Overview of the Photometric Pipeline: PSP and Frames

January 30, 1998

Chapter 1

Overview of the Photometric Pipeline: PSP and Frames

This document gives a quick view of the overall operation of the Photometric Pipeline. It should be the equivalent of a short talk, one in which the speaker mentions the structure of a complex problem and briefly touches on a number of issues. It will not describe exactly how the Pipeline operates — for that, you should try looking at the documentation for specific routines or at the source code.

This document was converted from $(PHOTO_DIR)/doc/overview/overview.html using html2latex. It is available as
`http://www.astro.princeton.edu/rhl/overview/overview.html`.

We have a number of technical papers[1] on photo. These should probably be integrated with this document at some point.

This is a first draft, not a finished product. We intend to modify it as desired by readers (especially the JPG) who require more information on specific topics. Please send any comments or suggestions to Zeljko Ivezic[2], Robert Lupton[3], or Jill Knapp[4] at Princeton. With your help, we can improve this document.

- How to Find Things in Photo

[1] See: http://www.astro.princeton.edu/~rhl/photomisc
[2] mailto:ivezic@astro.princeton.edu
[3] mailto:rhl@astro.princeton.edu
[4] mailto:gk@astro.princeton.edu

- The Postage Stamp Pipeline ('PSP')
- The Frames Pipeline ('Frames')
- Photo's Fundamental Datatypes
- Saved Outputs Of Photo's Measure Objects
- Miscellaneous Algorithms
- Known Bugs
- References

1.1 How to Find Things in Photo

This document is not a precis of the photo code; if you want to find out exactly how things are done, you'll want to read the source.

To run photo, you'll first have to acquire a standard SDSS environment, and install all products required to run `astrotools` (or `photo` if you don't want to build it yourself). You can then say `setup photo` and proceed to the next paragraph.

The TCL top-level routines to run the PSP is in

```
$PHOTO_DIR/etc/ps_pipeline.tcl
```

you may want to use cvs to check out your own copy of photo. There's more code in `$PHOTO_DIR/etc/ps_pipeline_procs.tcl` and `$PHOTO_DIR/etc/photo_procs.tcl`.

The frames pipeline routines are in the same place, but called `frames_pipeline.tcl` and `frames_pipeline_procs.tcl`

The main routines are called `run_ps_pipeline` and `run_frames_pipeline`, and each expects one argument, the name of the file that gives the desired order of business, and the location of all files required to run the postage stamp or frames pipeline.

If you are familiar with emacs' TAGS files, you can profitably say `sdssmake tags` in `$PHOTO_DIR` to make browsing the source easier; tags are built for both C and TCL symbols.

Chapter 2

The Postage Stamp Pipeline

2.1 Table of Contents

2.1.1 Introduction to the Postage Stamp Pipeline

As described in Photo Introduction, Photo operates on a frame by frame basis (the photometric data stream from each CCD in the photometric array is cut into frames of 2048 x 1362 pixels). However, the processing needs information for the entire run: quantities such as the point spread function, the sky brightness, and the flat field and bias vectors are not local quantities, but are determined as a function of time, smoothed where appropriate, and interpolated to each frame. This part of the data analysis is performed by the Postage Stamp Pipeline (PSP) whose tasks include:

- calculation of the bias, sky and flat field vectors for each frame
- analysis of postage stamps cut out by Serial Stamp Collecting Pipeline (SSC), and characterization of the point spread function for each frame
- calculation of preliminary astrometric and photometric solutions for each frame from the data provided by the astrometric and monitor telescope pipelines

Input to PSP is provided through a set a files which contain the description of each CCD (information on bad columns etc.), bias vector and data quartiles produced by the DA system, and postage stamps cut out by SSC[1]. Performance of PSP algorithms is controlled by the processing plan and tunable parameters files. Outputs from PSP include bias and flatfield vectors for each frame, as well as bias drift values, sky level, and PSF characterization, and are written to a directory specified by user in psPlan.par file. In addition, user can choose to produce several PostScript plots which can be used as quick QA tests.

