A GUIDE TO GALAXY PHOTOMETRY WITH PPP

H. K. C. Yee

Table of Contents

ppp *Version: 6.0, October 2007, (under construction)*

Preamble: Since the last version of the PPP user's guide (2004), very significant changes have been made to the program. While the algorithms are basically the same, the program has become more automatic to use, including a one command pipeline for reducing multiple chip data (geared to CFHT MegaCam data). This is a simplified version of the user's guide, focusing on using **ppp** for galaxy photometry.

I. INTRODUCTION TO PPP

PPP (A *Picture Processing Program*) is an interactive image processing package that uses a simple command language. It is a package chiefly designed to analyse direct images with emphasis on galaxies. For a description of the algorithm, the rationale behind the method (and madness), and tests of simulated images, see Yee (1991, *PASP*, April, **103**, 396). Some improvements to the algorithm, including how to deal with variable PSF, are described in Yee et al. (1996, ApJS, 102, 269). Additional modifications and improvements have been implemented since. It must be emphasized that **ppp** is an experimental program and it will continue to evolve and change over time. It is thus not meant to be a "public program", and potential users are requested to obtain permission from me before using it.

In this document, commands are written in **bold** face and optional inputs in *italics* to facilitate reading. Also, an exclamation mark after a command denotes explanatory comments.

1.0 **Set-Up (Unix/Linux)**

If you want to start with the executable (assuming it works), you will also need will need to obtain the following files: ppp (the executable), and grfont.dat (pgplot fonts). The pgplot fonts file is required if you are planning to make plots (see Section 1.6). Starting **ppp** is simple with the executable file. First make sure that the file is executable (i.e., use the unix command *chmod 775 ppp.*) In your .login (or equivalent) file set the following *setenv* statements:

setenv PGPLOT_FONT "*full pathname*/grfont.dat"

You may need to recompile the program. In this case, you will need to obtain the *.f, *.inc, and the makefile (makeppp8kx). The makefile available is what works on my computer, and you need to modify it to suit your system. You will also need the pgplot library and the cfitsio library. The source codes have been written so that it will compile properly for both PC linux and MAC linux. Note that the two executable are not transferable.

1.1 **Data Structure**

ppp uses a large array INTEGER*4, IXB, to hold images in the memory for operation. This array is referred to as the *ppp core* or *buffer*. This one-dimensional array is partitioned into 2-dimensional arrays that are called *frames*. The last frame is a "scratch" array, and is not to be used unless you know what you are doing. The total size of the buffer can be changed only in the source code. The standard version has an array size (just) sufficiently large to allow operation on 8k images; you can change to default size of the image to allow more frames in the buffer. (Linux Fortran, or at least the verson I use, do not allow a total size of larger than two frames of 13k$\times$13k $\times$).

The data are normally stored in the disk as FITS files (**name.fits**). A directory (i.e., index) of the file names (in a specific unix directory) is kept in a file called **pppdir.dat**. **PPP** reads and updates the directory file each time it is run. The command **dir** will display the current disk picture directory, and also the core/buffer directory. To operate on any image, it must first be read into one of these buffer frames via the appropriate reading routine. The reading of the disk image is via subroutines in cfitsio, and **ppp** can read images of arbitrary type (including compressed and rice compressed), **but** the image is ALWAYS read in as integer*4. Historically, the image array has been kept as integer*2 to save memory space; and as computer technology advanced, it has been expanded to integer*4 to allow considerably more flexibility. However, integer arrays are continued to be used, rather than floating point, because of the huge amount of modifications needed in order to change from integer to real, including some algorithm which depends on the use of integers for the pixel value. **IMPORTANT**, thus, images MUST have enough dynamic range in integer values before it should be used by **ppp**. E.g., if an image contains values between 0.000001 and 0.001 (as, e.g., flux calibrated images from Spitzer), all the pixel values will be read in as identically 0. The recommendation is that the image should be scaled so that the rms in the sky is at least 3 or 4 ADU, preferably around 10 ADU or more. This must be done outside of **ppp** (e.g., using IRAF).

ppp normally is run by calling it in the (unix/linux) directory where the images reside. However, for the purpose of easier book-keeping, a better way is to run it from a directory where you want to store the result. Images in another directory can be accessed by setting the parameter *imdir* to the full path name of the image directory. (See §1.5 for setting parameters.) In this case, all inputs and outputs pertaining to the images will be defaulted to *imdir*. However, all other inputs and outputs (such as position and photometry files, see Section 2 and 3) are defaulted to the directory from which you called **ppp**.

It is possible to set the image directory to another user's directory in which you have read privilege, but not write privilege, with some limitations (e.g., not able to uncompress an image).

Associated with the first *nposfil* (= number of position files) frames are position array buffers analogous to the image buffers. *nposfil* is a parameter hardwired in the program, much like the number of maximum frames. These are 1-dimensional arrays of X-Y co-ordinates (see §1.4) which store the integer pixel positions of objects that are used primarily in photometry commands. Each position array holds *nobmax* object positions. The parameters *nposfil* and nobmax are presently set to 2 and 70,000, and can only be changed by adjusting the include file *objcat.inc* and recompiling the program. Position files that are saved in the disk are named **name.pos**.

1.2 **Command Structure**

The grammatical structure of **ppp** is simple. Commands consist of a command word, one or more frame numbers, and optional input parameters. Only the first 4 letters of a command word are significant. If the frame number(s) or other input parameters are absent and required, **ppp** will request these automatically. Many commands request additional information after entrance into the command. For most commands, *default* entries (mostly the previously used values) to queries for inputs are available. (However, it is found that linux does not always preserve the previously used values; so, do not count on this without looking at the default value before accepting.) Note that the values from the previous session of **ppp** are not kept except for global parameters accessible by the command **set** [see 1.5].)

For commands available use **help** in **ppp**.

To escape back temporarily to unix/linux while in **ppp**, use **control z** to put **ppp** in the background. (Unix/linux's **fg** will bring it back). Or, alternatively, for escaping to a single Unix command, use *%unix command*. You may be able to interrupt the execution of a **ppp** command to get back to the **ppp** prompt by issuing a **control d** (i.e., end of data) ONLY while the **ppp** command is waiting for an input. (Note: in linux, the control-d works sometimes and not other, and when it does not, ppp is terminated.) There is no way to interrupt a command while it is being executed. **Control c** will end the **ppp** session completely! This is because almost all **ppp** commands are run as subroutines, and not separately spawned programs.

PPP can be run in batch mode by defining the input file and output file. In unix, this is equivalent to the command:

ppp < input file name > output file name &

Here the iput file is an ascii file with each line representing the (exact) input into **ppp** for running what you want, including the inputs requested by each **ppp** command.

A history file of ALL input entries can also be created by setting the *idhis* parameter to greater than 50. The history file is then automatically stored in **fort.n** where **n** is the *idhis* parameter.

Besides the commands in the first level of **ppp**, there are two sublevels of commands: **posf**, and **mfo**. The commands in the **posf** level operate on position file buffers and have the same grammar as those in **ppp**. The commands in the **mfo** (Multiple-File-Operator) menu allow the user to operate on images in the disk directly and in multiples. The grammar for **mfo** is similar to that of **ppp**, except that the "frame numbers" now refer to the file numbers in the **ppp** disk directory. (You can use **dir** to list the directory numbers.) The file numbers are specified as a string of numbers, with the feature that '–' between 2 numbers includes all numbers in between also. E.g.:

trim 2 5 7–10

will trim disk file number 2, 5, 7,8,9, and 10. Note that **mfo** commands alter the images on disk permanently, whereas **ppp** commands operate only on the image in the buffers and leave the disk image unchanged. (To save an image changed by **ppp** in the ppp core, use **wdis** to write it to disk. Note: this command may still have some problems.) Also note that **mfo** commands use frame 1 as the operand. Hence, once you use anything in **mfo**, frame 1 will be altered.

1.3 Data Input and Output

1.3.1 To and From Disk

ppp reads and writes images in FITS format (please see important note in 1.1 on data structure). There are 2 commands for reading files that are already in the **ppp** directory. **rfd** *f n* reads the n^{th} directory file into frame *f*, whereas **rdis** *f name* reads the same file by specifying the name (no qualifier necessary) of the file instead of the file number. If a disk file is not listed in the **ppp** directory, one can use **rdis** to read it, and a new entry will be made to the directory. Note that it is a better practice to use **rdis** for reading in a batch job, as that will make the batch independent of the state of the directory. Note that **ppp** automatically uncompresses a compressed file before reading if the image file is compressed. Exiting with **quco** intead of **quit** will invoke a background job to recompress all FITS image in the directory *imdir*.

ppp offers a choice of two compression algorithms, set by the parameter *ipppfile*. For *ipppfile*=0, the compression is the UNIX command **compress**; and for −1, **gzip**. **ppp** also reads rice compressed files (.fz), but does not have commands to write or compress images into .fz.

The command for writing to disk is **wdis** which writes a short integer FITS file with no scaling in the unix version. The **ppp** directory is updated whenever a image is written to disk.

