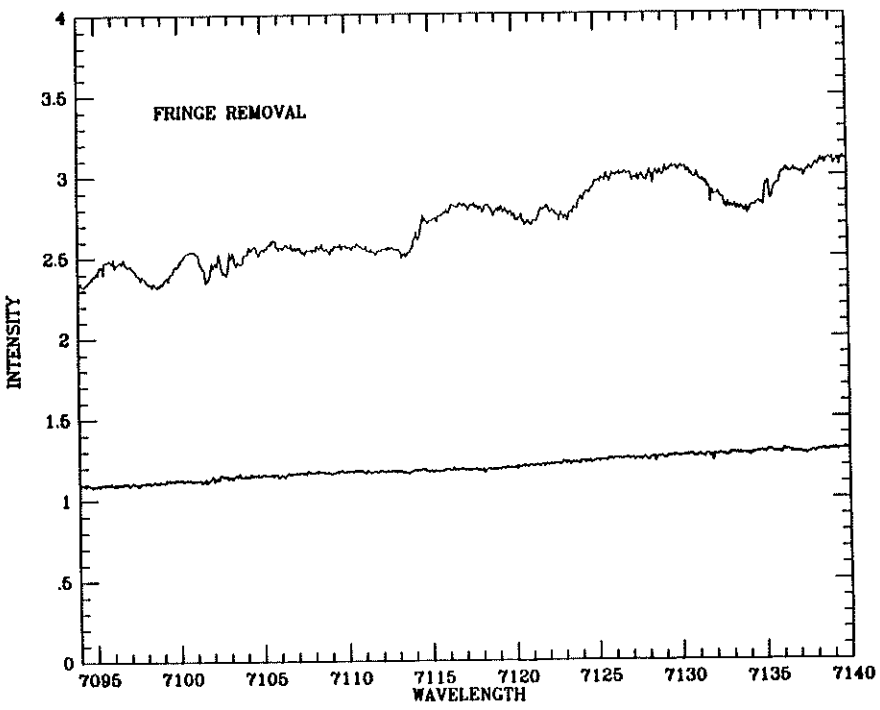



**Figure 6.1**: Effect of Fringing. Top curve shows the spectrum of a star before fringe-correction with the method described in the text, and the bottom curve shows the spectrum after fringe correction.

same for the flat field and for the object. As this is never exactly the case in reality one may wonder about the accuracy of this method. Figure 6.1 shows the spectrum of a star before (upper curve) and after (lower curve) being fringe-corrected using this method. The fringes have been almost completely removed from the data. This method has proved to give good results even at wavelengths where the fringes have amplitudes of 25 %.

## 6.5 Wavelength Calibration

When it comes the time to reduce a wavelength calibration (thorium-argon) image using *HAMREDUCE* you must set `BKW=0` to avoid doing any background subtraction on the image. We also suggest that you set `USEFLAT=0`, `USEFP=0`, and `USEDARK=0` in order to speed up the reduction of your image. After you are done extracting the thorium-argon spectrum use the VISTA command `EWAVE` to calibrate it. This

has the form:

```
GO: EWAVE 1 [XOFF=MMM] [TRACE] [TTY] [REJ=NNN] [FAST]
```

where the thorium spectrum is assumed to be in BUFFER 1 in this example. The optional keywords TTY and TRACE print out a list of the lines identified and the fit parameters, respectively. The keyword REJ is a hard rejection limit in mÅ determining the maximum residual allowed between the wavelength of a line and the wavelength determined from the fit; REJ=75 is generally sufficient. The keyword XOFF defines the distance in pixels of the CCD position from the nominal center of the echelle pattern. Its value can be determined from the grating rotation (written in the FITS header with the FITS card name GRATING; type BUF 1 FITS FULL to find the value of GRATING in your image) using

$$XOFF = 0.8791827 \times GRATING - 434172$$

or

$$XOFF = 0.8803644 \times GRATING - 434739$$

whether you are using the "old" TI CCD or the "new" NSF # 4 CCD, respectively. Finally, the FAST keyword invokes a faster matrix inversion solver for the fit. When this keyword is used, the wavelength calibration of frames containing about 600 lines takes 4–5 cpu minutes instead of 30–40 minutes. The accuracy reached by the wavelength solution is the same with or without the FAST keyword; therefore, there is absolutely no reason *not* to use the FAST keyword; it is simply a relic from the days of debugging the fitting algorithm.

Once a calibration has been performed on the thorium spectrum you should write the BUFFER containing the wavelength calibrated image (BUFFER 1 in the example above) to disk. You can copy the wavelength parameters onto any other spectrum using the command COPW:

```
GO: COPW 2 1
```

where again BUFFER 1 holds the thorium and BUFFER 2 holds the other image. Then if you type, e.g., PLOT 2 R=94 you will get a plot of order 94 with the appropriate wavelength scale along the x-axis.

Below is a sample DCL command file used to reduce and wavelength calibrate a thorium-argon image and copy the wavelength parameters onto another file. Again, the example uses directory syntax appropriate to the Lick VAX 11/780.

```
$ DEFINE V$PRODIR DRA0:[CCDEV.RELEASE3.HAM]
$ DEFINE V$CCDIR DRA1:[SCRATCH.USERNAME.ECHELLE]
$ VISTA :== R DRA0:[CCDEV.RELEASE3.BASE]VISTA
$ VISTA

  RP HAMREDUCE
  GO
  SET BKW=0                          ! Skip background subtraction.
  SET USEFP=0                        ! Don't correct for the fixed
                                     ! pattern.
  SET USEDARK=0                      ! Don't correct for dark
                                     ! counts.
  SET USEFLAT=0                      ! Don't use a flat-field.
  CONTINUE
  22                                 ! File to reduce.
  1
  STRD.ORD
  MYTHORIUM

  EWAVE 1 TTY TRACE XOFF=390 FAST REJ=75
  WD 1 MYCAL FULL                    ! Write out the calibrated
                                     ! image.
  RD 2 MYOBS23
  COPW 2 1
  WD 2 MYOBS23 FULL                  ! Write the calibrated
                                     ! observation.
  QUIT
$ EXIT
```

## 6.6 Flux Calibration

The purpose of the fluxing procedure is to put all of the orders of a frame on the same intensity scale. It is necessary to flux your data if you want to compare the relative strength of features at different wavelengths or if you wish to merge images which cover different parts of the echelle format. Note that flux calibration is *not* necessary if you only want to measure equivalent widths in an object with a well-defined continuum. In this case, the only thing you need to do is to normalize the

continuum using, for instance, the spline fitting routine `ISPLINE`.

The top-level procedure used for fluxing Hamilton data is called *HAMFLUX*. It first asks you for the name of the file containing the reduced standard star which will be used to determine the flux curve of each order. Next you need to provide the name of the file containing the wavelength parameters to be used. The procedure will then correct for the mean (smooth) atmospheric extinction and, if necessary, for the presence of atmospheric absorption features between $6800 \leq \lambda \leq 7400$Å (orders 76–83) by linearly interpolating the continuum over each atmospheric blend. This last step is essential since the flux points that will be used later were determined from standard stars that have been corrected for the presence of these atmospheric bands.

Next the procedure asks you for the the file name containing the flux points. These files are kept with the other flux calibration files (.FLX extension) in the default flux calibration directory. (In the example below, they are in the directory `DRAO:[CCD.SPEC]` on the Lick VAX 11/780. See Chapter 2 for details on default data file directories). The flux points listed in these files were determined from spectra obtained with the UV-Schmidt spectrograph at Lick. Each order should contain at least 30 of these flux points. Note that, although the flux points extend from 3600Å to 7650Å, it is in practice quite difficult to flux Hamilton data below about 4000Å (order $\gtrsim 140$) because of the presence of many strong lines in the spectrum of the standard stars.

At the end of the procedure, BUFFER 1 contains the fluxed standard star, BUFFER 4 contains the flux curves normalized to an exposure time of 1 second of all the orders, and BUFFER 5 contains the atmospheric correctors used to remove the atmospheric features from the spectrum of the standard star. A plot of BUFFER 4 along a given column will show a smooth curve except at a few orders (Figure 6.2, upper curve). These anomalous flux curves coincide with the position of strong absorption lines in the spectrum of the standard stars or the position of atmospheric lines that have not been corrected in the *HAMFLUX* procedure (e.g. order 87: H$\alpha$, order 117 or 118: H$\beta$, order 110: Mg I, order 91: atmospheric band, etc.). The procedure *FINTERP* corrects for those anomalous flux curves by replacing them with the result of the linear interpolation of the adjacent flux curves (Figure 6.2, lower curve). Once you are satisfied with the flux curves of all the orders you can then run the procedure *HAMFLUX2* which will flux calibrate your objects (reduced the same way as the standard star) using the flux curves produced by *HAMFLUX*.

Since it requires about 10 cpu minutes to create a flux image using *HAMFLUX* and about 2 cpu minutes to flux an object using *HAMFLUX2* (for a VAX 11/780