2.1.2 Postage Stamp Pipeline Inputs

The relevant inputs to the PSP are:

- Processing plan and tunable parameters
- Information on CCD chips
- Bias vector
- scFang* and other files from SSC
- Photometric and astrometric calibration files

PSP finds the location of all input files by reading psPlan.par file. A detailed description of input files is given as a part of

Survey Interface File Formats[2] Web page. These files include (an * stands for a string describing run, filter, camera row and frame sequence number):

- **Processing plan and tunable parameters**

[1]See: ssc.html
[2]See: http://www-sdss.fnal.gov:8000/dm/flatFiles/FILES.html

  - *psPlan.par*: Processing plan to control the postage stamp pipeline. Each plan targets the data for one imaging run and from one column in the imaging array.
  - *psParam.par*: Tunable software parameters for the postage stamp pipeline.

- **Information on CCD chips** These files are produced by the DA system, and are changed seldom. Each observing report is accompanied by each of these files, specifying which CCD configurations, calibrations and bad pixels apply for that night's observing.

  - *opConfig*.par*: CCD configuration file which specifies the read-out characteristics (number of amplifiers and location of overscan columns) for all chips.
  - *opECalib*.par*: CCD calibration file which specifies the electronic characteristics (read noise, gain, full-well, and bias level).
  - *opBC*.par*: CCD bad pixel file which specifies the location of bad pixels.

- **Bias vector**

  - *idB*.fit*: There is one bias vector per chip determined from a calibration run (a normal drift scan with the "shutter" closed). The bias vector is the median over all rows in the run; i.e., each value in the bias vector is the median value of all pixel values along its associated column. Overscan columns are included, and medianed the same way as the data columns. No other processing is done (produced by *DA*).

- **scFang* and other files from SSC**

  - *scFang*.fit*: These files contain data quartiles and 65x65 stamps cut out by SSC for each frame separately. These are the only files coming from SSC that are necessary to run PSP (if the files listed below are not available, the PSP will make some reasonable assumptions about the behavior of the outer parts of the point spread function).
  - *scWingList*.fit*: A list of 200x200 stamps (bright stars).

- *scWing*.fit*: 200x200 postage stamps (several per camera column and run) used to generate extended PSF wings. PSP can run without 200x200 stamps.
- *scFrameList*.fit*: A list of whole frames, cut out by SSC, which contain at least one very bright star.
- *scFrame*.fit*: One or two raw frames per camera column and run containing at least one very bright star. PSP can run without these "jumbo stamps".

Note that PSP does not yet handle the scFrame* files.

- **Photometric and astrometric calibration files**
  - *exPhotom*.par*: A set of extinction coefficients for a single night produced by MT calibration system (produced by *excal*)
  - *kaCalObj*.fit*: Calibrated objects (positions and magnitudes in each band) detected in the MT pipeline. Each file contains the object list for one MT sequence (produced by *kali*).
  - *asTrans*.fit*: Astrometric transformations for every field in a single imaging run. It transforms frame (row,col) coordinates to great circle (mu,nu) coordinates for a given inclination (produced by *astrom*).
  - *psCT* : Photometric calibrations as a function of time. This file is actually produced by PSP (during an earlier run), and is used when no calibrated MT patches exist as input (produced by *psp*).

There is another input file, so called *cellprofile file* , which is internal to Photo. Its full path has to specified in *psParam.par* file by using keyword `cellprofile_file`. This file contains precalculated PSF models to be used in the PSF characterization, and can be produced by running `sdssmake`.

2.1.3 Postage Stamp Pipeline Outputs

Tha main output from PSP includes the flat-field vector, the point-spread function parameters and preliminary photometric and astrometric calibrations. A detailed description of all output files is provided as a part of

Survey Interface File Formats[3] Web page. The output files are (an * stands for a string describing run, filter, camera row and frame sequence number):

- *psFF*.fit*: Flat-field vector for a single frame (actually, an inverse flat-field vector, meaning that the raw frame must be multiplied, instead of divided by this vector). The values are stored as unsigned integers. The TSHIFT keyword gives the amount by which each value has been multiplied (this scaling allows precise values while maintaining pure integers).
- *psCB*.fit*: Preliminary photometric and astrometric calibrations, sky value and slope, bias drift, point-spread function parameters, gain, and dark variance for all frames in a single camera column of an imaging run.
- *psCT*.fit*: Photometric calibrations as a function of time (counts for a 20th magnitude object, in each band, as a function of time). There is one time-stamped photometric calibration for every calibration patch intersected by the run for which processed MT data exists. The time-stamp indicates the time at which the calibration patch was observed by the imaging camera during this run. This file is sometimes used as an input file to PSP (when no calibrated MT patches exist).