1.4 Co-ordinate System

PPP refers to the first co-ordinate as I (or X), and the second co-ordinate as J (or Y). An image starts with pixel (1,1). Note that **ppp** defines the center of the pixel (1,1) as position (1.0,1.0) [NOT (0.5,0.5) or (0.0,0.0)]. Coordinates in a position file are defined in order of the J co-ordinates (i.e., from bottom to top in the default orientation).

1.5 Setting Global Parameters

ppp has a number of parameters that can be set permanently. These include parameters for photometry, graphics display, default picture size etc. The parameters are created (from the source code) in its program default values when **ppp** is run for the first time in a directory and stored in a file call **ppparm.dat**. You can get a listing of the settable parameters by **set** *list*, which lists all the parameters with a terse explanation for each.

The parameters can be changed via the command **set** with 2 inputs: **set** *parameter–name value*

There are 2 **set** options that allow one to tailor the default values. "**set** *save*" will save the current parameter values to disk as default values so that they are kept for the next session. "**set** *default*" resets the default values to those created by the **ppp** program. Note that **ppp** automatically updates the default values if there has been a change or additions to the parameters when a new version of **ppp** is run for the first time. However, beware that when this happens, all the parameters that one *save*d before will revert back to the program default values. The original parameter file is saved in *ppparm.dat.OLD*, allowing you to refer to the old saved values to reset the parameters to your specifications.

For setting the maximum frame size permanently, one needs to **set** *nu* and *nv* (not via **msiz**), save the default by **set** *save*, and exit **ppp** and re-enter. (Or do a **msiz** in the same session.)

1.6 Graphics

All graphics in **ppp** use Tim Pearson's plotting package **pgplot**. There are two important parameters that must be tailored to the computing system on hand: *splot* and *hplot*. These 2 parameters represent the default devices for plotting interactively (i.e., screen), and for the hard-copying device with which the created plotting file is to be printed later. The default screen plotting device is /xwin; although anything supported by pgplot (e.g., /sun for sunview windows; /tex for Tektronic like graphic terminals) can in theory be used. In order to use plotting routines, the font file for pgplot must be set using:

setenv PGPLOT_FONT "*pathname*/grfont.dat"

Most plotting routines use the same cursor commands (although not all commands have been implemented in all plotting routines). See the description of cursor commands for **sdis** in **help**. One plotting routine that requires a more detailed description is the contour plotting routine **cplt**. The inputs for **cplt** (e.g., sky level determination and isophot levels) are almost identical to those of isophotal surface photometry and will be described in Section 4.2. The best way to learn about the command is to run it a few times. For many uses, especially for plotting a single object, the most useful contour level option to use is the logarithmic one with the scaling by the peak value of the brightest object (option *s*). This method of plotting allows one to compare the shape to very high precision of one object to another (in separate plots). There are several useful **set** parameters for **cplt**. The parameter *cplevel* sets the ratio of one level to the next for the *s* option. For example, the default value of 0.5 plots contours that are a factor of 2 apart beginning with the brightest pixel. (I.e., each contour is 0.5 of the previous one.) One useful application of this is to determine an accurate FWHM by plotting an object after it has been sinc-shifted to have an integer centroid. The parameter *cplx* controls the lowest level of the *s* option as multiples of the sky sigma. A smaller *cplx* will allow one to plot deeper down into the noise. The parameters *itrans* and *ilabel* allow one to transform the co-ordinates of the axes (e.g., from pixels to arcseconds) and to plot labels as specified in an existing file of positions and ascii characters. This is useful for creating contour plots suitable for publication.

1.7 A Primer for SAOimage

The current version of **ppp** uses SAOimage or ds9 for image display. The display is set by **set** display, with either sao or ds9 as the input (sao is the default). The following works only with sao, though a .reg file can be created to be input into ds9.

Unfortunately SAOimage does not lend itself to be easily integrated into **ppp**, mainly due to its rather clumsy way of reading and outputting co-ordinate information (which is the primary method of communication between **ppp** and SAOimage). Hence, all interactions between the two programs require explicit actions from the user to allow the two to know about co-ordinates of marked points. What follows is a very brief summary for the use of SAOimage itself.

An image is displayed from **ppp** using the command **sao** *n*. **ppp** computes the optimal parameters for the SAOimage command which it spawns.

The main thing that needs to be familiarized with is the menu bars on the top of the display. The top bar allows one to choose the menu available on the lower bar. E.g., if you choose *color*, the lower bar gives a number of choices, (although dragging middle button of the mouse while in **color** will always change the contrast). The right-most button on the lower bar provides an additional set of menu for the same upper bar choice. E.g., under *cursor*, there are two sets of lower bar choices, toggled by the button *region*: one allows you to choose the shape of the cursor, the other allows you to display, read, and write, among other things, cursor value files. One should experiment with various buttons and the mouse to learn about the various functions.

SAOimage uses "reg" files (with names like: **saoimage.reg**) to store and transmit co-ordinate/cursor information. Various commands in **ppp** produce .reg files to pass onto SAOimage for display or edit. A .reg file is read in through the buttons *cursor*, *region*, *read*, and the name of the file. (Yes, very clunky.) And written out by the *write* button under the same menu. The .reg file is edited via first choosing an appropriate cursor shape (*cursor*, *region*, marker shape), which shows up as green. After which, one can then drag it to the desired co-ordinate (left mouse button), and type either **e** for a red marker, or **s** for a yellow marker. The exact usage will be covered under each **ppp** function. **ppp** uses the convention **saoimage*n*.reg** for the .reg file names, where *n* is the frame number with which the file is associated.

The only two other functions of note are: (1) the *pan* button, which allows you to zone and center an image; and (2) the *etc* button, under which are *quit* for leaving SAOimage, and *new* allowing you to restart SAOimage with a different look-up-table range.

It is hoped in the near future, the display routine will be updated to DS9.

1.8 Fixing Bad Pixels

ppp has many commands for fixing bad pixels, cosmetic flaws, bleeding columns, cosmic rays, and non-flatness in an image. Only one method is discussed here – the interpolation over bad pixels using a bad pixel map. The command for creating a bad pixel map is **crbp**. This basically creates an image in which one marks a set of specified (bad) pixels with the value 1 while leaving the rest as 0's. The command allows one to mark the bad pixel areas using several geometric patterns: e.g. columns or rows of specified lengths, boxes, circles, points, and meteor trails (i.e., sloped lines). The best way to define a bad pixel map for a particular CCD is to median filter several images and check by eye the bad pixels. The bad pixel map can then be made by creating a macro file that runs the command **crbp** with the correct entries. This could be done by either typing it out long hand (running the command **crbp** will show the format), or using an interactive method described below. In general it is much better to save the macro file (which is equivalent to a list of the bad pixels) than to save the bad pixel image itself (which takes up more disk space).

The easiest method for creating the bad pixel macro file is interactively using SAOimage. This method can be used to create any mask (whether to correct for bad pixels, or to mask out bleeding columns, or unwanted areas). However, this method is restricted to creating mask regions that are rectangular in shape only. To create a bad pixel area, first pick the rectangle cursor in SAOimage. Change the size of the box (by dragging the middle button of the mouse) so that it encloses the desired area. The box should be within

1/2 a pixel of the desired boundary. Save the box position by hitting either the **e** or **s** key (for red or yellow color boxes). Note that you are able to save *two* sets of bad pixel regions (i.e., the two colors). Hence, unless you want two different macros, do not mix the colors. Once it is all done, write out the saoimage region file (SAOimage button: cursor, region, write), calling the file saoimageN.reg, where N=frame number of the image. In **ppp** enter the command **saocrbp** *n*. The command will read the SAOimage reg file and create two macro files: one for the red boxes, and the other, the yellow boxes. These files can then be run to make the mask frame, and also renamed. It is recommended that one should save this macro file instead of the bad pixel image.

Once the map is made, the command **bpfix** can be applied to the image(s). This command performs a bi-linear interpolation over the bad pixels and then adds the appropriate noise to make the new pixels look "normal". Often, for reason of clarity (so one knows where the bad pixels are), one may not want to have the bad pixel fix with noise added. In that case, set the *inoise* parameter to 0. Two other related commands are **bpfx** and **bpfy** which do the interpolation across the I or J direction, respectively. **bpfix** can be used in the **mfo** mode to fix images in the disk permanently once one is satisfied that the map works properly.

Other useful commands for creating bad pixel files are for cosmic ray detections and bleeding columns. The useful ones may include **crm1** and **crm2** (cosmic ray map using one or two frames), and **bldm** (map of bleeding saturated columns). These are described in later sections.

For many tasks, sometime it is desirable to create maps for individual fields for the purpose of masking out bleeding columns, cosmic rays etc. For example, in the object finding routine, **afim** (see section 2), masks can be made using the same method for bleeding columns or halos of bright stars for the purpose of cutting down the number of spuriously detected objects.