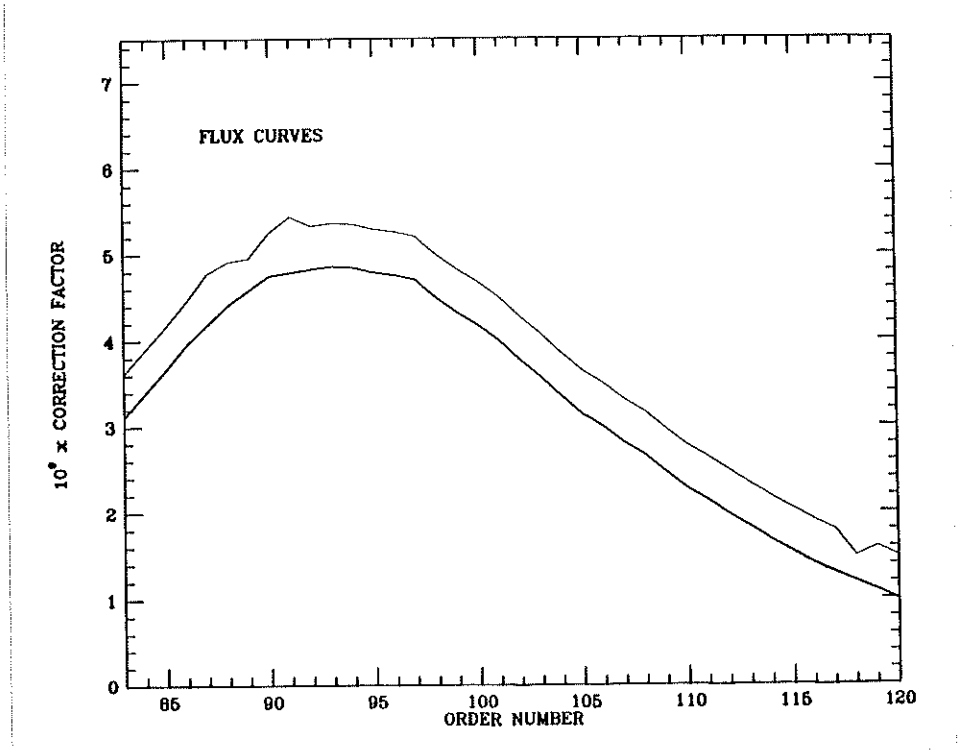



**Figure 6.2:** ...

running VMS 4.3) both of these procedures should probably be run in BATCH mode. Below is a sample DCL command file used to run *HAMFLUX*:

```
$ DEFINE V$PRODIR DRA0:[CCDEV.RELEASE3.HAM]
$ DEFINE V$CCDIR DRA1:[SCRATCH.USERNAME.ECHELLE]
$ VISTA :== R DRA0:[CCDEV.RELEASE3.BASE]VISTA
$ VISTA

  RP HAMFLUX
  GO
  FF.CCD                          ! File name of the extracted
                                  ! short-decker flat.
  STRD.CCD                        ! File name of the extracted
                                  ! standard star.
  THAR.WAV                        ! File name of the
                                  ! wavelength-calibrated Th-Ar.
  DRA0:[CCD.SPEC]STRD.HAMFLX      ! File name containing the flux
```

```
                                    ! points.
  WD 1 STRD.FLX FULL                ! Write to disk the fluxed
                                    ! standard star.
  WD 4 STRD.CRV FULL                ! Write to disk the flux curves.
  WD 5 STRD.COR FULL                ! Write to disk the atmospheric
                                    ! corrector.
  QUIT
$ EXIT
```

After interactively correcting STRD.CRV for any anomalous orders (using the procedure *FINTERP*) you can then run *HAMFLUX2*:

```
$ VISTA


  RP HAMFLUX2
  GO
  FF.CCD
  THAR.WAV
  OBJECT.CCD                        ! File name of the object to be
                                    ! fluxed.
  WD 6 OBJECT.FLX FULL              ! Write to disk the fluxed object.

  QUIT
$ EXIT
```

Finally, a word about the accuracy of these fluxing procedures. The Hamilton spectrograph should not be considered a good instrument to accurately measure *absolute* fluxes. Comparisons of the flux curves of standard stars obtained at the beginning and the end of the night have shown, however, that the accuracy of *relative* fluxes of features within the same order in a well-exposed standard star is about 2–3% while it is 5–30% (depending on the atmospheric conditions, guiding errors, position of the object in the sky, etc) for relative fluxes of features at the top and bottom of an Hamilton image. The uncertainty of the relative fluxes in the objects other than the standard stars greatly depends on the quality of the raw image. Non-linearity effects of the CCD at low light levels can for instance affect poorly-exposed frames. Accurate relative fluxing becomes in this case almost impossible.

## 6.7 Merging the Orders

Once your data are fluxed (or the continuum of your object normalized) you can start combining the orders together. Merging the orders of a well-exposed standard star or normalized object is generally quite straightforward. You first have to extract from the flux-calibrated image of your star the orders you wish to merge using the command EXTSPEC. For instance:

```
GO: EXTSPEC 10 1 ORD=87
GO: EXTSPEC 20 2 ORD=87
```

will extract order 87 from BUFFERs 1 and 2 and put it in BUFFERs 10 and 20, respectively. The command MASH should *not* be used since it doesn't preserve the wavelength information. The next step is to put all of the spectra you want to merge on the same linear or logarithmic wavelength scale with the command ALIGN (use for the value of the keyword DSP the *smallest* dispersion of your spectra so you do not lose any spectral information and don't forget to type the NOFLUX keyword) and finally use MERGE to actually combine the orders together:

```
GO: MERGE 10 20 NOZERO
```

Note that it is preferable to avoid using the NOMATCH keyword in MERGE and use instead appropriate values of W in ALIGN before merging. The intensity at the edges of the orders sometimes curves up or down so, in order to obtain a better match between different orders, it is recommended to exclude the first and last 20 pixels or so in an order using, for instance, the BOX command. If you wish to merge different frames of an object which is not a flux standard you generally need to multiply one of the images by a constant to obtain a better match in intensity between the frames. This "grey shift" corrects for any passing clouds, seeing changes, etc. between the observation of the standard star and your object.

# Chapter 7

# Dirty Tricks

The VISTA plotting routines are as general as we could make them, and as a result, there is a wide range of possibilities. Many of these take the form of what we call "dirty tricks" — little hooks that can enhance the appearance of a plot, without trying to re-write huge sections of code to add custom features. These are some of our favorites. Be warned, however, that this is genuine "black belt" stuff.

## 7.1 User Defined Axes for Plots

The `USER` keyword in the `PLOT` command may be used to make custom scalings and labelling of the user coordinates. They work by way of the coordinate cards provided in the FITS header. These cards are as follows, one set for each axis (row and columns):

```
For Columns:  CRVAL1, CRPIX1, CDELT1, CTYPE1

   For Rows:  CRVAL2, CRPIX2, CDELT2, CTYPE2
```

The internal coordinates of any image are the so-called "pixel" coordinates, simply: Columns and Rows. Images displayed by VISTA are always in pixel coordinates by default (except for wavelength calibrated spectra). The user may define coordinates which have some physical meaning (like angular measure in arcseconds or wavelength in angstroms). These are called "world coordinates" or "physical coordinates" (in the sense that their units have some meaning in the world outside the machine). The transformation between "world" and "pixel" coordinates is through the FITS header cards shown above. The tranformation equations are:

```
X(world) = CRVAL1 + CDELT1*(Column - CRPIX1)

Y(world) = CRVAL2 + CDELT2*(Row - CRPIX2)
```

Where X and Y axes correspond to Columns and Rows respectively. In general, the FITS coordinate cards correspond to:

```
CRVALn  =  Origin along Axis n in World Coordinates
CRPIXn  =  Origin along Axis n in Pixel Coordinates
CDELTn  =  Scaling Factor giving relation between Pixels
           and World Coordinates
n=1 for columns
n=2 for rows
```

The units of the world coordinates are identified by the `CTYPE1` and `CTYPE2` FITS header cards for Columns and Rows respectively. These contain character strings which give the units. The units of the pixel data itself (*e.g.*, pixel intensities) are given by the `BUNIT` FITS header card.

Confusing? How about an example. You have a 2-D image in which Columns are oriented East towards West in the sky (RA), and Rows are along North towards South (Dec). After some processing, the image pixels measure flux in millijanskys (mJy) at 2 microns. You want to plot profiles of the image of the object (a planetary nebula) through the center along both RA (by plotting along Row 128), and Dec (by plotting along Column 67). The pixel scale in both rows and columns is 0.″54/pixel, and you want the origin to be the central star of the planetary, which is at `(C,R)=(67,128)`. The transformation equations between image pixels (C,R) and world coords (RA,Dec) is then:

```
 RA = 0.0 - 0.54*(C - 67)
Dec = 0.0 - 0.54*(R - 128)
```

Note that because RA increases from West to East, and column number increases in the opposite direction, the scale factor is $-0.54$ rather than $+0.54$. Since the Row number increase North to South (top to bottom), the scale factor is $-0.54$ as for columns. This means that the necessary FITS coordinate cards you must load are:

```
CRVAL1 = 0.0          CRVAL2 = 0.0
CRPIX1 = 67           CRPIX2 = 128
CDELT1 = -0.54        CDELT2 = -0.54
```

And, to get the units to be plotted right, you would load these units cards into the header:

```
CTYPE1 = 'Right Ascension (arcsec)'
CTYPE2 = 'Declination (arcsec)'
BUNIT = 'Flux (mJy)'
```

Since the units cards are used for keeping track of the image internally to VISTA, you need to proceed with a little caution. Suppose the image of interest is in Buffer 1. We want to copy it into Buffer 2 and then start monkeying with the FITS cards to get our plots. Use the following procedure:

```
GO: COPY 2 1
GO: FITS 2 FLOAT=CRVAL1 0.0
GO: FITS 2 FLOAT=CRPIX1 67.
GO: FITS 2 FLOAT=CDELT1 -0.54
GO: FITS 2 FLOAT=CRVAL2 0.0
GO: FITS 2 FLOAT=CRPIX2 128.
GO: FITS 2 FLOAT=CDELT2 -0.54
GO: FITS 2 CHAR=CTYPE1 'Right Ascension (arcsec)'
GO: FITS 2 CHAR=CTYPE2 'Declination (arcsec)'
GO: FITS 2 CHAR=BUNIT 'Flux (mJy)'
GO: COPY 2 2
```

The FITS cards for the pixel-to-world coordinate transformations have all been loaded. The command `COPY 2 2` makes VISTA happy again, as it now knows where the image is. Now, make the plots, first along RA (Columns), then along Dec (Rows):

```
GO: PLOT 2 R=128 USER
GO: PLOT 2 C=67 USER
```

In order to be as general as possible, I chose an example in which both axes of the image were changed. If the image was a spectrum (1-D image), then you would only have changed the `CRVAL1`, `CRPIX1`, `CDELT1`, `CTYPE1`, and `BUNIT` cards in the FITS header. In the example given, I've used familiar angular coordinates. They could just as easily have been microns along a slit and angstroms of wavelength

perpendicular to it. There are no limitations to the transformation or the units, except that the transforms be linear in one and only one pixel coordinate (true, there is a FITS card called `CROTAn` which handles rotations of the given axis, but VISTA doesn't handle them in its current incarnation).

It is important to note that VISTA uses the FITS header cards in a somewhat non-standard way internally. This is discussed in detail in Appendix D. The bottom line of this and related dirty tricks which deal with changing these particular FITS cards is to be very circumspect. A little lack of caution, and VISTA could crash out from under you — most often with a fatal virtual memory error (VMS will tell you, Unix will just have it wink out of existance with a "bus error, core dumped" message).