In addition to the above *FITS* files which contain data to be used by downstream pipelines, there are several additional diagnostics files (mostly in PostScript format) which help in assessing the reliability of PSP outputs:

- *psDiag*.par*: A text file containing PSP messages showing the code progress through its all major parts. In its present form does not mean much.
- *psPlots*.ps*: A plot with numerous panels showing the frame dependence of various relevant quantities.
- *psPlotsQU*.ps*: A plot with 2 panels showing the distribution of the parameters describing the size and shape of stars which are used for the PSF determination for this frame. Also shows the rejection boundaries for these parameters.

[3]See: http://www-sdss.fnal.gov:8000/dm/flatFiles/FILES.html

- *psPlotsRejecStats*.ps*: A plot with 2 panels showing the number and percentage of stars rejected for various reasons.
- *psPlotsCellProf*.ps*: A plot with 2 panels showing the radial profiles of all stars, and of stars selected for the PSF determination for this frame.

Production of the above PostScript files is controlled by a set of flags defined in *psPlan.par*.

2.1.4 Postage Stamp Pipeline Algorithms

PSP tasks can be divided into 3 parts:

- Determination of bias, sky and flat field
- Characterization of the point spread function
- Preliminary astrometric and photometric calibration

Each part is independent of the other two, and their flow charts [4] are presented separately. The aim of this document is to list what happens to the data on its way through those 3 PSP segments, and to describe the algorithms for determining the output quantities. However, note that this document does not describe source code which can be studied directly by (per)using C (*.c) and Tcl (*.tcl) files.

Determination of bias, sky and flat field

Within the usual operating range, CCD detectors respond linearly to the incoming light with the response varying from pixel to pixel. Also, some signal is detected even in the absence of incoming light and has to be subtracted from the raw data. In addition to these two corrections, flat-fielding and bias, sky brightness (i.e. the background level) has to be known before the properties of detected objects can be measured.

When the drift scanning technique is employed, pixels along the scanning direction (rows) are integrated together, and all three above quantities depend only (at a given time) on the chip columns. Currently, the bias, flat-field and sky are determined on a frame by frame basis (corresponding to intervals of about 55 sec). Another simplification is that the overall bias shape as a

[4] See: psp_flowcharts.html

function of column is rather stable and it is sufficient to continuosly monitor only its scale shift called the bias drift. For this purpose each chip has an overclock region, a 40 columns wide strip not exposed to the incoming light.

Two kinds of input data are used for the bias, flat-field and sky determination:

- Bias vector
- Data quartiles

The bias vector is determined in a *bias run* (a normal drift scan with the "shutter" closed); each value in the bias vector is the median value of all pixel values along its associated column. This vector provides the overall bias shape as a function of column, and comes from idB*.fit files (one file per chip).

Data quartiles are determined for an *imaging run*. They include 25%, 50% (median), and 75% quartiles of the pixel values in each column, and are evaluated by the DA system on a frame by frame basis. These values are copied by SSC pipeline from the DA output files to scFang*.fit files which are part of the PSP input.

Bias drift for each frame is evaluated as the difference between the mean 50% quartile value in the overclock region (mean over 40 columns) determined for the imaging run, and that determined from the bias vector. Overclock region columns with negative 50% quartile values (DA system's signal that the value is unreliable) are not taken into account. If a chip has two amplifiers, the bias drift is evaluated for each one separately ('left' and 'right' bias drifts).