1.9 **Boxes**

You can limit the action of some commands (mostly arithmetic and statistics) to a specific area of the image by defining a box. A box is defined by either its largest and smallest (I,J) co-ordinates or by its center and extents in I and J. (See appendix for the appropriate commands.) You can define a total of 10 boxes. For commands that accept a box number as argument, the action of the command will be limited to that box limits. Note that not all commands accept a box. If you want to use a box, the command must be specified completely in one line (i.e., it will not request a box if you didn't specify one). The following is an example of using boxes:

boxe 3 ! define box #3 via its end co-ordinates
105 324 ! smallest I,J
154 373 ! largest I,J
#performed the specified picture arithmetics within the box
par p1=0.8*p1+0.2*p2 box3

II. OBJECT FINDING

The reader is urged to read thoroughly the paper (Yee, 1991), describing the (somewhat) crowded-field galaxy photometry algorithm, and the relevant sections in the CNOC1 technique paper (Yee et al. 1996, ApJS, 120, 269), detailing some improvements, before continuing on with this manual or using the photometry program.

Note that **ppp** uses the concept of creating one MASTER integer position file usable for all frames/colors of the field. The main advantage of this is that it simplifies color measurements, and it also allows derivation of upper limits for objects not detected in a particular filter. The disadvantage is that objects with color such that it is detected only in filter(s) other than the primary one used for object finding will be missed. There are procedures to deal with the above situation.

2.1 **Before Object Finding**

2.1.1 *Getting Some Basic Information About the Images*

In general, it is a good practice to obtain some minimal information about all frames of the field in question. This will also help in catching mistakes in the observing log, FITS header, or other problems with the images.

For images with multiple frames or colors you need to have reference objects in each picture so that you can determine the relative shifts of the pictures. (Note that even if the images were not intentionally shifted between exposures, there may still be a small shift.) The quickest way to do this is to display the image and locate one or two bright unsaturated stars. One can use the cursor to get an approximate central co-ordinate. Note that if you are in *cursor* mode in SAOimage, entering the cursor position with the *label* button on will mark the co-ordinate of the marked object. If you have an approximate position of the center, you can also use **tty** (which prints out on the **ppp** window the pixel values surrounding the point) to determine the exact peak point.

Although not necessary, if you want to determine the exact (partial pixel) centroid, you can use the command **cent**. If you want to do a quick check of the stability of the photometry from frame to frame, use **lpho** to get photometry on the same star(s) from different frames of the same filter. The background and rms/pixel values of a frame can be obtained using a variety of commands. The most useful ones are **rsmo/csmo**, **gsky**, and **sky**. The first two allow one to determine the statistics of a whole bunch of rows. (e.g., **rsmo** 1 200-210), whole the last one allows one to determine the sky in various geometric shapes (e.g. circle, ring...). Note that **rsmo/csmo** does not work with more than 20 rows/columns at a time. **gsky** determines the same for a series of rows (up to 40) spaced evenly across the picture. **rsmo** gives the statistics of a particular patch of the image (thus allowing one to avoid areas with very bright object(s)), and **gsky** is better for determining the average *global* values across the whole image.

Often one may also want to fix cosmetically the frames before object finding and photometry; e.g., replacing or masking a meteor trail, bleeding columns etc. In general, there are numerous routines (like cutting and pasting, scaling and replacing pixels etc. – advanced **ppp**!) that can be used for this purpose.

2.1.2 *Matching the Frames*

If you have several frames for the same filter, or several filters for an image, you may or may not want to match and trim the frames so that they cover an identical area on the sky. IRAF can stack shifted frames, but the edges will be less deep than the area covered by all frames. You may have to decide whether missing a little bit off the edge of one filter is a problem worth making the total field smaller. Note that if an object co-ordinate falls outside of a frame, **ppp** will return a magnitude of 99.9, and will not produce a problem.

The simplest and most consistent way to match multiple frames is to trim all the shifted images (from all filters) of the same field to a dimension such that they cover the same area. This can be done with **mpic** in **mfo**. The parameter *iversion* allows you to create a new version of the picture in the **ppp** directory. A default of *iversion*=0 would mean that it operates under normal **mfo** rules: i.e. the old original version of the **.fits** file is deleted automatically. (Note: Currently, Unix version has no option of purging – the picture is always purged after an **mfo** operation. The parameter *iversion* applies ONLY to **mpic**, but not other commands.) **mpic** requires as input the (integer) co-ordinates of a reference star from each frame (see 2.1.1). It will trim all the pictures so that that the reference star will have the same co-ordinate in all the frames.

This should be the final dimension of the pictures for analysis, since the area of the frame matters in doing galaxy counts.

e.g.: **mpic** 23 25-27 28 ! match PPP directory files 23,25,26,27,28
! MPIC then requests a reference co-ordinate
! for each frame.

2.1.3 *Masking Bleeding Columns and Diffraction Spikes*

The command **bldm** creates a mask which marks the bleeding columns and the diffraction pattern of bright stars. This is important for object finding, as these features will produce false detections that will contain enough counts to make them indistinguishable from real objects. The command requires a seed position file in which saturated stars are marked. The bleeding columns are produced by a connected-pixel algorithm with padding; while the diffraction spikes masking require empirical estimate of 4 parameters from the actual data. The diffraction spike pattern is defined by a length and the position angle of the spikes. Four spikes of equal length at 90^o to each other are assumed. The angle of the spikes are set by the parameter *difang*. The length of the spike is set by a quadratic with 3 constants set by the parameter array *dsc*. See the batch job example for details.

2.2 **Creating a Position File**

Object finding is more an art than a science; trial-and-error and analysing simulated data are some of the best ways to figure out what is the best procedure. Moreover, some degrees of manual intervention is usually necessary.

There are a half dozen commands for object finding in **ppp**, but by-and-large the work horse is **afim** or **afin**, and can (and should) be used with most images. **afim** does the object finding assuming there is a mask image to block off undesired area, whereas **afin** does not look for the mask image. Other algorithms are not well tested and may not even work, and hence, should not be used.

2.2.1 **Detecting Objects**

The basic object finding algorithm is described in Yee (1991). It computes the flux in a small aperture (3 pixels default) around any local maximum, using a small ring around the point as sky (size set by *iir* and *ior*. Any local maximum with flux above a threshold (best set by using the parameter *thresky*, expressed as units of local sky sigma above sky) will then be accepted as a candidate for further checking. The pre-1992 version checks for possible contaminant from cosmic ray detections using a simple sharpness test set by the parameter *iwd*. *iwd* is simply the ratio of the peak pixel to the next highest pixel.

A more sophisticated routine for determining whether a candidate should be accepted into the position file was implemented September 1992. This essentially eliminates most of the cosmic ray detections. Furthermore, it also limits the detection of false objects in the halos of bright galaxies and stars. These additional checks can be turned off by setting the parameter *iwd* to less than zero. In this case, the absolute value of *iwd* is used for checking for sharpness of objects.

The additional test first checks for connectivity of pixels around the candidate object. The user is queried for a diameter for the test of connectivity. The program examines the pixels within the diameter to determine whether they are above a certain threshold (which is set by the set parameter *conthres* as the fraction of sigma of sky). The candidate is accepted only if all the pixels within the aperture are above the threshold. This effectively eliminates most (sharp) cosmic rays and noise spikes. The diameter for connectivity check should be set at about 1 seeing disk, normally around 2 to 3 pixels. For faint detections, (with the threshold set by the parameter *smsig*, expressed as multiples of sigma of sky), a smoothing is done before the connectivity check.

The second check is for eliminating false object identifications in halos of bright stars and galaxies, and optical reflections in the image. The program examines a small area around the candidate object by first smoothing it. If the smoothing makes the local maximum disappear, then it tries to locate a new maximum immediately around the pixel. The search extends as far as *imerad*–1 pixels from the original detection, where *imerad* is a **set** parameter. Note that the *imerad* is also used as the radius at which all objects within *imerad* pixel from each another are merged as one. Hence, *imerad* should be set at about slightly less than one seeing disk FWHM. If there is still no such maximum, the candidate is eliminated. The command will query the box size for the smoothing; normal value is 3 pixels.

After all the pixels in an image have been checked, the algorithm does two more checks on the whole tentative position file. First, it tries to eliminate ghost images by checking preset position offsets from all saturated objects. Second, it tries to eliminate double finds (which occurs for faint oversampled objects) by merging any objects within *imerad* pixels from each other.

It is found that optimal procedure for detecting faint and low-surface galaxies is to perform object finding with a smoothed image. However, cosmic ray detections in a smoothed image often will deceive the cosmic rejection algorithm. Hence, the recommend procedure is to smooth the image (using **tsmo**, tapered box smooth), run object finding, and then recheck the resultant detection for cosmic-ray detection using the unsmoothed image. The latter is done using the command **rejc**.

2.2.2 *Running* **afim**

There are three basic steps: set up the parameters, create a mask frame, and run **afim**. (For the case with no mask, use **afin**; note that the inputs for the two are slightly different.) The basic inputs for **afim** are described in the batch job example.