## 7.2 Putting Labels on Plots

The `INT` keyword of the `PLOT` command puts you temporarily into the interactive MONGO command environment, with the specified plotting data loaded into the X and Y plotting arrays. Thus, you can have the full power of MONGO to make a custom plot at your disposal, without the intermediary of having to write the data (somehow) into a 2 column text file of X and Y points and then leaving VISTA.

The interactive MONGO prompt is an asterisk (*). Once you have the MONGO prompt, you can enter all valid interactive MONGO commands. To leave interactive MONGO and return to VISTA, type the `END` command.

The X and Y data are pre-loaded by the `PLOT` subroutine. Pre-prepared MONGO macros may be read in using the MONGO command `READ`; do not use the `INPUT` command.

There are a number of ways to put labels on a plot once you are in the interactive MONGO command level. The `PUTLABEL` command is the best way, specifying the location of the label with `RELOCATE`. For example: you wish to label the H$\alpha$ and H$\beta$ emission lines in a spectrum of a planetary nebula (with no redshift). The spectrum is in buffer 10, and you want the plot to run from 4000 to 7000Å, with the lowest intensity at 0.0, and the highest intensity autoscaled. The labels are to be centered over the lines. To do this, you would issue the following sequence of VISTA and MONGO commands. The VISTA commands are indicated with the `GO:` prompt, and the MONGO commands with the * prompt.

The first pass is to only plot on the screen to get the location of the labels right. The second pass, you repeat the relocate and putlabels that gave the best

results, this time blind (*i.e.*, you don't see a plot or the results of the plot), and the hardcopy is generated.

```
        GO: PLOT 10 XS=4000 XE=7000 MIN=0. INT

        * RELOCATE 4861 1.5E-15
        * PUTLABEL 5 H\gb
        * RELOCATE 6563 5.0E-15
        * PUTLABEL 5 H\ga
          .
          .
      (iterate until the labels look right)
          .
          .
        * END

(now, do it again, but make a hardcopy)

        GO: PLOT 10 XS=4000 XE=7000 MIN=0. INT HARD
        * RELOCATE 4861 1.5E-15
        * PUTLABEL 5 H\gb
        * RELOCATE 6563 5.0E-15
        * PUTLABEL 5 H\ga
        * END

        GO:
```

While in interactive MONGO, you can erase the screen, re-scale the plot, put down your own axis labels and so forth, as if you had read in the data from a external file using the XCOLUMN and YCOLUMN commands in MONGO. If you are on a terminal with an interactive cursor, then you can use the CURS command in MONGO to make placing the labels faster. The possibilites are virtually endless.

If you are unfamiliar with the MONGO package, a copy of the Lick MONGO User's Manual has been provided with the VISTA distribution package, and should be available from your VISTA custodian.

## 7.3 Plotting RA and DEC on Contour Maps

The CONTOUR command also has a USER keyword, but since this necessitates fooling with FITS header cards used as internal array index pointers by VISTA, some care must be taken to avoid crashing VISTA. The use of the FITS header cards is identical to that described above for user defined coordinates on plots. Consider the planetary nebula example from above. We now wish to make a contour map with the origin on the central star centroid which has been found to be at (C,R)=(67.2,127.9), and with contours spaced every 2 mJy beginning at 1 mJy. The image is in Buffer 1. Copy it into Buffer 2, and load the appropriate FITS cards as follows:

```
GO: COPY 2 1
GO: FITS 2 FLOAT=CRVAL1 0.0
GO: FITS 2 FLOAT=CRPIX1 67.2
GO: FITS 2 FLOAT=CDELT1 -0.54
GO: FITS 2 FLOAT=CRVAL2 0.0
GO: FITS 2 FLOAT=CRPIX2 127.9
GO: FITS 2 FLOAT=CDELT2 -0.54
GO: FITS 2 CHAR=CTYPE1 'Right Ascension (arcsec)'
GO: FITS 2 CHAR=CTYPE2 'Declination (arcsec)'
GO: COPY 2 2
```

above, and then type the following command:

```
GO: CONTOUR 2 LOW=1.  DIFF=2.  USER TITLE
```

to make the contour map. An example of a contour map (not of a planetary nebula) made this way with VISTA can be found in Pogge (1988, *Ap.J.*, **328**, 519). A simple proof that a VISTA dirty trick is potentially acceptable by the editor of the *Ap.J.*!

## 7.4 Preparing Images with Boxes and Scale Bars

Color photography of the images displayed on the color monitor can make very nice, impressive slides for presentation. A few simple hooks were built-in to the

TV command to give users some latitude in preparing images for display. Actual tricks for photographing off the monitor are not discussed as we cannot seem to get a consensus among the photographers at Lick about how to do it. The only common points are: (a) all agree to use color slide film, ASA100, and (b) to use a long focal length lens so that the curvature of the monitor screen doesn't make the edges go out of focus. Beyond that advice, you're pretty much on your own as to technique.

Few things are more irritating than to have a speaker show slide after slide of objects without some identifying label. Great stuff, to be sure, but what is the name of that object? Also, most images don't have scale bars. So someone has to interrupt with "what is the image scale we're looking at?". The problem here is one of proper form. You've been staring at these images for months. You know them by name and the image details intimately. But, don't forget that (1) your audience has probably never seen these objects before, (2) isn't necessarily up on their names, (3) doesn't know a thing about your instrument (like image scale), and (4) isn't going to know any of it because you told them before flashing up the slide. Put it all up in front of them so they can see it for themselves, and it will be greatly appreciated.

Built into VISTA's TV command are two keyword called TITLE and TOP which facilitate putting up image titles. TITLE puts the object title (FITS header OBJECT card) on the TV display, without plotting the tickmarks or numbers along the axes. TOP puts the label over the image, rather than below it (which is default). It's nice to have a box (border) around the image, but the axis ticks and labels giving pixel numbers cause too much clutter. Face it AEDs don't do so hot with axis labels despite our best efforts unless the image is really big. The trick here is to have a border around the image without the ticks. This is accomplished using the TVBOX command. How about an example.

Suppose you want to display the image in Buffer 1 with a border around it and the title "A Cool Galaxy" centered over the image. Let's say that the image zero and span are '0.0' and '1200.0' respectively, and you want to use the inverse black and white color map (photonegative), and clip the colormap roll-over. You would do this as follows:

```
GO: CH 1 'A Cool Galaxy'
GO: TV 1 Z=0.0 L=1200.0 CF=IBW CLIP TITLE TOP
GO: BOX 1 SR=SR[1] SC=SC[1] NR=NR[1] NC=NC[1]
GO: TVBOX BOX=1
```

We're using TVBOX with a box which exactly surrounds the image displayed to make our border for us. The keywords TITLE TOP on the TV command tell VISTA to display the image without axis ticks and labels, with the image title centered over the top of the image. Note that it was necessary to change the object title to our desired title using the CH command before displaying the image.

A scale bar, say one indicating 10″ on the sky would be nice too, it gives your viewer and idea of the size. In Appendix B, a short command procedure called "BAR" has been given which will draw a scale bar on the image for you. Put this procedure into a file called "BAR.PRO" in your procedure directory. Since we are clipping the image color scale with the TV command at L=1200.0, we will set the "intensity" of the bar to be 1199.0 (1200–1). Pick a place for the bar to start (it will be drawn to the right of the place you pick) in rows and columns. I'm going to pick ROW=300, COL=18. For the image in question, 10 arcseconds on the sky corresponds to 17 pixels on the image. Use this procedure to draw the bar into the image, and redisplay the image. BAR will ask for the place to draw the bar, its length, and its intensity. NOTE: BAR will change data values in the image, not just draw on the screen. Best thing to do is copy the image into another buffer before hitting it with BAR.

```
GO: COPY 2 1
GO: CALL BAR
IMAGE NUMBER ? : 2
PUT BAR IN WHAT IMAGE ROW ? : 300
STARTING COLUMN ? : 18
LENGTH ? : 17
INTENSITY VALUE ? : 1199.
GO: TV 2 Z=0.0 L=1200.0 CF=IBW CLIP TITLE TOP
GO: BOX 2 SR=SR[2] SC=SC[2] NR=NR[2] NC=NC[2]
GO: TVBOX BOX=2
```

Now, if this is what you want, get your camera and shoot away.

# Appendix A

# VISTA Command Summary

The following is a summary of standard VISTA commands, intended to be used as a quick reference guide. Commands are listed by function. Detailed descriptions of each of the commands may be found in the VISTA Help Manual, or by typing `HELP <command name>`.

## A.1 Stopping VISTA

The end a VISTA session, type:

```
QUIT
```

## A.2 VISTA Command Syntax

The basic command syntax for all VISTA commands is as follows:

```
COMMAND imbuf req#1 req#2 ... [opt#1] [opt#2] ...
```

Where:

`COMMAND` is the command name. You do not need to type the full name. Commands may be abbreviated to the shortest unambiguous form.

`imbuf` is the image buffer the command is to operate on. It is an integer. In some cases, more than one image may be required.

`req#n` are the required keywords. These may NOT be abbreviated.
`[opt#n]` are the optional keywords. These may NOT be abbreviated either.

Command Keywords:

The keyword syntax used throughout is:

```
uservalue1 KEY1 KEY2=uservalue2 KEY3=user3,user4
```

Capitalized words are command keywords. Lowercase words are numerical or string values that the user must supply.

Extended Commands:

Commands can be extended to more than one line by ending the line with the "|" character (the "pipe" character, not a lowercase l), hitting ⟨RETURN⟩ and entering the rest of the command on the next line. Commands are limited to 256 characters in length.

Multiple Commands:

Multiple commands can be chained together on a command line separated commands by semicolons (;).

Command History:

The command history is a list of the last 20 VISTA commands. It allows you to see what you have typed in, and with the command recall facility (% command below), you can recall any command listed by `HISTORY`.

```
HISTORY [output redirection]
```

Repeating and/or Modifying Previous Commands:

```
%             repeat last command
% newkey      repeat last command with new keyword "newkey"
%xyz          repeat last command beginning with pattern "xyz"
%xyz newkey   as above, but add new keyword "newkey"
```

Editing Previous Commands:

```
EDIT
```

Defined a Command Synonym (Alias):

```
ALIAS [synonym] ['command'] [output redirection]
```

Remove a Command Synonym:

```
UNALIAS synonym
```

On-Line Command Help Facilities:

```
HELP [subjects] [ALL] [output redirection]
```

or

```
? command
```

Appropriate Command Search:

```
APROPOS topic_1 topic_2 ...
```

`APROPOS` suggests possible VISTA `HELP` topics based on a keyword search. The topics are the principle sections of the help manual. Similar to the Unix `apropos` command.