Flat-field and sky level determination assumes that only a minority of pixels are covered by objects, so that most of them are representative of the sky level. In addition, it is assumed that the sky level does not vary significantly across the frame (about 14 arcmin) and during the exposure time (about 55 sec). With these assumptions, the flat-field correction (vector) for a given frame is determined from the variation of median pixel values for each column. The mean value of these medians is taken to be the sky level for this chip and frame. Sky level error is determined from the mean difference of the 25% and 75% quartiles.

Thus determined sky value, and flat-field and bias vectors have to be further refined. Flat-field and bias vector values are not determined for bad chip columns and those which have negative quartiles. In addition, large

objects comparable in size to the whole frame can affect the pixel values distribution, which might be no longer representative of the sky level. For these reasons, flat-field and bias vectors, and sky values have to be smoothed and interpolated over a sequence of frames. Details of this procedure and parameters from psParam.par which control its performance are described elsewhere[5].

This part of PSP also calculates the dark variance from read noise and gain values, and the mean gain in case of 2 amplifiers.

Characterization of the point spread function

The star/galaxy separation (and object recognition in general) cannot be done without the detailed knowledge of the point spread function (a finite size image of a point source due to the atmospheric turbulence, telescope optics, and pixel response effects). The point spread function (PSF) in general varies with time and is currently determined on a frame by frame basis (it is assumed that the PSF does not vary accros a frame).

There are two PSF descriptions used in Photo. The first one is used for objects classification in Frames and utilizes a pre-calculated catalog of models based on a sum of circularly symmetric Gaussians. In this representation, the PSF is described by the relative weights of these components, and indices indicating their widths, in a model which provides the best-fit to a profile characteristic of the PSF. The fitting is done on a cell by cell basis, where a cell is a union of pixels which traces both the angular and radial structure of an object's image (an object is divided into 12 angular sectors, and each sector is further divided radially into cells, with the number of pixels in each cell increasing with the radial distance; for details see the document SDSS Profile Extraction). The second representation describes the PSF in terms of best-fit model parameters determined by fitting a sum of two Gaussians and a power-law wing to the PSF profile, and is used for subtraction of bright objects (this representation handles much larger dynamic range than the first one). More details about PSF representations can be found elsewhere[6].

As it can happen that some frames do not have enough bright stars for the PSF determination (especially in u band), several frames are combined together. The number of frames to be considered is a free parameter specified,

[5]See: ff_interp.ps
[6]See: psf_types.ps

as well as all other free input parameters which control the PSF determination, in file psParam.par.

Images of objects whose profiles are used for the PSF determination are supplied by the SSC pipeline in input files as three types of stamps. 65x65 pixel stamps are the basic type which is necessary to run the PSP. They contain moderately bright objects, usually from several to about a hundred per frame. Bright objects needed to characterize the outer part of the PSF (i.e. the power-law wing) are supplied as 200x200 pixel stamps, usually several per camera column and run. Images of extremely bright objects are supplied as whole frames. The PSP can run without the latter two types of stamps, in which case it assumes that the index of the power-law wing is -3 (after Racine 1996), and fits only its scale.

The most complex task in determining the PSF is to select objects whose profile is representative of the PSF, from all objects provided in stamps. In essence, the algorithm is based on successive clippings of 'odd' profiles until a uniform sample is obtained. Such clippings can be divided into 2 groups:

- rejections based on the properties of an individual profile
- rejections based on the statistical properties of the whole sample

¡U¿Individual rejection¡/U¿

As the objects are extracted from the postage stamps (i.e. counts for each cell are determined), a set of flags is set, each indicating that something is not right with that particular object. Objects that have any of the following flags set are excluded from the subsequent consideration for the PSF determination:

- OFFCHIP: part of object is off the chip
- SATURATED: object contains saturated pixels
- NOPROFILE: object is too small (faint) to estimate a profile
- NOAPMAG: profile is too small (to calculate aperture magnitude)
- BADWIDTH: a fit of 2 Gaussians results in too much power in the wide Gaussian
- BADCENTER: the object's center cannot be found
- BADPSF: cannot fit a Gaussian to the profile
- NOSTOKES: Stokes parameters cannot be calculated

- ASYMMETRIC: found to be asymmetric on rotation
- MULTIPLE: object is multiple (second brightest peak found is too strong)
- BADSKY: bad sky level (e.g. the wings of a bright star)

While the remaining objects have well determined profiles/properties, not all of them are representative of the PSF for a given frame.