2.2.2.1 *Important Parameters for Object Finding*

(1) *iwd* — *iwd* sets the criterion for rejecting an object during object finding routine. Candidate objects whose ratio of the peak pixel value to any of the surrounding pixel values is larger than *iwd* are deemed too sharp.

(2) *iir* and *ior* — This sets the inner and outer diameters of local sky determination for candidate objects during object finding. Normally *iir* should not be changed. By setting *ior* larger, one does better in finding faint objects next to bright ones. However, the cost is increased cpu time.

(3) *adusat* — the saturation level in ADU. This must be set for dealing with finding saturated stars (which do not have a nice single peak structure). See Section 2.3 for more details.

(4) *thresky* – Object finding threshold expressed in units of local sky rms noise. Typical values between 0.85 to 1.2 for an unsmoothed image and 2.5 to 3.3 for a smoothed image. (If set to zero, an object finding threshold is requested while running **afim/afin**.) This is the primary parameter you would play with in order to find the right object finding threshold. If set too high, objects will be missed; if set too low, too many spurious objects will be located. One should experiment with this parameter whenever dealing with some new type of data (whether CCD or S/N) for the first time. Note that this threshold is in local sky rms, rather than global (which was the case for earlier version). The local sky rms threshold produces much better results, especially around edges and bright halos of stars and galaxies where the rms values could be much higher; but at the cost of longer CPU time. A global sky rms can be used by entering *thresky* as a negative number.

(5) *conthres* – the connectivity size for accepting a candidate object, expressed as aperture diameter. Pixels within 1/3 of *thresky* around a candidate are connected. If the size is smaller than *conthres*, the object is rejected. Typical value: 3.

(6) *smsig* – the condition for smoothing before doing the pixel connection test. If the object has flux less than *smsig* × sky sigma, the section around the candidate will be smoothed. Typical value: 4.0.

(7) *iobjsm* – type of smoothing used for connectivity test. 0 is for simple box car. It turns out different smoothing shapes has little/no effect on the algorithm.

(8) *imerad* – radius for merging closely spaced detected objects. Any objects within *imerad* pixels of each other will be merged (by computing the centroid of the set of objects). This is automatically done after every object finding routine. Typical values: 1 or 2.

2.2.2.2 *An Example Batch Job File for Object Finding*

The following is an annotated batch job in which most of the features described above are incorporated. The # sign indicates a comment. This example does not include using weighted average of two filters for the object finding, and also assume that there are 3 images available in the buffer. In the case when only 2 are available (certain to be the case for Mosaic CCd data), some shuffling of images, including writing temporary images out to disk to be recopied into the buffer later on may be required.

```
# Set up parameters, see section 2.2.2.1
set iecho 1
set inoise 0
set adusat 21000
set iwd  5
set conthres .3333
set smsig 5
set imerad 2
# smoothing for CR/noise spike check: iobjsm=0: box, 1: tapered box
set iobjsm 1
# set up diffration spike pattern:
#  a0+a1*x+a2*x^2, where x=ADU counts of star in 10^7 ADU units
set dsc
12 12 10
# angle of diffraction spike, positive = top to left (anticlockwise).
set difang 49.3
# read in image called image002.fits, use rdis, or rdf
rdis 1 image002
# replace all negative pixels by saturation level pixels
# input line 1: ADU threshold below which pixels are replaced
# input line 2: replacement value
rple 1
0
22000
# create bad column map in buffer 3
# input line 1: mask buffer number (cannot be on the command line)
# input line 2: I,J size
# input lines: coded entries for mask positions
```

```
# last line: a blank (important!)
crbp
1
2048 2048
b 180 1 183 300
b 184 1 186 100
b 187 1 193 75
b 459 1 462 4093
# create bleeding column and diffraction spike map.
# In this case, the saturation stars are found using ppp; see Section 2.
# set the threshold high, and use negative to use global sky (faster).
set thresky -100
# probably good to use a somewhat large sky ring for sky
set iir 15
set ior 29
# set the saturation adu value to near the top of the saturated values
set adusat 28000
# satf inputs
# line 1: output .pos file name
# line 2: comments
# MUST have 2 blank lines at end (lines 3 and 4)
satf 1
0920b2sat
0920 field b2, saturated stars
# make bleeding columns and diffraction spike
# bldm inputs:
# line 1: mask buffer number
# line 2: padding: 0= no corners, 1=corners padded
# note that diffraction spikes are set by dsc and difang.
# Set dsc's to 0 if no diffraction spike masking is desired.
bldm 1
2
0
# sum the bad column (2) and the bleeding column (2) map, result in buffer 2
sum 2 3 2
# smooth the image with a tapered 3X3 box
tsmo 1
3
# set the threshold for real object finding, also narrow the sky ring a bit
# note that if thresky is not set to a non-zero number, afim will ask for
# a threshold, but in different units.
set thresky 2.8
set iir 11
set ior 19
# set the object finding area limits, defined by the smallest and largest IJ.
# if no limits are set, the whole image is used.
```

```
# plfi input:
# line 1: smallest IJ
# line 2: largest IJ
pfli 1
101 101
2000 2000
# object finding
# afim inputs
# line 1: output .pos file name
# line 2: comments
# line 3: mask frame number (this line not needed for afin)
# line 4: pixel diameter for connectivity test
# line 5: smoothing box size
# line 6: number of ghost images per star (leave blank for none)
# additional N lines: N delta pixel x and y of ghost(s) from star centroid.
afim 1
0920b2tem
0920 b2 raw obj list
2
3
3
0
# now do additional cosmic ray rejection, need to read back unsmoothed image.
rdis 1 image002
# replace negative pixel values by saturation value.
rple 1
0
22000
# reject cosmic ray detections, a file for reference reject1.pos is created.
rejc 1
# save the position file again, first go into posf menu section
posf
save 1 0920b2tem
0920 b2 raw obj list, CR rejected
# come out of posf
q
# remove the rejection pos file if you don't want to see it
# exit ppp and compress images
quco
```

2.3 Editing the Position File into the Final Master File

For most images, if smoothing is applied and if the parameters are set up appropriately for the image, almost all objects detectable by eye will be detected by **afim**. Although over the years the object finding algorithm has been modified and improved repeatedly, perfection is rare! There are two types of errors that may require visual inspection and manual editing of the position file: missed close doubles, and spurious objects from cosmetic imperfections that were not masked out properly. The main effect of the former is that stars with a close unmarked companion (either another star or a galaxy) may be misclassified as resolved, while the latter may add unwanted objects to the catalog. It is up to you to determine if your scientific objectives may be compromised if such editing is not carried out.

Here is the procedure for checking the objecting finding using SAOimage. First the image is read in and displayed (**rdf** and **sao**). The position file is read in (**rpof**), and the command **scoo** *n* is used to create a .reg file by the name of *saoimage***n***.reg*. If you read in the position file prior to displaying the image, the .reg file will be automatically created. On the SAOimage canvass, read in the .reg file, and the marked objects are displayed as red squares. You will want to remove the labels (**label** button) to facilitate seeing the markers. There are two types of markers: the large ones are the objects found by **ppp**, and the small ones are the ones added by hand (if the file has been edited before). Also displayed is a grid pattern in yellow lines with the center of each grid marked by a yellow cross. The grid is made to facilitate an orderly examination of the frame. With the proper zoom, one can examine one grid retangle at a time.

There are two basic operations: deleting objects, and adding objects. First under **cursor** (and **region**), choose the circle cursor marker. To delete an object, point the cursor and type **d**. To add an object, point the cursor, click the left mouse button to move the (green) cursor marker, and type **e**. A red circle will mark the object you have just added. Note that both must be done under the **cursor** menu.

Some hints on efficient use: for each grid, first do all the deleting, then the adding. The objects that are important to delete false objects around bright galaxies an stars (which are rare) and bleeding columns and diffraction spikes (if they are not masked properly). Around a very large galaxy, you have to make the decision as to whether a marked object is in fact just a HII region or a chunk of a spiral arm. Occasionally you may want to delete false doubles (look for markers that are very close together). If you have too many false doubles, it may mean you have not set *imerad* large enough, or your images are so oversampled that you have not smoothed it enough. Do not spend time checking if *every* object marked is real. If they sit in the middle of nowhere (i.e., not in the halos of bright objects), they will simply get no flux and be classified as 0.

For adding objects, the most important thing is to look for missed close doubles. I find turning the markers on and off (**view** under **cursor**) is extremely useful to see whether a possible object should be included. Sometimes you might want to change the contrast, or even the look-up table range (**new**) to see things clearer. If your image has a strange way of saturating, sometimes badly saturated stars will be missed, it is important to add those back in. After going through the image, I usually dezoom the image by a factor of 2, and look at the big picture to see what else may be missing. The human eye picks out different things under different zoom factors.

It is important that you do all the following steps for saving the new position file properly, so as not to lose all the work that you have done in editing the position file! When you are satisfied with the result, write out a new *saoimage***n***.reg* file (under **cursor**, **region**, **write**). You should overwrite the previous file. In **ppp** use the command **spos** *n* to read the .reg file to update the position file. (Ignore the output about 0 object deleted.) Go into **posf**, use the command **merp** to merge any closely marked objects (usually due to human error of entering an object twice). Then use **save** to save the new position file.