## A.3 VISTA User-Defined Variables

### A.3.1 Defining VISTA variables

Definition of a VISTA variable:

```
SET variable_name=value
```

or

```
variable_name=value
```

Note that there can be NO SPACES separating the `=` from either the variable name or the value. Variable names in VISTA are insensitive to case, so `IGGY` is the same as `iggy` is the same as `IgGy`, and so forth. Variable names may contain both characters and numbers.

### A.3.2 Arithmetic Operations among VISTA Variables

The following table lists the arithmetic operations supported by VISTA.

| Symbol | Function | Examples | |
|---|---|---|---|
| + | addition | B+C | 2.0+5 |
| - | subtraction | X-Y | 123-54 |
| * | multiplication | X1*X2 | 56.4*60.0 |
| / | division | A/B | 125/0.54 |
| - | negative sign | -X | -12.3 |
| ^ | exponentiation | A^0.5 | 12^2 |
| = | equate | A=B | A=B=3.5 |

Note that there must be no spaces between the operator and the operands. For example:

```
    X=A+B-C          is valid
but
    X = A + B - C    is not
```

Expressions are evaluated in the same order as they are in Fortran. You may change the order of evaluation using parentheses "()". For example:

```
    X=(B+0.53)*10^(45.6/(A+5))
```

Arithmetic expressions among VISTA functions may be of arbitrary complexity. The only requirements are that all variables appearing on the right hand side of equations be previously defined and that parentheses be balanced. NOTE: SPACES MAY NOT APPEAR ANYWHERE WITHIN AN ARITHMETIC EXPRESSION.

### A.3.3 VISTA Functions

VISTA supports the following functions. Arguments of functions may be numbers or previously defined VISTA variables (as appropriate). Functions can appear as arguments of other functions, and can contain expressions of arbitrary complexity. Spaces are not allowed anywhere within a function.

Arithmetic Functions:

```
INT[E]          nearest integer to the expression E
ABS[E]          absolute value of E
MOD[E,I]        E modulo I (remainder of E/I)
IFIX[E]         integer part of E (truncation)
MAX[E,F]        the larger of E or F
MIN[E,F]        the smaller of E of F
LOG10[E]        Log base 10 of E
LOGE[E]         Log base e ("natural log") of E
EXP[E]          e raised to the power E
                  (Use ^ for all other exponentiations)
SQRT[E]         square root of absolute value of E
RAN[A,B]        returns a random number between A and B
```

Trigonometric Functions:

```
SIN[E]          sine of E (E in radians)
SIND[E]         sine of E (E in degrees)
COS[E]          cosine of E (E in radians)
COSD[E]         cosine of E (E in degrees)
ARCTAN[E]       arctan of E, returns radians
ARCTAND[E]      arctan of E, returns degrees
```

Functions to Extract Image FITS Header Information:

```
NR[B]       number of rows of the image in buffer B.
NC[B]       number of columns of ...
SR[B]       starting row of ...
SC[B]       starting column of ...
EXPOS[B]    exposure time of ...
RA[B]       right ascension in seconds of time of ...
DEC[B]      declination in seconds of arc of ...
ZENITH[B]   zenith distance in radians of ...
UT[B]       UT time of start of exposure in hours of ...
```

Advanced Image Pixel Functions

```
GETVAL[I,R,C]   returns the value of the pixel at row R and
                column C in image I.
SETVAL[I,R,C,V] returns the value of the pixel at row R and
                column C in image I, then sets the value of
```

```
                  that pixel to be equal to V.
WL[I,P]           returns the wavelength of pixel P in
                  image I (a wavelength calibrated spectrum).
PIX[I,W]          returns the pixel number corresponding to the
                  wavelength W in image I.
```

### A.3.4 Variable Related Commands

Define a Variable or Set of Variables:

```
SET var1=value1 [var2=value2] ... [var15=value15]
```

Note that you may omit SET if desired.

Evaluate an Expression and Print the Result:

```
TYPE expression1 [expression2] ... [expression15]
```

### A.3.5 String Variables

To Define String Variables:

```
STRING name ['format string'] [expressions]
STRING name '?query'
```

Print List of All User-Defined Strings:

```
PRINT STRINGS [output redirection]
```

Substituting String Variables into a Command Line:

```
COMMAND {STRING} [rest of command]
```

## A.4 Tape and Disk Input/Output

### A.4.1 FITS Tape Input/Output

Mounting and Dismounting Tapes:

```
MOUNT [UNIT=n] [BPI=m]

DISMOUNT [UNIT=n]
```

Listing of Tape Contents:

```
TDIR [comment] [UNIT=n] [BRIEF] [output redirection]
```

Reading and Writing Images:

```
RT imbuf tape_number [UNIT=n] [NOMEAN]

WT imbuf [PDP8] [ZERO=z SCALE=s] [NOAUTO] [UNIT=n]
         [BITPIX=m]
```

Initialize or Edit a VISTA FITS Tape:

```
INT [firstimage#] [UNIT=n]
```

### A.4.2 VISTA Disk Format Image Input/Output

List the Default Disk Directories:

```
PRINT DIRECTORIES
```

Change the Default Directory Assignment:

```
SETDIR IM [DIR=directory_name] [EXT=extension]
```

and

```
SETDIR SP [DIR=directory_name] [EXT=extension]
```

IM is for images, SP is for spectra.

Read an Image from Disk:

```
RD imbuf filename [SPEC]
```

Write an Image to Disk:

```
WD imbuf filename [SPEC] [FULL] [ZERO=z SCALE=s]
```

Write a Spectrum into an ASCII Text File on Disk:

```
PRINT spbuf SPEC >filename
```

## A.5 VISTA Buffers

Listing of VISTA Buffers:

```
BUFFERS [bufs] [FULL] [FITS[=param]] [output redirection]
```

Copying Images among VISTA Buffers:

```
COPY imbuf source [BOX=b]
```

NOTE: The syntax of COPY is:

```
COPY <TO this buffer>  <FROM this buffer>
```

Deleting a VISTA Buffer:

```
DISPOSE [buf] [buf2] [buf3] [...] [ALL]
```

Create a Blank Image:

```
CREATE imbuf [BOX=b] [SR=sr] [SC=sc] [NR=nr] [NC=nc]
             [CONST=c]
```

Changing Name of an Image in a VISTA Buffer:

```
CHANGE buf 'new_name'
```

Change or Remove Individual FITS Header Cards:

```
FITS buf [FLOAT=name] float_value
```

or

```
FITS buf [INT=name] integer_value
```

or

```
FITS buf [CHAR=name] 'character string'
```

or

```
FITS buf [REMOVE=name]
```

Edit a FITS Header:

```
HEDIT buf
```

You will have the FITS header placed in the default screen editor appropriate to your installation.

## A.6 Display of Images and Spectra

### A.6.1 Image Display and Interaction

Display an Image on a Color Monitor:

```
TV imbuf [span] [zero] [L=span] [Z=zero] [BOX=b] [BW]
         [CF=colormap] [NOLABEL] [LEFT] [RIGHT] [NOERASE]
         [CLIP]
```

Load or Define a Color Map:

```
COLOR [CF=filename] [BW] [INV]
```

Put an up Interactive Cursor on an Image Display:

```
ITV
```

The operation of `ITV` depends on your installation of VISTA.

Overlay a `BOX` on a Displayed Image:

```
TVBOX [BOX=b] [SIZE=s PIX=r,c]
```

### A.6.2 Image Hardcopy

Grayscale Hardcopy on a Versatec:

```
VTEC imbuf [span] [zero] [L=span] [Z=zero] [BOX=b]
           [OLD] [INV] [CLIP]
```

Gray Halftone Hardcopy on a PostScript Device:

```
POSTIM imbuf [L=span] [Z=zero] [BOX=b] [CLIP] [INV]
             [TITLE] [PORT]
```

### A.6.3 Line Graphics and Spectrum Plotting

Plot an Image Row, Column or a Spectrum:

```
PLOT imbuf [R=n] [C=n] [CS=c1,c2] [RS=r1,r2] [MIN=f] [MAX=f]
           [XS=f] [XE=f] [LOG] [SEMILOG] [GRID] [INFO] [HARD]
           [OLD] [SWAPXY] [PIXEL] [NOERASE] [NOPRINT] [HIST]
           [USER] [NOLABEL] [INT]
```

Keywords ([R=n] thru [RS=r1,r2]) are omitted if imbuf contains a spectrum.

Plot an Image Contour Map:

```
CONTOUR imbuf [BOX=b] [LEVELS=(L1,L2,...)] [LOW=l] [RATIO=r]
              [DIFF=d] [FID=l] [SCALE=s] [USER] [TITLE] [DASH]
              [TR=(X0,X1,X2,Y0,Y1,Y2)] [EXACT] [HARD]
              [NOERASE] [NOPRINT] [FULL] [NOLABEL]
```

Specify or Change the Default Graphics Devices:

```
TERM  (none)  [TERMINAL=vterm]  [HARDCOPY=vhard]
```

## A.7 Marking Image Segments and Pixels

### A.7.1 VISTA Image BOXes

Define a BOX or Image Subsection:

```
BOX box_num [NC=n] [NR=n] [CR=n] [CC=n] [SR=n] [SC=n]
```

Print a List of Current Boxes:

```
PRINT BOX
```

Overlay a BOX on a Displayed Image:

```
TVBOX [BOX=b] [SIZE=s PIX=r,c]
```

### A.7.2 Image Masks

Mask Image Regions:

```
MASK  [R=r1,r2] [C=c1,c2] [BOX=b] [PIX=r,c]
```

Un-Mask Image Regions:

```
UNMASK  [R=r1,r2] [C=c1,c2] [BOX=b] [PIX=r,c]
```

Create an Image to Display Current Image Mask:

```
MASKTOIM imbuf [BOX=b] [SR=sr] [SC=sc] [NR=nr] [NC=nc]
```

Store the Current Image Mask on Disk:

```
SAVE MASK=filename
```

Retrieve an Old Image Mask from Disk:

```
GET MASK=filename
```

NOTE: Masks may also be created using the CLIP command (see below).