¡U¿Statistical rejection¡/U¿

From the pool of objects which survived the initial rejection, a smaller sample is defined on a frame by frame basis. It contains all the objects recorded on a number of frames preceeding and succeeding the current frame. For such a sample a list of statistical clippings is performed. Performance and turning on/off each of these steps is controlled by parameters specified in psParam.par. Here they have the same names as in that file and are printed in `typewriter` font. The clipping algorithms include and are executed in the following order:

- deviation from a median in each cell: for each cell, objects that deviate more than `ncellclip`*sigma, where sigma is their quoted error for that cell, are marked. All objects that are marked in this way in at least `nbadcell` cells are rejected. In addition, objects that deviate more than `maxdev`*sigma in **any** cell are rejected, too. The purpose of this step is to reject obviously deviant objects.
- shape of the extracted profile: shape is described by the Stokes parameters Q and U, whose values range from 0 to 1 (0,0 is a circularly symmetric profile). First the median position in the Q-U plane is found, and then the distribution of distances from that position is clipped above `psf_nsigma2`*sigma, where sigma is the distribution width. Note that this step does not assume circularly symmetric PSF (although the model forms used to describe PSF are circularly symmetric) to allow for errors in tracking and temporal changes of the PSF.
- size of the extracted profile: the distributions of the model parameters for a 2 Gaussian best-fit model, sigma1, b (relative strength of the second component), and sigma2 are clipped above and below `psf_nsigma1`*sigma, where sigma is a distribution width.
- clipping on best-fit model PSF: a pre-calculated model is fitted to each object and the most populated model is found. All objects whose best-

fit model differs from that one are rejected. This step sometimes rejects too many stars and is currently turned off.

- clipping until the sample is uniform: for the remaining objects the mean profile is found, and then for each object its mean deviation (in the units of quoted error) from the mean profile is found. All object whose deviation is larger than `chisq_cutoff` are rejected. If the mean deviation of the remaining objects is larger than `chisq_max`, object with the largest deviation is also rejected. This process is repeated until none of the objects deviates from the mean more than `chisq_cutoff`, and until the average deviation is less than `chisq_max` (presumably `chisq_max` < `chisq_cutoff`), or until only `min_starnum` objects are left.

Note that objects which are rejected for some frames can be used on other frames.

Once a sample of 'good' objects for a given frame is formed, the best-fit pre-calculated model is determined from their mean profile. This profile is also used to calculate several photometric corrections (e.g. aperture correction).

The same sample of stars is used to find a composite profile[7]. The composite profile is formed from a sample of stars which greatly vary is brightness, and has increased dynamic range. If 'wing' and 'frame' stars are available, their profiles are added to the composite profile, and a sum of 2 Gaussians and a power-law wing is fitted to it. In the case that a composite profile does not extend far enough (i.e. 'wing' and 'frame' stars are not bright enough), it is assumed that the index of power-law wing is -3 (after Racine 1996), and only its scale is fitted. In extreme cases (corresponding to the ratio of the intensity of the last point in the profile to the central intensity greater than 0.001) the power-law wing is ignored completely, and only 2 Gaussians are fitted to the composite profile.

Preliminary astrometric and photometric calibration

Astrometric part is trivial. Six coefficients which transform frame (row, col) coordinates to great circle (mu,nu) coordinates for a given inclination are read from asTrans*.fit file and simply copied to psCB*.fit file. Note that

[7]See: http://www.astro.princeton.edu/~rhl/photomisc/composite.ps

these transformations are given for the photometric CCDs only (not the astrometric CCDs).