2.4 Transforming the Master Position Coordinates to Another Image

If the images from different filters are not registered exactly, then a transform of the master position file is needed before photometry can be done on the other filters.

There are two methods of transforming the position file. The simple method of simply shifting in x and y is applicable for images that were taken with the same detector and have no differential distortion or rotation between them. In this case, use the command **shift** in **posf** to shift the master file, after determining the relative shifts in x and y in the two images by looking at some bright reference (non-saturated) star. There are also commands in **posf** to rotate and scale the position file, which can be used to transform position files for images with simple non-distorted images.

In the case of differential distortion, or much more complex situations, (e.g., MOS images, images from IR and optical detectors), a more generalized transformation process should be used. This is a program, written by Peter Stetson for DAOphot, which compares two position files, find a set of matching objects, computes a 2-D transformation, and apply the transformation to the whole file. Let the two images be A and B, with the master file coming from A, and you want to apply the master file to image B. First, a position file should be created from B, typically using a higher threshold. Then objects in B and A are matched, and from the matched objects, a transformation from A to B is formed. The transformation is then applied to the master list from A, and now photometry can be done on B using the transformed master file. The command for do this is **matp** *n1 n2*, where n2 is the frame number containing master list, and n2 the temporary matching list. No image is required. **ppp** spawns a task to run Stetson's program, and the following setenv statement must exist:

setenv POS_MATCH "*pathname*/match.csh"

(setenv POS_MATCH "/d/makalu/hyee/bpppx/match.csh" in Toronto)

The file match.csh contain a shell script to run the program **daomaster**. The shell script should be edited to reflect the path of **daomaster** in your system. It should be noted that the program expects the objects common to the two files to be within 10 pixels of each other. If this is not the case, a preliminary scaling, shifting, and rotating must be done to the master file to bring the two to close proximity. In this case, positions for 2 reference objects in each frame should be obtained. The command **mttp**, which requests co-ordinate of the reference objects, should be used.

The following is an example batch job of creating a matched master position file for another filter. This example assume a master file was created using a deeper R image, and one wants to match it to the B image.

```
# set parameters: similar to object finding
set ... etc
# obtain a rough object position file from the B band
rdis 1 A2390B
# set up bad column map if any, otherwise use afin
crbp
1
2048 2048
b 180 1 183 300
b 184 1 186 100
b 187 1 193 75
b 459 1 462 4093
# set threshold, should be relatively high
set thresky 4.0
# objects finding, (assuming there isn't already a pos file).
afim 1
A2390Btem
A2390 B, temp pos file for matching
2
```

```
3
3
0
# read in the master pos file, note that if the image is sectioned,
# frame 2 needs to have the section number set manually by sect 2 n
rpof 2 A2390ml
# if the two images are reasonably close in position, use matp,
# otherwise uses mttc.
# the current pos file in 1 is used to compute the transformation,
# the position file in 2 is moved to 1 and transformed.
matp 2 1
# save the transformed master position file, now in the B frame system
posf
save 1 A2390Bml
A2390 master list for B, matched from R
# exit posf
q
# exit ppp and compress
quco
```

If you need to run the more generalized **mttp**, here is an example:

```
rpof 1 posfile1
rpof 2 posfile2
# mttc inputs:
# line 1: pixel size of frames (in order of the mttc command line input),
#         this could be the actual pixel sizes, or any 2 numbers in the
#         proper ratio. In this example, one CCD has a pixel size of 15μ
#         the other (frame 1, the 2nd input frame) 21μ.
# line 2: IJ co-ord of first obj in the first input frame
# line 3: IJ co-ord of first obj in the second input frame
# line 4: IJ co-ord of 2nd obj in the first input frame, this is needed
#          only if there is a rotation. If not needed, enter 0 0
# line 5: IJ co-ord of 2nd obj in the second input frame.
mttc 2 1
15 21
324 567
457 678
0 0
0 0
```

III. FAINT-GALAXY PHOTOMETRY

3.1 Introduction

There are a large number of photometry commands. They can be divided into three categories: fixed-aperture photometry, variable-aperture photometry, and surface photometry. The first two categories can be further separated into operations involving a single object or a file of object positions such as that created by the procedure in Section 2.

Fixed-aperture photometry is relatively straight-forward and simply involves an input of co-ordinates and aperture size. The various commands are listed in **help** and will not be discussed further here. Surface photometry is still in a somewhat experimental stage, and will be discussed in Section 4. Variable-aperture photometry, designed specifically for faint-galaxy photometry in moderately crowded fields, involves the most complex algorithm, and is described in detail in the following subsections.

3.2 Faint-Galaxy Photometry

There are three main concepts used in the faint-galaxy photometry algorithm which do not exist in the simple aperture photometry program. These are: provisions for dealing with close neighbours, growth curve analysis, and star-galaxy classification. The various commands for faint-galaxy photometry attend to these points with varying complexity and sophistication. In this manual only **apph**, the most automatic and most sophisticated one, will be described. The reader is referred to to Yee (1991) for the basic photometry method of **apph**, and to Yee et al. (1996) for the variable star-galaxy classification algorithm.

3.2.1 General Description of apph

There are 5 stages in the photometry algorithm. They are all described in detail in Yee (1991). First, the positions of the objects are refined so that they represent the closest integer centroids of the objects. Next, for each object, neighbours within 2 times the maximum radius of the growth curve aperture are masked according to the relative brightness of the objects. The third step is to produce the growth curve for each object by integrating the net flux within concentric circles centered on partial pixel centroids. Fourth, the growth curve is analysed to determine the maximum radius of integration. Finally, a reference PSF growth curve is created from bright reference stars and the growth curves of all objects are compared with it to compute the classifier, C_2, for star-galaxy classification. All the pertinent information is then stored in the output **.phm** file, and normally, a C_2 versus instrumental magnitude plot is created.

The usage of the photometry program is very easy and almost completely automated. However, because of the large amount of automation, there are subtle complexities that affect the results that one must be able to recognize. The photometry program is controlled through the setting of a dozen or so parameters, most of which can be left at their default values. The ultimate test for the correct settings for the parameters is the analysis of simulated data (see Yee 1991).

3.2.2 Variable PSF (incomplete)

Almost all CCD images covering a field of more than a few arc minutes have a PSF that varies across the field. This problem is especially severe in images obtained through a focal reducer or from the new large field Mosaic CCD cameras. Needless to say, a variable PSF has a great impact on star-galaxy classification. **ppp**deals with variable PSF across the image by using local PSF's as references for star-galaxy classification. The nearest star(s) in each quadrant, defined by orthogonal lines from the object to be classified, are chosen and averaged to form a local reference PSF for the object.

There are two main parameters for setting up the variable PSF classification. *npsfsec* determines which type of PSF routine is used. This should nominally be set to -1 for variable PSF algorithm. *nvpsf* sets the number of PSF's in each quadrant to be used, with 1 or 2 usually recommended. The value 1 is best for images rapidly changing PSFs and adequate in almost all instances, while 2 is recommended if the field contains many high surface brightness, compact galaxies, such as the core of a rich cluster.

The set of stars chosen for the variable PSF classification algorithm is normally created automatically using one of two commands. The command **aref** is a totally automatic algorithm for picking objects suitable to be PSF references, while the command **ref** under the menu **dsut** allows for iterative inspection of the selection process. Both commands produce a .ref file for input into the photometry command.

3.2.3 Parameters

The description of the parameters is divided into two sections. The first one describes those that most frequently need changing; the second set, occasionally need varying. There are many remaining parameters, such as those which control the way the masking is done. Those are not described, and are used only for detailed testing in "advanced **ppp**ing".

(1) *napcen* — this sets the aperture diameter in pixels for the aperture used to determine centroids. The nominal setting is either 3, 5, or 7. Normally, it should be set to between 1 and 1.5 FWHM. It should never be less than 3. Note that this aperture is also used as the default for the automatic centroid refining algorithm (which should be closer to 1 FWHM). The photometry command queries whether the *napcen* default should be used, and it can be over-ridden at that point if a smaller value is desired.

(2) *adusat* — the saturation level in ADU. This must be set to the proper value for correct classification. Objects containing pixels greater than *adusat* will be automatically classified as class 4, (saturated stars), and are omitted from the reference star list. Note that occasionally when averaging 2 images, a bright star may be saturated in one frame but not the other (which occurs especially frequently when the images are under-sampled); in the averaged picture, the star may have apparent counts below saturation but in fact is saturated. Thus, setting *adusat* too close to the real saturation value may be dangerous. A good way to check for saturation is to examine the bright stars in the C_2 plot.

(3) *ise* — the aperture number (NOTE: make sure that you distinguish between aperture number and aperture diameter which is in pixels) for determining one of the two error estimates. This one produces an uncertainty for a specific small aperture whereas the other error estimate pertains to the maximum aperture adopted by the growth curve analysis (which is usually bigger). Normal setting for this is the aperture number corresponding to about 2 to 3 FWHM, usually $ise = 2$ or 3.