## A.8 Image and Spectrum Arithmetic

### A.8.1 Basic Arithmetic

The basic image arithmetic commands are:

```
ADD      image#1 [image#2] [CONST=c] [BOX=B] [DR=dr] [DC=dc]
SUBTRACT image#1 [image#2] [CONST=c] [BOX=B] [DR=dr] [DC=dc]
MULTIPLY image#1 [image#2] [CONST=c] [BOX=B] [DR=dr] [DC=dc]
DIVIDE   image#1 [image#2] [CONST=c] [BOX=B] [DR=dr] [DC=dc]
                 [FLAT]
```

The basic two-image operations are:

```
ADD image#1 image#2     implies   image#1 = image#1 + image#2
SUB image#1 image#2     implies   image#1 = image#1 - image#2
MUL image#1 image#2     implies   image#1 = (image#1)(image#2)
DIV image#1 image#2     implies   image#1 = (image#1)/(image#2)
```

The basic one-image operations are:

```
ADD image CONST=c       implies   image = image + c
SUB image CONST=c       implies   image = image - c
MUL image CONST=c       implies   image = c(image)
DIV image CONST=c       implies   image = (image)/c
```

### A.8.2 Advanced Image Arithmetic

| | |
|---|---|
| Square Root of an Image: | `SQRT imbuf [SGN] [SIGN]` |
| Log (Base 10) of an Image: | `LOG imbuf` |
| Exponentiate ($e^x$) an Image: | `EXP imbuf` |
| Tangent of an Image (degrees): | `TAN imbuf` |
| ArcTan of an Image (returns degrees): | `ARCTAN imbuf [0to180]` |

## A.9 Image Statistics

Mean of the Pixel Values:

```
MN imbuf [NOBL] [BOX=b] [NOZERO] [MASK] [PIX=p] [SILENT]
         [W=w1,w2]
```

Full Image Pixel Statistics:

```
ABX imbuf box1 [box2] ... [ALL] [W=w1,w2] [SILENT] [MASK]
              [TOTAL=var] [MEAN=var] [HIGH=var] [LOW=var]
              [HIGH_ROW=var] [HIGH_COL=var] [LOW_ROW=var]
              [LOW_COL=var] [SIGMA=var] [output redirection]
```

Either a list of box numbers or the keyword `ALL` must be given, otherwise `ABX` will simply terminate without producing output.

Compute "Sky" Level of an Image:

```
SKY imbuf [BOX=b] [SILENT] [CORNERS] [MAX=c]
```

SKY returns the Mode of the pixel intensity distribution.

Plot a Histogram of Image Pixel Intensities:

```
HISTOGRAM imbuf [BOX=b] [NOLOG] [BIN=bin] [XMIN=xmin]
                [XMAX=xmax] [YMIN=ymin] [YMAX=ymax] [HARD]
```

## A.10 Image Processing

### A.10.1 Image Size, Orientation, and Position

Window an Image to a Smaller Size:

```
WINDOW imbuf BOX=b
```

Merge Images or Spectra:

```
MERGE im1 im2 im3 im4 ... [NOMATCH]
```

Compress an Image or Spectrum:

```
BIN imbuf [BIN=b] [BINR=br] [BINC=bc] [SR=sr] [SC=sc] [NORM]
```

Rotate an Image:

```
ROTATE imbuf [LEFT] [RIGHT] [UD] [PA=degrees] [BOX=b]
```

Reverse the Order of Rows or Columns of an Image:

```
FLIP imbuf [ROWS] [COLS]
```

Shift an Image:

```
SHIFT imbuf [DC=f] [DR=f] [SINC] [NORM] [RMODEL=i] [CMODEL=i]
            [MEAN]
```

## A.10.2 Changing Image Pixel Values

Correct an Image for Baseline Subtraction Noise:

```
BL imbuf [JUMP]
```

Changing Individual Pixel Values:

```
SET X=SETVAL[imbuf,row,col,newvalue]
```

Replace Pixels Outside a Given Intensity Range:

```
CLIP imbuf [MAX=f] [MIN=f] [VMAX=f] [VMIN=f] [BOX=b]
           [MASK] [MASKONLY]
```

Gaussian or Boxcar Smooth an Image or Spectrum:

```
SMOOTH imbuf [FW=f] [FWC=f] [FWR=f] [BOXCAR]
```

Image Median Filter and Pixel Zapper:

```
ZAP imbuf [SIG=f] [SIZE=s] [SIZE=r,c] [BOX=b] [TTY]
```

Interactive Pixel Zapper:

```
TVZAP [SIG=f] [SEARCH=s] [TTY]
```

Compute Median of Several Images:

```
MEDIAN imbuf im1 im2 im3 [im4 im5 ...] TTY
```

Fit a Surface to an Image:

```
SURFACE imbuf [BOX=b] [PLANE] [SUB] [LOAD] [DIV] [MASK]
        [NOZERO] [PIX=N] [output redirection]
```

Replace an Image by a Spline (Non-Interactive):

```
SPLINE imbuf [R=r1,r2,...] [C=c1,c2...] [W=w1,w2,...]
             [AVG=a] [SUB] [DIV]
```

Cross-Correlate two Images or Spectra:

```
CROSS imbuf image#1 image#2 [BOX=b] [RAD=r] [RADR=r] [RADC=c]
```

Interpolate across rows, columns, or masked pixels:

```
INTERP imbuf [BOX=b1,b2,...] [COL] [ROW] [ORD=n] [AVE=a]
             [MASK]
```

## A.11 Spectroscopic Reduction Routines

Throughout this section, the keyword spbuf will be used to refer to a VISTA buffer which contains (or will contain) a spectrum, and the keyword imbuf will be used to refer to a VISTA buffer which contains (or will contain) a 2-D image.

### A.11.1 Spectrum Extraction from 2–D Images

Simple Spectrum Extraction:

```
MASH spbuf imbuf SP=i1,i2 [BK=b1,b2] [COL] [COL=c1,c2]
                 [ROW=r1,r2] [SKY=s] [NORM] [REFLAT] [SUB]
                 [MASK]
```

The basic syntax of MASH is:

```
MASH <into this spectrum> <from this 2-D image>
```

Optimal Weighting Scheme Extraction:

```
EXTRACT spbuf imbuf SP=s1,s2 BK=b1,b2 BK=b3,b4 [SKY=s]
              [VAR=v] [SUB] [SORDER=sord] [PORDER=pord]
              [RONOISE=r] [EPERDN=eperdn]
```

Extraction Using a Spectrum Centroid Map:

```
SPECTROID spbuf imbuf [SP=s1,s2] [BK=b1,b2] [SPW=ds] [BKW=db]
              [MODEL=m] [DLOC=d] [LOC=r] [NOSHIFT] [FIT=p1,p2]
              [TRACE] [SELF] [LOAD] [NOMASH] [TAGALONG] [TAG]
              [MOMENTS]
```

### A.11.2 Wavelength Calibration

Identify Lines in a Wavelength Calibration Spectrum:

```
LINEID spbuf [FILE=lineid_file] [ADD] [TTY] [INT]
             [CEN=wavec] [DISP=disp] [output redirection]
```

Compute Wavelength Calibration from LINEID Output:

```
WSCALE spbuf [ORD=n] [TTY] [INT] [output redirection]
```

Copy a Wavelength Scale between Spectra:

```
COPW spbuf source [source2]
```

Put a Spectrum on a New Wavelength Scale:

```
ALIGN spbuf DSP=disp W=(lam,pix) [LOG] [LGI] [MS=n] [FLIP]
            [V=f] [Z=z] [DP=dp] [SILENT] [RED=z] [DERED=z]
```

Correct Wavelength Scale using Night Sky Lines:

```
SKYLINE sp1 [sp2] [sp3] [sp4] ... [sp15] [INT]
```

### A.11.3 Flux Calibration

Compute a Flux Curve using Standard Star Spectra:

```
FLUXSTAR spbuf [fluxfile] [AVE] [POLY=n] [WT=w] [SYSA] [SYSC]
```

Flux Calibrate a Spectrum:

```
FLUX spbuf
```

Correct for Atmospheric Extinction:

```
EXTINCT spbuf [CTIO]
```

### A.11.4 Spectrum Fitting and Stretching

Fit a Polynomial to a Spectrum:

```
POLY spbuf ORD=n [SUB] [DIV] [LOAD]
```

Fit a Spline to a Spectrum:

```
Non-Interactive:
      SPLINE spbuf [R=r1,r2,...] [C=c1,c2...] [W=w1,w2,...]
                   [AVG=a] [SUB] [DIV]

Interactive:
      ISPLINE spbuf [XY] [AVG=a] [SUB] [DIV] [HIST]
```

Stretch a Spectrum into a 2-D Image:

```
      STRETCH imbuf spbuf [VERT] [HORIZ] [SIZE=s] [START=st]
```

### A.11.5 Echelle Reduction Routines

Wavelength Calibrate an Echelle Image:

```
      EWAVE imbuf [XOFF=x0] [PORD=nw] [MORD=nm] [REJ=rej]
                  [TTY] [TRACE]
```

Copy an Order of an Echelle Image into a Spectrum:

```
      EXTSPEC spbuf imbuf ORD=nord
```

## A.12 Stellar Photometry Routines

### A.12.1 Locate Stars on an Image

Locate Stars Interactively:

```
      MARKSTAR [NEW] [RADIUS=r] [NOBOX] [STAR=s1,s2,...] [AUTO]
               [DR=dr] [DC=dc] [RSHIFT=rs] [CSHIFT=cs]
```

Locate Stars Automatically:

```
      AUTOMARK imbuf [RADIUS=rad] [RANGE=low,high] [REJECT=rej]
                     [BOX=b] [NEW]
```

Compute RA and DEC for Stars:

```
      COORDS [output redirection]
```

### A.12.2 Measure Stellar Brightnesses

Measure Brightnesses by Aperture Photometry:

```
APERSTAR imbuf STAR=rs SKY=r1,r2 [SKY=NONE]
                [GAIN=g] [RONOISE=r]
```

Generate a Point Spread Function Image:

```
PSF psfbuf [SIZE=n] [SKY=n] [AVERAGE] [RADIUS=r] [BI]
```