(4) *iapis* — sets the size (in pixels) of the ring area for the determination of sky value. It is the diameter of the outer circle minus the inner circle. It should be set at a reasonably large number, such as $15''$ to $20''$ larger than the largest aperture.

(5) *ideld* — Δ diameter in pixels between successive apertures. Normally this should be 1 or 2. The point is to set this so that the maximum number of apertures should be as close to, but not over, 20. Thus, the smaller the pixel size is the larger *ideld* is. For example, for a $12''$ maximum aperture, *ideld* should be 1 for $0\farcs75$ pixels, but 2 for $0\farcs4$ pixels.

(6) *isdap* — aperture number for zero point determination for comparing growth curves to reference growth curve. Only 1 and 2 are allowed. Normally, for undersampled data (FWHM$<$ 2), it should be set to 1, and for oversampled (FWHM$>$ 3), it should be set to 2 (which actually is $1\frac{1}{2}$).

(7) *bgfrac* — this sets the factor of increase in the maximum aperture diameter for bright galaxies. Bright galaxies normally are much larger than the maximum diameter set for the growth curve, and hence their luminosities are underestimated, sometimes by as much as half a magnitude. The program, at the end of the photometry, goes back to the object list and checks for bright galaxies (but not stars), and re-does photometry for all of them using a maximum diameter of *bgfrac*×(original maximum diameter).

(8) *bgmag* – All galaxies brighter than this INSTRUMENTAL magnitude will have their photometry redone with the enlarged maximum aperture. It is found that −12 to −12.5 usually works fairly well.

(9) *iapmin* — sets the minimum (i.e., first) aperture diameter in pixels. This is usually set at 3 (minimum allowable value). In cases of extremely oversampled data, a larger value can be used.

(10) *skcmed* and *skcmean* — these set the estimator for sky values from a combination of the median and the mean of the sky pixels:

$$\text{sky} = skcmed \times \text{median} - skcmean \times \text{mean}$$

Although the normal accepted modal sky estimate is 3*median–2*mean, it is found through simulations that this, while it predicts the sky value more accurately, nevertheless systematically produces magnitudes too bright when compared with the true input. The reason is that one does not really want to predict the exact sky value, but the "background" value which includes faint unseen objects which would contribute to the flux of the object if not subtracted. Simulations show that using $skcmed = 2$ and $skcmean = 1$, which over-estimate the sky, produces more accurate and consistent answers.

(11) *irefmin* — the minimum number of reference stars used for computing C_2, the star-galaxy classifier.

(12) *npsfsec* — This sets the method used for automatic procedure for determining the standard PSF for classification. If *npsfsec* is set to 0 or 1, the standard PSF is determined by averaging all usable bright stars in the field. Other settings are for variable PSFs. For *npsfsec*$= -1$, a file of specified reference stars is to be supplied by the user (created by the command **ref** in **dsut**. For each object, an interpolation of the nearest 4 reference stars is used to create the comparison PSF for classification. For *npsfsec*$= n$, where $2 \leq n \leq 5$, the image is cut into n sections and individual PSF references are created (automatically) within each section. Note that both variable PSF methods require a large field where there are many bright unsaturated stars.

(13) *rrefmax* – sets the maximum distance in terms of the fraction of the frame size within which a reference star is accepted for determining the reference PSF for classification for *npsfsec*$= -1$.

(14) *nvpsf* – sets the number of PSF per quadrant used to form the average local PSF for star-galaxy classification. Nominal values: 1 or 2. Maximum 4.

(15) *sseqmag* –

3.3 Using apph

The command **apph** is the most advanced of the various photometry commands in **ppp**, optimized for faint galaxies in reasonably crowded fields. **apph** will either choose its own PSF, or accept a file of preselected PSF. In the former, it is not able to deal with variable PSF. In the case of variable PSF, a PSF file must be created either semi-automatically (via **ref** in *dsut*) or completely automatically via **aref**. It is recommended that the crowded-field photometry commands be run as a batch job under most circumstances. The normal procedure for photometry involves reading in the frame(s); performing cosmic fixes and other arithmetic operations, if necessary; setting up all the relevant parameters; reading in the master position file, transforming it if necessary; and then running **apph**. A typical example is shown at the end of this section.

apph asks several questions. It begins with requesting the name of the output file and comments to be put in the header of the output photometry (**.phm**) file. Then it queries the maximum aperture diameter in pixels. (E.g., Yee and collaborators use $12''$ for all their galaxy counts.) Normally this is rounded-off to the next largest integer pixel diameter. The program then performs the iterative centroiding routine to derive the best integer centroid for each object, and it requests information several times for this procedure (see example). This re-centroiding is necessary because often when the same master position file is applied to different filter frames, the partial pixel shifts may create small differences in the best integer positions. The program proceeds to perform photometry, and two choices are offered for creating the PSF. The program selects the reference stars itself, or if a reference star file name is entered, it will then use that file. Finally, a C_2 *vs* mag plot is created and the results are saved in a .phm file.

The following is an annotated example of running **apph** as a batch job.

```
# set parameters
set iecho 1
set hplot /ps
set inoise 1
# adusat should be set conservatively to make sure saturated stars are
#   recognized as such
set adusat 24000.
set ise 2
set napcen 3
```

```
set ideld 2
set imrobj 1
set isdap 1
set iapis 36
set bgmag -11.25
set bgfac 3.0
set npsfsec -1
set rrefmax .35
set nvpsf 1
# do not use diffraction spike map in bpfi
set dsc
0 0 0
# read image
rdis 1 A2390R
# replace all negative pixels by saturation level pixels
# input line 1: ADU threshold below which pixels are replaced
# input line 2: replacement value
rple 1
0
22000
# fix bad columns and bleeding columns by interpolate across them
crbp
3
2048  2048
b   299 1045 300 1046
# bleeding stars: read pos file, create bad pix map (bldm), fix bad pix(bpfi)
# note that one can also just read in the master list pos file.
rpof 1 A2390Rsat
bldm 1
2
2
bpfi 1 2
# read in master list position file
rpof 1 A2390ml
# make auto reference PSF file, if needed
# input line 1: threshold above sky in units of ADUSAT
# input line 2: aperture diameter for growth curve, should be the same as apph
# input line 3: output file name, .ref appended.
aref 1
.1
29
A2390R
# do photometry, inputs for apph:
# line 1. output .phm name
# line 2. comment
# line 3. max aperture for growth curve in pixels (odd)
```

```
# line 4. iterative centroiding: y(es)
# line 5. pixel diameter for centroiding:
#         recommended: 3 or 4 for the primary image, 4 or 5 for other filters
# line 6. max shift allowed in interative centroiding:
#               recommended: 1 for the primary filter image, 2 for others
# line 7. .ref file name  (if npsfsec set to -1).
apph 1
A2390R
A2390 R 600sec
29
y
4
1
A2390R
# next filter, B
rdis 1 A2390B
rple 1
0
22000
# read in B filter master list, if created by matp.
rpof 1 A2390Bml
# If there is simply a simple shift between the two filters,
# this can be done more economically by shifting the original ml.pos file:
#      rpof 1 A2390ml
#      posf
#      shift 1 2 -1  ! the shift is, say, 2 in x and -1 in y
#      q
crbp
3
2048  2048
b   299 1045 300 1046
bldm 1
2
2
bpfi 1 2
aref 1
.07
A2390B
apph 1
A2390B
A2390 B 1500sec
29
y
4
2
A2390B
```

exit and compress image
quco

For Unix machines, a “batch” job can best be run as a background job. To do this, create an input file (say, abc.pppin) and then the job can be started using:

ppp < pppin.abc > abc.pppout &

The output of the background job is in abc.pppout. Note that in some unix set up, if abc.pppout already exists, it will cause the job to crash.

3.4 Outputs from Photometry Commands

The multi-aperture photometry commands produce a files, **name.phm**, which contains a 2-line header plus 3 lines of information per object. The header contains the following information.

Header line 1: comment, date of creation, list of reference stars #'s, C_2 criteria

Header line 2: sky aperture, 1^{st} aperture diameter, last aperture diameter, size of the frame, *ise*, extrapolation aperture (not necessarily the largest aperture), estimated FWHM.

data line 1: object number, centroid x, centroid y, best growth-curve total instrumental magnitude, magnitude extrapolated to extrapolation aperture, S/N in aperture *ise*, S/N in adopted aperture (IDS), interpolated peak value (do not use), number of neighbours, empty slot (for **auph**, ellipticity), empty slot (for **auph**, position angle), sky, rms of sky per pixel, C_1 (See Green and Yee 1984), classifier 1 (not used), C_2, classifier 2.

data line 2: integer position, IDOB (0 if found by object finding routine; 1, if added manually), instrumental magnitudes of first 10 apertures, IDS (adopted aperture).

data line 3: *BG denotes large aperture used, (if *BG, next 2 numbers are adopted aperture size, and aperture number), instrumental magnitudes of next 10 apertures.