Measure Brightnesses using PSF Fitting:

Interactive:

```
FITSTAR psfbuf [FILE] [FLAT] [PLANE] [INTER] [LOCAL] [NOSUB]
                [RADIUS=r] [SEGMENT]
```

Batch:

```
BFITSTAR imbuf psfbuf [FILE] [FLAT] [PLANE] [INTER] [LOCAL]
                      [NOSUB] [RADIUS=r] [SEGMENT]
```

### A.12.3 Photometry Files

Print Current Photometry File:

```
PRINT PHOT [BRIEF] [output redirection]
```

Modify the Entries in the Current Photometry File:

```
MODPHOT
```

Store the Current Photometry File on Disk:

```
SAVE PHOT=filename
```

Retrieve an Old Photometry File from Disk:

```
GET PHOT=filename
```

## A.13 Extended Object and Surface Photometry Routines

Compute Centroid of an Extended Object

```
AXES imbuf [BOX=b] [SKY=s] [W=w1,w2] [output redirection]
```

### A.13.1 Aperture Photometry

Aperture Photometry of an Extended Object:

```
APER imbuf [RAD=r1,r2,...,r10] [MAG=M1,M2,...MN] [C=r,c]
           [STEP=size,n] [SCALE=s] [OLD] [INT] [REF]
           [output redirection]
```

Print Current Aperture Photometry File:

```
PRINT APER [output redirection]
```

Store the Current Aperture Photometry File on Disk:

```
SAVE APER=filename
```

Retrieve an Old Aperture Photometry File from Disk:

```
GET APER=filename
```

### A.13.2 Surface Photometry by Elliptical Contour Fitting

Surface Brightness Profile by Contour Fitting:

```
PROFILE spbuf imbuf [N=n] [ITER=n1,n2] [SCALE=s] [CENTER]
                    [PA=f] [INT] [FOUR]
```

Print Current Surface Photometry File:

```
PRINT PROFILE [output redirection]
```

Store the Current Surface Photometry File on Disk:

```
SAVE PROFILE=filename
```

Retrieve an Old Surface Photometry File from Disk:

```
GET PROFILE=filename
```

### A.13.3 Radial Brightness Profile Routines

Compute Radial Profile Along a Single Position Angle:

```
PRAD spbuf imbuf [PA=n] [CEN=n1,n2] [BOTH]
```

Compute Radial Profile by Azimuthal Averaging:

```
ANNULUS spbuf imbuf N=n [STEP=dr] [PA=pa] [INC=i] [CEN=r0,c0]
                        [SCALE=s] [FAST] [RAD=r] [PROF]
```

### A.13.4 Model 2-D Brightness Profile Generation

Reconstruct a Model Profile from a SURFACE Fit:

```
RECON imbuf [CR=r0] [CC=c0]
```

Model Image from User's Radial Brightness Profile:

```
TEMPLATE  imbuf spbuf [PA=n] [PAM=n] [E=n] [FIT=n1,n2] [SUB]
                      [GAUSS] [EXP] [HUB] [DEV]
```

## A.14 VISTA Command Procedures

Note that ANY valid VISTA command (or its unambiguous abbreviation) may be used in a procedure.

### A.14.1 Defining Procedures

Edit the Procedure Buffer:

```
PEDIT
```

Define a Procedure:

```
DEF [line_number]
```

Stop a Procedure Cold:

```
STOP ['A message']
```

End a Procedure:

```
END
```

Insert Lines into the Procedure Buffer:

```
IDEF [line_number]
```

End an IDEF and Keep Trailing Lines:

```
SAME
```

Remove Lines from the Procedure Buffer:

```
RDEF [line_number] [LINES=l1,l2]
```

Print Contents of the Procedure Buffer:

```
SHOW [output redirection]
```

Write the Procedure Buffer to Disk:

```
WP filename
```

Read an Old Procedure from Disk:

```
RP filename
```

### A.14.2 Executing VISTA Procedures

Start Procedure Execution:

```
GO [parameter1] [parameter2] ...
```

Call In and Start a Procedure as a Subroutine:

```
CALL procedure_filename [parameter1] [parameter2] ...
```

Return from a CALLed Procedure:

```
RETURN
```

Evaluate Parameters Passed to a Procedure:

```
PARAMETER [var1] [var2] [STRING=string1] ...
```

Trace Execution of a Procedure:

```
VERIFY Y   -or-   VERIFY N
```

Comment Lines in a Procedure:

```
! followed by any text, nothing following a ! is executed
```

### A.14.3 Variable Input/Output in Procedures

Formatted Output

```
PRINTF 'Format string' [expressions] [output redirection]
```

Prompted Input:

```
ASK  ['An optional prompt in quotes']  var_name
```

### A.14.4 Simple Flow Control in Procedures

Pause a Procedure:

```
PAUSE 'pause message'
```

Resume a PAUSE'd Procedure:

```
CONTINUE   -or-   C
```

Jump to a Labelled Line in a Procedure:

```
GOTO label_name
```

Label a Line as a GOTO Jumping Point:

```
label_name:
```

DO Loops:

```
DO var=from,to,[step]
      <execute these procedure lines>
END_DO
```

NOTE: No spaces are allowed in the argument list of the DO loop.

Execute on Error:

```
ERROR  VISTA_command
```

Execute on End-of-File:

```
EOF  VISTA_command
```

### A.14.5 Logical 'IF' Control Statements

VISTA supports a set of logical operations among VISTA variables to allow logical comparison (TRUE/FALSE) between two arithmetic expressions. The following table lists the supported logical operations:

| Symbol | Function | Examples | |
|---|---|---|---|
| > | greater than | A>B | A>2.5 |
| < | less than | A<B | A<100 |
| == | equal to | A==B | A==10 |
| ~= | not equal to | A~=B | A~=10 |
| <= | less than or = | A<=B | A<=5 |
| >= | greater than or = | A>=B | A>=3 |

NOTE: as with arithmetic expressions, no spaces must appear anywhere in a logical expression.

A set of simple IF block structures are illustrated below. In what follows, the syntax < > is used to denote complicated expressions which have no general form. For example, "<condition>" denotes a logical expression to be tested. For example, <condition> might be replaced by expressions like:

```
A==B
X~=0
A*(12.0)>=(B-EXPOS)/4.0
```

and so forth. The angle braces (< >) are being used to isolate the conditional statement for easy reading. They are NEVER typed.

Simple IF Block:

```
IF <condition>
      Execute these lines if <condition> is true.
END_IF
```

Simple Two-Level IF Block:

```
IF <condition>
      Execute these lines if <condition> is true.
ELSE
      Execute these lines if <condition> is false.
END_IF
```

Multi-Level IF Block:

```
IF <condition 1>
      lines to be executed if <condition 1> is true.
ELSE_IF <condition 2>
      lines to be executed when <condition 1> is
      false and <condition 2> is true.
   .
   .
ELSE_IF <condition N>
      lines to be executed when all conditions
      are false except <condition N>.
ELSE
      lines to be executed if and only if
      all other conditions are false.
END_IF
```

## A.15 External ASCII Data Files

### A.15.1 Opening and Closing Text Files

Up to 5 ASCII text file may be opened at one time. VISTA uses user-supplied "logical names" to distinguish between them.

Opening an ASCII Text File:

```
OPEN logical_name file_name
```

Closing an OPENed ASCII Text File:

```
CLOSE logical_name
```

### A.15.2 External File Operations

Read the next line of an ASCII Text File:

```
READ logical_name
```

Data Syntax for Lines READ from a File:

Each column of an ASCII text file is read into a VISTA variable named:

```
@LOGNAME.n
```

Where: LOGNAME is the user-assigned logical name (see OPEN) and n is the column number.

Skip Selected Lines in an ASCII Text File:

```
SKIP logical_name line [line1,line2]
```

Reset File to READ from First Line:

```
REWIND logical_name
```

Get File "Statistics" :

```
   STAT variable=MAX[arithmetic expression]
or
   STAT variable=MIN[arithmetic expression]
or
   STAT variable=FIRST[arithmetic expression]
or
   STAT variable=LAST[arithmetic expression]
or
   STAT variable=COUNT[logical_name]
```

where "arithmetic expression" is any valid VISTA arithmetic expression which must include one (or more) of the "@LOGNAME.n" variables.

## A.16 Miscellaneous Commands

Clear the Terminal Screen:

```
CLEAR [TEXT]
```

Read the VISTA Welcome Message and Old News:

```
NEWS
```

Clock the Execution Time of a Command:

```
TIME VISTA_command
```

Turn Prompt Bell On/Off or Ring the Bell:

```
BELL Y   -or-   BELL N   -or-   BELL R
```

Execute an Operating System Command from VISTA:

```
$ Any Valid Operating System Command
```

Temporarily Return to Operating System Level:

```
$   with no arguments
```

To return to VISTA type LOGOUT.

# Appendix B

# Sample VISTA Command Procedures

This chapter is a short collection of simple VISTA command procedures, some of which pertain to image processing tasks mentioned in the text. The point is not to give a detailed account of how to make and use VISTA procedures, but to archive some of our favorites. The best way to learn how to write VISTA procedures is by imitation. These examples should suffice quite nicely. They range from rather simple (almost trivial) to a little involved. It would help if you were familiar with the VISTA Help Manual, and had it handy to explain what is going on. Consider these as exercises for the reader to figure out. There is no particular order. Enjoy!