3.5 Producing the Final Object Catalog (incomplete)

PPP commands create a catalog of instrumental magnitudes. An additional step is needed to produce the final catalog of objects by calibrating the photometric data and computing the astrometric positions of the objects. Although I have a separate program, called **mspos** and tailored to my own scientific programs, to do this, it will not be described in this document. The purpose of this section is to alert the user on what to look for in the photometry data to catch obvious errors which may affect the result.

There are basically 2 major problems that a galaxy photometrist should be on the lookout for: (1) bright stars mis-classified as galaxies due to various reasons; and (2) faint objects confused by the halos of bright stars or galaxies or diffraction spike causing them to have too large an adopted aperture and hence too bright a magnitude. Both of these are serious problems as they affect the statistics of the relatively small number of bright galaxies significantly, if uncorrected.

IV. MOSAIC FORMAT IMAGES

The advent of large mosaic CCD images has created significant challenges for simply structured program such as **ppp**. The increase of image size from 2K to as large as 12K has taxed the memory capability of the Fortran language so that a simple brute force increase in array size is currently not possible. Version 4 of **ppp** deal with this problem using a relative inelegant solution of subsectioning any image larger than 4500X4500 into smaller sections before feeding it into the program. The object finding and photometry are done using these subsections. The headers of the position and photometry outputs contain all the subsectioning information required to reconstruct the overall catalog in the global pixel co-ordinate. The subsectioning of the image is done with edge padding of 100 pixel so that there is no edge effect at the boundaries of the sections. To read an image that require sectioning, use **rd8k**, which operate like **rdis**, but requests the section number.

The size of the sections are set by *nsi* and *nsj* with the current maximum being 4400 and the default being 4100. The edge is set by *nedge*, with a default of 100 pixel. Once a frame within **ppp** is recognized as a section, the section number is displayed in the directory, and all input and output files (e.g. .pos, .phm, .pc2, .fix files) will have the section number appended.

V. SURFACE PHOTOMETRY

5.1 Azimuthally Averaged Surface Photometry

There are 2 commands for doing surface photometry using concentric circular apertures, **spho** and **ssph**. The latter allows one to measure azimuthally averaged profiles in a sector of a circle.

Before running any of the surface photometry commands, the centroid of the object has to be determined. (I.e, find the integer pixel centroid, and then use **cent**). The command first asks for a name of a file for the output of the profile data. It then requests the partial pixel centroid of the object. Next, it queries for information for the creation of the apertures to be used. It does this by first requesting the largest diameter (in pixels), and then a series of input of diameters and increments. For example, an entry of "1 2" sets up a series of apertures with increments of 2 pixels in diameter, starting with the diameter of 1 pixel. This aperture setting will continue until the next aperture set. Suppose you enter "25 6" as the second in a series, then the apertures will have increment of 2 from diameter 1 to 25, and after that, the increments will be 6 in diameter. (I.e., the diameters are 1, 3, 5,... 23, 25, 31, 37... etc). You can enter as many different sets of apertures as you wish until the maximum aperture is reached. Of course, the idea here is that you increase your aperture ring size at increased radius to compensate for the lower S/N at the edge of the galaxy. The next information requested is the sky value. Here, you can either measure it using the geometric shape options offered (e.g., ring, borders of rectangle etc), or you can enter your own numbers. The latter is recommended if your object is very large, or if there is some question about the flatness of the image over the size of the galaxy. You can pre-determine the background using a variety of methods, e.g., average of a bunch of boxes or circles around the galaxy etc; see command **sky**.

At this point, we take an aside to discuss error determination. Often one wants to push surface photometry down to a surface brightness of considerably less than σ_{pix}, rms per pixel for the sky. The idea is that by averaging over a large annulus, the error is thereby reduced and the major contribution comes from the actual systematic uncertainty in the sky value used (rather than photon statistics). If the picture is perfectly flat, then the sky can be determined to great precision by using a very large area. However, if the sky is not flat, caution is required. A reasonable method is to determine a set of sky values using small apertures around the galaxy. The dispersion or range of sky values can then give an idea of the accuracy of the sky determination. In the case of averaging over the full 360^o, the non-flatness of the image is less of a problem in that the systematic error from one side will be compensated to varying degrees by that from the other. However, this is not true for either **ssph** or **isop**. All the surface photometry commands use $\sigma_{\rm pix}/(N_S-1)^{\frac{1}{2}}$ as the default systematic sky uncertainty, where N_S is the number of pixels used to determine the sky. (I.e., this assumes a perfectly flat image.) The user is allowed to over-ride this by entering what is deemed appropriate. The program further adds the uncertainty due to sky photon fluctuation within the aperture and the photon noise from the galaxy within the aperture. For the latter, the parameter *ccdgain* is used. However, normally the contribution of the noise from the galaxy photon statistics is very small, and most of the time, the gain can be left at its default value regardless of the real value.

After all the information concerning sky values and uncertainty has been entered, the program then produces a series of surface brightness output. (To be described, since it is being changed at this point). As with total photometry commands, the output surface photometry is in instrumental unit; i.e., instrumental magnitude per square pixel. It is up to the user to create programs for further analysis of the data.

The command **ssph** operates in exactly the same manner with the exception that it requests the sector of the galaxy to be used. (Note that **ssph** will warn you that it will use the last frame as a storage for the map defining the sector.) The sector is defined by a position angle and the full-width of the opening angle of the sector. The position angle is defined as positive for counter-clockwise from the 12 o'clock position on the image. (See Section 1.4 for orientation of images.)

5.2 Isophotal Surface Photometry

The isophotal surface photometry command, **isop**, creates a map of apertures by connecting pixels belonging to certain surface brightness levels (i.e., creating maps of the contours). Because of this procedure, often one may want to consider smoothing the image first. This will allow more accurate results at the faint levels since the program needs to determine the connected area of brightness above a designated level, and in region where sky noise dominates, inaccurate answers would result. However, in this case, for the best result, the profile near the center may have to be done separately with full resolution and then combine the two results later.

This command requests the integer centroid instead of partial pixel centroid. It then requests information about sky and error the same way as the **spho** command. Next, it queries the number of levels of isophotal surfaces and the values of the levels in ADUs. The four options for the levels are: (1) user input, in which you enter whatever you like (as long as they are decreasing); (2) previous values – whatever that was used last time (of course, except the first time you are using it in a **ppp** session); (3) linear – linearly divided levels starting from the peak (I.e., each level will be ADU_{peak}/N, where N is the number of levels desired); and (4) logarithmic – the same as the *s* option of the contour plot command; i.e., the levels are defined starting from the peak with each level being *cplevel* times the previous level, (also see *cplx* in §1.7). The fourth option usually works the best if you want to get down to levels comparable to the sky noise. Note that **isop** works only with rounded-off integer ADU levels. You are then requested to enter a file name to store the result. The output consists of: level number; ADU value of the level (with an integer sky ADU subtracted); instrumental magnitude per pixel (computed with the actual fractional ADU sky value subtracted); uncertainty; the centroid of the isophotal contour (weighted with each pixel equally; so it is NOT an intensity weighted centroid); the equivalent radius of the isophotal surface [i.e., (total pixel $/\pi)^{1/2}$]; major axis of the best fitting ellipse; fitted minor axis, and the ellipticity. (The procedure of fitting the ellipse is that of Dodd and MacGillivray 1986, *A. J.*, **92**, 706; but note that there are errors in their equations as published.) The second line of output for each level consists of the total number of pixels, the total integrated net counts within the enclosing isophot (i.e., not just between this and the previous isophot), the instrumental magnitude enclosed by the isophot, and the uncertainty. Note that, again as in the photometry commands, **ppp** produces instrumental magnitude data only, and it is up to you to write/use other programs to convert it into whatever usable form or graphic display you need.

There are several points of consideration in using **isop**. The first is that contaminating objects (especially "large" objects) can cause severely incorrect results. The problem occurs when an isophot level suddenly got connected with the offending object. Second, at faint levels, sky flatness becomes extremely important. Do NOT go down to levels with brightness approaching the amplitude of non-flatness in the background over the size of the galaxy. Always examine the image closely to evaluate whether the faintest levels are to be trusted.

VI. PSF SUBTRACTION

6.1 **Overview**

PPP uses a non-parametric method for the subtraction of PSF. The main application is for deconvolving a point source from an extended source, as in the decomposition of the quasar component from its host galaxy. It is not intended to be used as a "stellar" photometry algorithm (although it works reasonably well for that too). A detailed description of the method can be found in Yee and Green (1988, *A. J.* **94**, 618). A more complex use of the algorithm can be found in Yee (1988, *A.J.*, **95**, 1331) in which the four components of the lens 2237+030 are decomposed from the lensing galaxies. The basic principle of the method is to subtract the largest PSF possible from the object without producing a negative radial gradient towards the center of the residual object (which is assumed to be a galaxy). Due to the nature of the problem the command is intended for, the algorithm uses only information from the central few pixels (as opposed to using the whole PSF shape). The reason is that in an object with an extended fuzz component (i.e., the galaxy), the central part will be least contaminated by the fuzz component. An algorithm which makes the best fit of the PSF to the object (e.g. DAOPHOT) may not be suitable since it will tend to over-subtract as it also tries to maximize the fit on the wing.

6.2 **Setting up the PSF**

There are 3 steps in doing the subtraction. First, both the PSF and the object have to be shifted in such a way that they are both sampled in the same manner by the pixels. A PSF is chosen and re-sampled (i.e., shifted) so that the 4 pixels immediately around the central pixel is symmetric in flux. This means that the integer co-ordinate of the central (peak) pixel is the exact centroid co-ordinate. (In practice, this is not achievable exactly, as it is almost impossible to have PSFs perfectly symmetric throughout.) This is done via **cpsf** which centers the PSF. The shifting is done by a sinc function interpolator (as used in the command **sshift**). This method of shifting preserves both the noise and shape characteristics of the image perfectly (i.e., does not smooth the image like bi-linear interpolation as in the command **shift**). However, it will work properly only if the image is sampled at least at the Nyquist frequency, more than 2.4 pixels per FWHM. For an undersampled image, ringing will occur. Furthermore, any cosmic ray detections, cosmetic imperfections will also produce ringing. Thus, the image may have to be prepared by fixing such bad pixels around the object to be **cpsf**'ed by using **crbp** and **bpfix**.

The command **cpsf** cuts out a small part of the image around the object and places it in another frame (to be requested). The size of the created frame is $2\times$ *isincf* + 1, where *isincf* is the size of the sinc function filter in pixels. The parameter *isincf* can be changed by **set**. The command then asks for the integer centroid of the object and the sizes of inner and outer apertures to be used for centroid determination. It then proceeds to shift iteratively the central part of the cut-out image attempting to center the peak on the central pixel of the image exactly. After 5 iterations, it offers the user several options. One should continue to iterate until the next-guesses of additional shifts in both directions are identically 0 (i.e., a perfect solution is found), or begin to oscillate between identical negative and positive values. At this point, the user should enter option *s* to shift the whole cut-off image and exit the command.

The second step is to repeat step one for the object to be subtracted; thus creating two images that are sampled in exactly the same manner. After this, there are 2 optional steps. The first is the option to smooth the wing of the PSF using **smpsf**. This command uses a program written by Daniel Nadeau (Univ. de Montréal) to smooth the wing of the PSF by azimuthally averaging over sectors of specified angle of rings with width thinner than one pixel. This in general should be done unless the PSF is considerably brighter than the object to be subtracted. However, this should not be done if the PSF is crowded by another object (which should not be used in the first place!). In practice, it is always useful to display the intermediate results before proceeding to the next step. The command asks for the (partial pixel) centroid of the PSF and then a series of radii and angles. The smoothing is done for the area between the first and last radius entered. For the area between successive radius entries, the angle input tells the program the azimuthal angle of the sectors over which smoothing is done. In general, it is good to use small angles for small radii, and increase the angle as one goes out. It is good practice to inspect the PSF to determine at what radius smoothing may be useful. Note that you must NOT smooth the inner 5 pixels of the PSF, otherwise the PSF subtraction will be incorrectly done! The command lastly requests the size of the ring strip over which smoothing is done. A number of PSFs can also be averaged after **cpsf** to obtain better S/N ratio in the wing. (But be aware of the warning about non-linearity discussed later in this section.)

6.3 **Iterative PSF Subtraction**

You have the option of performing the PSF subtraction on the cut-out image of the object, or the original complete frame. In the latter case, you should do a sinc shift (**sshift**) on the whole image with the shift determined from **cpsf** for the object. (Note that this will take a long time.)

To perform the PSF subtraction, the command **psfs** is used. After answering a series of questions concerning the positions and apertures to use, the command requests information concerning the subtraction. First it asks for a diameter of the region to be subtracted. The user thus may want to examine the image of the PSF before hand to determine how large a diameter to use. Second, it requests the scaling factor of the PSF. For most applications in which the user wants to find the answer, a 0 should be entered here. Lastly, should 0 be entered for the previous input, the program requests a gradient test method. The algorithm determines the best subtraction by iteratively subtracting a PSF scaled by different factors until just before the gradient towards the nucleus of the residual object turns negative. For testing gradients along each direction individually, an integer representing the radius over which the gradients are formed should be entered. This should normally be 1, or sometimes 2. In this case, the gradients along 8 directions are computed, and all 8 are tested. To invoke the option of selectively testing only gradients along certain directions, enter the radius as a negative number, after which the program will request further information. Under most circumstances, with the exception when both the PSF and the object are of very high S/N ratios, the other options of testing an average gradient give much better solutions. The options $m1$ and $m2$ uses the median of pixel values in rings around the center to apply the gradient test. $m1$ compares the peak pixel with the median of the 4 immediate neighbouring pixels, whereas $m2$ compares the central 5 pixels with the surrounding 8 pixels. The options $a1$ and $a2$ are the same with the exception that averages instead of medians are used. The final output of the command consists of the fraction of the peak of the object subtracted and the scaling of the PSF for the best solution. The scale is also expressed as Δ magnitude – i.e., the difference in magnitude between the PSF and the point component of the object.

6.4 **Accuracy and Uncertainty**

PSF decomposition is a difficult operation especially when the object in question contains only a faint underlying extended object. There are several considerations that the user must keep in mind. First, the uncertainty in the decomposition depends on the N/G ratio (where N is the nuclear component and G is the extended [i.e., galaxy] component). For large N/G, the error on the extended part will be large, and the subtraction tricky. For small N/G, it is much more robust. The accuracy of the decomposition also depends on the angular extend of the galaxy. Typically, using simulated images (with perfectly repeatable PSF's in the same frame), a nuclear component 3 times brighter than the host galaxy with a redshift of about 0.3

with an ordinary seeing of 1.″3 can be decomposed to an accuracy of better than 10%. The user is urged to experiment with simulations in order to obtain a better understanding of the uncertainties of the process.

Another problem with the decomposition is the variation in the shape of the PSF as a function of position and intensity. Unlike DAOPHOT where one finds the scaled PSF best fitted to another star, in PSF decomposition, the matching of the PSF with the point source is extremely important. Sadly to say, so far, I have not seen a CCD chip/camera combination which produces absolutely uniform PSFs throughout the whole image. (This can be seen from residual images from DAOPHOT analysis – often one sees donuts and other imperfections in the residual which may be comparable in flux to the extended emission from a bright quasar!) In general, a thick chip produces a more stable PSF shape across the chip. Another important consideration is the intensity – a very slight non-linearity in the CCD response will give erroneous answer in this algorithm. Choosing a PSF of similar peak intensity to the object in question normally gives the best result. A faulty residual image due to divergent PSF shapes can be recognized normally by its sharp edges, and "non-galaxy"-like characteristics. Again, experiments should be conducted with real data by performing subtractions using several PSFs in the same frame, including subtracting one star from another, to determine the stability and characteristics of the PSF shape.

VII. COSMIC RAY DETECTION REMOVAL (incomplete)

7.1 Overview

This section is slightly out of place in this manual. The cosmic ray (CR) removal algorithms in **ppp** are designed mainly for multi-object spectroscopy data. However, they can be used for direct imaging also, if care is applied so that real objects are not mistaken as CRs. The reader is referred to the section on CR removal in the CNOC redshift survey technique paper, Yee, Ellingson, & Carlberg (1996, ApJS, **102**, 267) for a discussion of the basic method. Removing CRs from a spectroscopic image is inherently complicated because of the different components in such an image: object spectrum, emission lines, sky lines, sky, and totally blank areas. Moreover, the sky can change its color and sky line strengths in a relative short time of an hour or less, further complicating matters.

The most important aspect of decosmic-raying is to make sure that real features are not touched, and CRs are not fixed in such a way that they mimic real features. For this reason, the method I describe will include visual inspection and interactive procedures.

6.2 Decosmic-Raying with Two Frames

Cleaning CRs when there are two frames is relative simple. The algorithm basically compares two frames pixel by pixel. Pixels that are deviant are then deemed CRs and a map of the affected pixels are created. These pixels are then replaced by the corresponding pixels from the other image.

APPENDIX A: INSTALLING ppp

Installing **ppp** on a Solaris machine is very simple as the executable binary is almost entirely self-contained. Here it is step by step:

1. Place the **ppp** executable binary file in the desire directory
2. Change the file to an executable file (since when it is ftp'ed it loses its designation as an executable), e.g., chmod 755 ppp.
3. Place the match.csh and daomaster (files needed for running matching position files) in the same directory (or elsewhere), and remember to chmod for daomaster if necessary.
4. Edit the match.csh file so that it reflects the full pathname of daomaster.

5. If you do not have pgplot, obtain the grfont.dat file for pgplot fonts.
6. Place the following two statements in your .login or .cshrc file so that your computer can find the files:

   setenv PGPLOT_FONT "*pathname*/grfont.dat"

   setenv POS_MATCH "*pathname*/match.csh"
7. Make sure you have SAOimage running in your system.

You are now ready to go.