## B.1 A Sample STARTUP Procedure

This is the startup file of one of the authors. It defines a number of convenient command aliases.

```
SETDIR IM DIR=SCRATCH:
SETDIR SP DIR=SCRATCH:
ALIAS TV    'TV NOLABEL CLIP'
ALIAS COP   'COPY'
ALIAS CL    'CLEAR'
ALIAS RT    'RT NOMEAN'
ALIAS TBOX  'TVBOX'
ALIAS ZAPIT 'TVZAP SIG=2'
ALIAS RAIN  'COLOR CF=RAIN'
ALIAS WRMB  'COLOR CF=WRMB'
```

```
ALIAS BW    'COLOR CF=BW'
ALIAS IBW   'COLOR CF=IBW'
ALIAS WTF   'WT BITPIX=32'
ALIAS WT    'WT BITPIX=16'
ALIAS WD    'WD FULL'
ALIAS RS    'RD SPEC'
ALIAS WS    'WD SPEC FULL'
END
```

## B.2 Interpolate across known bad columns

This is a short procedure to interpolate across known blocked columns on the TI 500×500 CCD in use on the 1-meter Nickel Telescope at Mt. Hamilton. It is also an example of the use of the PARAMETER command which allows procedures to be "called" like subroutines from within VISTA without reading them in. For example, to interpolate out the bad columns on an image in Buffer 5, you would issue the VISTA command:

```
GO: CALL BADCOL 5
```

and away go the columns.

```
!
!  BADCOL.PRO   Remove major blocked columns on TI 500X500
!
PARAMETER IMNO
BOX 9  SR=280 SC=405 NR=220 NC=5
BOX 10 SR=412 SC=445 NR=88  NC=5
INTERP $IMNO BOX=9,10 COL
BOX 9  SR=0   SC=29  NR=500 NC=2
BOX 10 SR=410 SC=47  NR=90  NC=1
INTERP $IMNO BOX=9,10 COL
BOX 9  SR=334 SC=190 NR=166 NC=1
BOX 10 SR=117 SC=360 NR=383 NC=2
INTERP $IMNO BOX=9,10 COL
BOX 9  SR=80  SC=227 NR=420 NC=1
BOX 10 SR=1   SC=411 NR=499 NC=1
INTERP $IMNO BOX=9,10 COL
END
```

## B.3 Prepare Summed Flats

This procedure will read in a set of flat-field images from tape and add them together to make a single, summed flat. The user is prompted for the tape unit number, the file number on tape of the first flat and the file number of the last flat. All flat fields between these will be summed, and the result put in Buffer 1. It is a good example of simple use of the ASK command to prompt for input, the PRINTF to print responses at the user, and of the use of a DO/END_DO loop.

```
!
! FLAT.PRO:  Read Flat Field Frames from tape and add together into
!            a single flat field.  Assumes that flats come from the
!            tape in adjacent files.  The summed flat is made in
!            BUFFER 1 and BUFFER 2 is used as working space.
!            Baseline correction is applied to all flats.
!
ASK 'Tape Unit Number ? : ' TAPE
ASK 'Enter file number of first FLAT ? : ' FLAT1
ASK 'Enter file number of last FLAT  ? : ' FLAT2
RT 1 $FLAT1 UNIT=TAPE NOMEAN
BL 1
DO TN=FLAT1+1,FLAT2
      RT 2 $TN UNIT=TAPE NOMEAN
      BL 2
      ADD 1 2
END_DO
DISPOSE 2
MN 1
PRINTF 'Summed Flat is in BUFFER 1'
END
```

## B.4 Draw a Scale Bar on an Image

If you want to draw a scale bar on an image to indicate angular scale, this procedure could come in handy. It is the procedure used in the chapter on "dirty tricks".

```
!
```

```
!   BAR.PRO:  Draws a scale bar of length NCOLS on an image
!
ASK 'IMAGE NUMBER ? : ' IM
ASK 'PUT BAR IN WHAT IMAGE ROW ? : ' ROW
ASK 'STARTING COLUMN ? : ' ZSTART
ASK 'LENGTH ? : ' NCOLS
ASK 'INTENSITY VALUE ? : ' VALUE
SET ZEND=ZSTART+NCOLS
DO Z=ZSTART,ZEND
      VAL=SETVAL[IM,ROW,Z,VALUE]
END_DO
VAL=SETVAL[IM,ROW-1,ZSTART,VALUE]
VAL=SETVAL[IM,ROW+1,ZSTART,VALUE]
VAL=SETVAL[IM,ROW-1,ZEND,VALUE]
VAL=SETVAL[IM,ROW+1,ZEND,VALUE]
END
```

## B.5 Cross Correlate 2 Images

This is a procedure to cross correlate two images and find the shift between them. It is a procedure version of the cross correlation technique for image registration discussed in Chapter 4. Among other things, it demonstrates the use of formatting statements in the PRINTF command to make spiffy looking output.

As an exercise, re-write the procedure to replace all of the ASK commands with a single PARAMETER command so that you could invoke CROSS with the command:

```
GO: CALL CROSS 1 2 3 2 5
```

which will cross correlate the image in Buffer 1 against the template image in Buffer 2, use Buffer 3 as a temporary work space, use a cross-correlation radius of 2 pixels, and limit the computation to the contents of BOX 5.

```
!
!   CROSS:   Cross Correlate two images and find relative shift
!
ASK ' BUFFER WITH THE IMAGE TO SHIFT ? : ' IMAGE
ASK ' BUFFER WITH THE TEMPLATE IMAGE ? : ' TEMP
```

```
ASK ' BUFFER TO USE AS WORKING SPACE ? : ' WORK
ASK ' RADIUS FOR CROSS CORRELATION (TYP=2) ? : ' WIDTH
ASK ' BOX TO USE ? : ' DABOX
CROSS $WORK $TEMP $IMAGE RAD=WIDTH BOX=DABOX
SURFACE $WORK LOAD
SET DENOM=4*COEFFC2*COEFFR2-COEFFRC*COEFFRC
SET DELR=MIDR+(COEFFC*COEFFRC-2*COEFFR*COEFFC2)/DENOM
SET DELC=MIDC+(COEFFR*COEFFRC-2*COEFFC*COEFFR2)/DENOM
PRINTF 'Relative Shift:'
PRINTF '   DELTA ROWS : %F12.6' DELR
PRINTF '   DELTA COLS : %F12.6' DELC
END
```

## B.6 Another Interpolation Procedure

Another bad region fixer-upper, and a good example of a simple use of IF/END_IF statements for flow control in a procedure. It also shows that you don't have to use upper case all of the time to write your procedures. In this case, the IF/END_IF statement is being used as an error trap.

```
printf 'Interpolation routine \N'
ask 'Enter ROW number    => ' rown
ask 'Enter lower COLUMN  => ' lcoln
ask 'Enter upper COLUMN  => ' ucoln
if ucoln>lcoln
  numcol=ucoln-lcoln+1
  box 10 nr=1 sr=rown sc=lcoln nc=numcol
  interp 4 row box=10
  printf 'Fixed row %i3 - column %i3 to %i3' rown lcoln ucoln
end_if
end
```

## B.7 Automatic Tape Archiving

This little procedure is a good example of how to get down and dirty with two very powerful VISTA utilities: String Substitution and reading data from external

ASCII files to provide command input. What this procedure does is read a pre-prepared listing of image files stored on disk, reads them into VISTA in order, and writes them onto tape. A way of making an "archive" of files stored temporarily on disk. This was originally written to make a tape archive of all of the data from a reduction session covering an entire night of observing. It is very handy for making archives of a hundred spectra cluttering up the scratch disk.

```
!
!   ARCHIVE.PRO:  Tape Archive Procedure.  This procedure archives
!                 images onto magnetic tape in FITS format.
!
PRINTF '       *** CCD Disk-to-TAPE File Archiving ***'
PRINTF '\N'
PRINTF 'Before continuing, be doubly sure that the files to be '
PRINTF 'written are correctly named and in the proper order in'
PRINTF 'the LISTing file.'
PRINTF 'This file can be created by typing:'
PRINTF '      $DIRECTORY/NOHEAD/NOTRAIL/OUT=fname.typ *.CCD;*'
PRINTF 'Finally, make sure the tape is loaded and mounted.'
STRING DIRECT  '?Enter the directory name containing the images ? '
STRING NAME    '?Enter the listing filename.typ ? '
STRING FILE    '%A30%A30' {DIRECT} {NAME}
ASK             'Enter the tape drive unit number ? ' TAPE
!
SETDIR IMAGE DIR={DIRECT}
SETDIR SPECTRA DIR={DIRECT}
OPEN LIST {FILE}                    ! Open LISTing file.
STAT NSPEC=COUNT[LIST]              ! Count the # of images in file.
DO I=1,NSPEC
      STRING DISKIM {LIST}          ! Get disk image name from file.
      RD 1 {DISKIM}                 ! Read image from disk.
      WT 1 BITPIX=32 UNIT=TAPE      ! Write to TAPE w/full precision.
      PRINTF '\N'
      PRINTF 'Wrote file %A30 successfully' {DISKIM}
END_DO
DISPOSE 1
END
```

## B.8 Automated Wavelength Calibration Procedure

This is a very complicated (admittedly overkill) procedure that is used quite often at Lick for automatic Wavelength calibration of spectra with lots of interaction with the user. One of the many things it does is interrupt procedure flow to allow the user to issue VISTA commands free of the procedure (for example, to load a flat-field frame if forgotten). It also makes heavy use of string substitution, and makes use of the output re-direction facility to write a log of the calibration session into an external text file. Dick Shaw and Rick Pogge are responsible for this little nasty.

To explain a peculiarity: in general, spectral images taken at Lick on the 1-meter and 3-meter Cassegrain spectrographs come out with the dispersion (wavelength) running along columns, while VISTA (and most humans) like to see dispersion running from left to right, or along rows as viewed on the TV screen. Thus, this procedure will first ROTATE the raw spectral images of the comparison lines before flat-field correction.

```
!
!  COMPAR.PRO:  Wavelength calibration routine.
!
!       Determines the polynomial wavelength scale from the
!       comparison lamps & linearizes the lamp spectrum.
!
PRINTF ' \N'
PRINTF 'Welcome to the WAVELENGTH CALIBRATION routine.'
PRINTF ' \N'
PRINTF 'Do you have a FLAT-FIELD (and MEAN) loaded into buffer 1?'
PRINTF 'Type "C" if so; otherwise set things up, then type "C".'
PAUSE
ASK 'Grating Number = > ' GRAT
ASK 'Tape Unit Number => ' TAPE
ASK 'How many lamp exposures will there be [<=8] => ' NLAMP
ASK 'Order of polynomial for fit:  => ' PORD
ASK 'Approximate Central Row of Spectrum => ' SPCEN
RS=SPCEN-10
RE=SPCEN+10
DO SNO=2,NLAMP+1
      SAVE=SNO+10
      ASK 'Image file number on TAPE ? => ' TN
```

```
      RT $SAVE $TN UNIT=TAPE NOMEAN
      BL $SAVE
      ROTATE $SAVE LEFT
      DIV $SAVE 1 FLAT
      MASH $SNO $SAVE SP=(RS,RE)
      DISPOSE $SAVE
END_DO
PLOT 2 MIN=0
PRINTF ' \N \N WAVELENGTH IDENTIFICATION from appropriate files... '
!
! Print list of available line ID lists.
!
$DIR [VISTA.BASE.LAMDIR]*.WAV
!
!  Call line ID routine with interactive line
!  selection & extra displayed info.
!
LINEID 2 INT TTY
PRINTF 'If you have blown the wavelength ID ... start again.'
PRINTF 'If things are O.K. type "C" to continue.'
PAUSE
IF NLAMP>1
DO SNO=3,NLAMP+1              ! ID lines from other lamps if needed.
      PLOT $SNO MIN=0
      $DIR [VISTA.BASE.LAMDIR]*.WAV
      LINEID $SNO INT TTY ADD
END_DO
END_IF
CLEAR
WSCALE 2 ORD=PORD TTY INT     ! derive polynomial scale.
IF NLAMP>1
      DO SNO=3,NLAMP+1        ! copy scale to other lamps.
            COPW $SNO 2
      END_DO
END_IF
PLOT 2 MIN=0
PRINTF 'Choose plot parameters from first calibration lamp...'
ASK 'Starting wavelength for plot (and polynomial) => ' SWV
ASK 'Ending   wavelength for plot                  => ' EWV
```

```
ASK 'Final Linear Dispersion for Spectra          => ' FD
CLEAR
!
! Write Reduction log info.
!
STRING OUT1 'COMPAR_G%I1' GRAT
STRING OUT2 '.LIS'
STRING OUTFILE '%A9%A4' {OUT1} {OUT2}
PRINTF '     ***   WAVELENGTH CALIBRATION   ***' >{OUTFILE}
PRINTF '_____________________________________________' >>{OUTFILE}
PRINTF 'Using VISTA Version 3.0' >>{OUTFILE}
PRINTF 'Grating %I2 -- Dispersion %F5.2 A/px' GRAT FD >>{OUTFILE}
PRINTF 'Wavelength scale calibrated from:' >>{OUTFILE}
PRINTF '       %F6.1 to %F6.1 \N' SWV EWV >>{OUTFILE}
PRINTF 'Lamp spectra mash'd over rows %I3 to %I3 ' RS RE >>{OUTFILE}
PRINTF '*********************************************' >>{OUTFILE}
PRINTF ' \N' >>{OUTFILE}
PRINT LINEID >>{OUTFILE}
PRINTF '*********************************************' >>{OUTFILE}
PRINTF 'Fitting summary:' >>{OUTFILE}
PRINTF ' \N' >>{OUTFILE}
WSCALE 2 ORD=PORD >>{OUTFILE}
PRINTF '*********************************************' >>{OUTFILE}
PRINTF ' \N' >>{OUTFILE}
!
PRINTF 'Lamp(s) used for Calibration :' >>{OUTFILE}
PRINTF ' \N'
PRINTF 'Write each lamp twice: polynomial and aligned'
COPY 10 2
DO SNO=2,NLAMP+1
      PRINTF 'Rename Spectrum if desired'
      CH $SNO                              ! Rename image.
      BUF $SNO FITS=OBJECT
      PRINTF 'Polynomial first: '
      WD $SNO FULL                         ! Polynomial
      BUF $SNO FITS=OBJECT >>{OUTFILE}
      PRINTF '\N Linear next:'
      ALIGN $SNO DSP=FD W=(SWV,1) LGI
      PLOT $SNO XS=SWV XE=EWV MIN=0 HARD INFO GRID
```

```
      WD $SNO FULL                          ! Aligned
      DISPOSE $SNO
END_DO
PRINTF 'Output is in file:  %A13' {OUTFILE}
COPY 2 10
PRINTF 'Valid calibrated lamp is loaded in buffer 2'
END
```

# Appendix C

# VISTA and the FITS Standard

This appendix is meant to discuss, briefly, the basic principles of the FITS standard, and how VISTA interacts with it. The main reason for this discussion is to clarify areas where VISTA has adopted some of the optional FITS header cards to somewhat non-standard applications, and troubles that can arise reading data from other sites. This primarily for experts, but since VISTA has a `FITS` command that allows the user to change FITS cards, it is possible (and demonstrated) that the user can crash VISTA right to the floor by tweaking a wrong card ("not *that* button!").

To get straight to one important point. In the current release of VISTA in use since 1986, images written to magnetic tape by VISTA are FITS standard, and have been read successfully by most other image processing programs in use among the astronomical community (IRAF, MIDAS, and AIPS). In addition, VISTA has been able to read FITS tapes brought from other institutions (*e.g.,* ESO), but it cannot at present read so-called "blocked" tapes written at 6250bpi density.

As will be discussed below, VISTA has adopted a few of the standard FITS header cards for use *internally* in a manner inconsistent with the accepted standard. This is perfectly harmless for most applications. Where VISTA collides with the standard is in operation which attempt to work in "physical" (or "world") coordinates — such as arcseconds, wavelength, etc. — explicitly. For most applications, like 1-D spectra, the difference vanishes. For more sophisticated applications, like calibrated 2-D spectra (this does not include echelle spectra, but rather long-slit spectra), there can be points of confusion. It is hoped that this discussion will serve to clarify the issues.

## C.1 The FITS Standard

FITS is an acronym for *Flexible Image Transport System.* It is an internationally agreed upon structuring standard governing the format used to write astronomical data to magnetic tape. The basic intent is to make any data written anywhere in the world completely portable to anywhere else, without having to have a separate program to read the data. One standard format, one standard set of conventions, one program (in principle) capable of reading everything.

For the most part, this has been accomplished. Some uses of the FITS format for storing multi-dimensional radio data (or Fabry-Perot Interferometer images) are more than some sites can handle, but when it comes to 2-D images, it works quite well. I won't go into great detail in this appendix, but rather refer you to the principal reference: *FITS: A Flexible Image Transport System*, Wells, Greisen, and Harten, 1981, *Astronomy and Astrophysics Supplement Series*, **44**, 363.

The basic structure of a FITS file, as it pertains to 2-D images, is as follows: Each file has a *Header* containing all of the information on the structure of the image data on the tape, and any auxilliary information describing the contents to the user (*e.g.,* object name, exposure time, date of observation, etc.). Following the Header are the data records, which contain the image data itself.

The way most programs, including VISTA, work is to first read the header. This will tell the program how to sort out the data records and essentially reconstruct the image from an (effectively) unstructured stream of numbers. The means that data storage is (in principle) as compact as possible, and intelligible to any program that can interpret FITS headers. This is the essense of "portability."

The organization of the header is relevant to the following discussion. The header is divided into "cards" each of which contains some piece of information. The first 8 characters of a card contain the name of the card. The header cards most essential for image untangling have been standardized. In brief, 6 cards are absolutely required for a 2-D image. These are, IN ORDER:

**SIMPLE** A logical variable, always "T" (True) if the image is a "simple" FITS image. All VISTA images are "simple" in this regard.

**BITPIX** An integer, it gives the number of bits per pixel of the image data. VISTA, and most other programs, support only BITPIX=16 and BITPIX=32.

**NAXIS** An integer, equal to the number of image axes. A 2-D image has "NAXIS=2". A 1-D image ("spectrum") has "NAXIS=1".

**NAXIS1** The number of pixels along the 'first' image axis. For 2-D images, this is the number of columns, or for spectra, the number of pixels (sometimes

called "channels").

**NAXIS2** The number of pixels along the 'second' image axis. For 2-D images, this is the number of rows, and for spectra, NAXIS2 is 1.

**END** In between the last *NAXISn* card and the end of the FITS header, you can have as many cards with as much info as you like. However, the last card must be "END".

Beyond this basic set of FITS cards, all of the other cards are considered optional. FITS cards can take real, integer, or character variables, single or double precision. The FITS standard also defines some basic 'optional' cards that most people use. Thus, while intended as 'optional', they have by common use become an integral part of the standard. Again, interested readers are referred to the aforementioned paper for details. VISTA has adopted some of these for its own purposes, and these are discussed in the next section.

## C.2 FITS Cards Important to VISTA

VISTA uses some of the FITS header cards info in non-standard ways that can potentially cause trouble if care is not taken. These are listed in the table below, with the VISTA usage and standard usage given.

| Special FITS Cards | | |
|---|---|---|
| FITS Card | VISTA Usage | Standard Usage |
| CRVAL1 | Starting Column of an Image | Value in physical coordinates along the columns axis of the reference pixel in columns |
| CRVAL2 | Starting Row of an Image | Value in physical coordinates along the rows axis of the reference pixel in rows |
| CDELT1 | On-Chip Binning Factor in Columns | pixel scale in physical coordinates along columns axis |
| CDELT2 | On-Chip Binning Factor in Rows | pixel scale in physical coordinates along rows axis |
| CRPIX1 | Array index of starting column | Array index of the refernce pixel in columns |
| CRPIX2 | Array index of starting row | Array index of the refernce pixel in rows |
| CTYPE1 | If a spectrum, indicates wavelength scale type | Units of physical coordinates along the columns axis (*e.g.*, arcsec) |
| CTYPE2 | Unused | Units of physical coordinates along the rows axis (*e.g.*, arcsec) |

Of particular importance to VISTA are the `CRVAL1` and `CRVAL2` cards, which are used to to determine the array indices for the image. If these are changed without care, it could cause VISTA to crash. If these cards are changed for any reason, you need to copy the image buffer into itself (*e.g.*, if you change them in the image header of Buffer 1, you need to issue the command: `COPY 1 1`). This operation will reset the internal array index pointers and keep VISTA happy. VISTA is internally consistent in its use of these cards, but there is demonstrated collision with some data from other institutions. For example, 2-D images which are wavelength calibrated along one axis brought to Lick from ESO caused a fair amount of havoc. An exception is within the plotting commands (`PLOT` and `CONTOUR`) which make use of the FITS coordinate cards as defined by the standard. Examples of this is

given in the chapter on "dirty tricks".

The principal point here is to keep in mind that VISTA works internally with array indices (*i.e.,* pixel number), not physical coordinates. Thus, if you add together two spectra that are wavelength calibrated together, they will be added *pixel-by-pixel*, not by wavelength. Unless the spectra overlap in pixels and wavelength simultaneously, the addition of the two will create garbage. As can be seen from the above table, VISTA does not use physical coordinate FITS cards in a manner consistent with the definitions outlined in the 1981 standard paper. This constitutes VISTA's primary weakness.