Photometric calibration is somewhat more involved. Its task is to determine, for each frame and band, flux20, the number of counts corresponding to a 20th magnitude object at zero airmass whose color index is `refcolor` (`refcolor` is currently hard-coded as 0). This is done by matching the postage stamps (after they passed individual rejection) to a list of objects provided by the MT calibration system (which are observed nearly simultaneously with the main survey, more details can be found in chapter Photometric Calibration and the Monitor Telescope[8] of the SDSS NASA Proposal, aka Black Book[9]). Postage stamps provide counts, and the MT list provides magnitudes and colors for the objects used in calibration. Assuming that log(flux20) varies linearly with the color index

`log(flux20) = log(flux) + 0.4*(m-20) - b*color + b*refcolor - k*z,`

where

- flux: aperture counts for an object from postage stamps.
- m, color: magnitude and color index for this object supplied by MT in kaCalObj*.fit files.
- refcolor: reference color for which flux20 is evaluated, set to 0.
- b: color term supplied by MT in exPhotom*.fit files. b will be determined for each color during the test year and, presumably, will be checked and updated as needed as part of the post-Photo calibration.
- k: extinction coefficients supplied by MT in exPhotom*.fit files.
- z: airmass, calculated by using information from scFang*.fit files headers

The first three terms in the above equation are evaluated as a weighted mean for a list of matched objects, with the weights equal to quoted inverse variances.

---

[8]See: http://www.astro.princeton.edu/BBOOK/PHOTCAL/photcal.html
[9]See: http://www.astro.princeton.edu/BBOOK

2.1.5 Running the Postage Stamp Pipeline

This document assumes that Photo and all other required SDSS products have been properly installed. For details of this procedure please refer to other SDSS documentation[10]. The aim of this document is to lead a user step-by-step through an execution with a lot of diagnostic output.

Assuming that Photo's home directory is $PHOTO, Photo is executed by

`$PHOTO/bin/photo`

which produces the following response ('¿' is the Unix shell prompt):

```
>$PHOTO/bin/photo
Executing commands in /u/products/IRIX/astrotools/v3_0/etc/astrotoolsStartup.tcl
Executing commands in /u/products/IRIX/dervish/v6_6/etc/dervishStartup.tcl:
Executing commands in /u/ivezic/photo/startup.shiva:
Executing commands in /u/ivezic/photo/startup.photo:
Executing commands in /u/ivezic/photo/startup.ZI:
photo>
```

In the case of using 'display' output (SAO Image),

`photo> set_mtv`

should be used for launching an SAO Image Display window.

Now, one can either change directories to a directory containing the psPlan.par file (e.g. $PHOTODATA_DIR), and then run the pipeline:

```
photo> cd $PHOTODATA_DIR
photo> run_psp_pipeline
```

or, supply the name of the file (including a relative or absolute path) as an argument to the pipeline script:

```
photo> run_psp_pipeline $PHOTODATA_DIR/psPlan.par
```

The following example is based on observations of the Hubble Deep Field by using the 3.5 m Apache Point Observatory reflector and SPICAM camera. All `display_*` keywords in *psPlan.par* file have been set to 1 (except `display_psf=2` and `display_flat=2`), and `verbose` to 0. For `verbose > 0` more text output is produced, with the text amount proportional to the selected value of `verbose` (on a scale 1-10). All plots described below are displayed on a monitor. However, a few of them can be also obtained in

[10]See: http://www-sdss.fnal.gov:8000

PostScript version as a part of the PSP output .

The following plots are shown (click on the name for an example):

- Flat_field, bias and sky:
  - (scaled) median quartile image:[11] Ncolumns x Nframes image of medians for each column (horizontal axis) and frame (vertical axis). Includes overscan columns.
  - 2D flat region:[12] Ncolumns x Nframes image of flat-field values, before smoothing and interpolation, for each column (horizontal axis) and frame (vertical axis). It does not include overscan columns.
  - bias[13] as a function of column (includes overscan columns).
  - sky[14] as a function of frame (before and after interpolation).
  - left bias drift[15] as a function of frame.
  - right bias drift[16] as a function of frame (plotted only when there is more than one amplifier per chip).
  - smoothed flat vector:[17] flat-field values as a function column for each frame after smoothing and interpolation.
- PSF determination:
  - raw postage stamps[18] for each frame and filter
  - corrected stamps which passed individual rejection. Only the data inside the red circles is considered in the PSF determination. The circle radius is equal to `psf_def_rad`, defined in *psParam.par* file. The lower left corner of each stamp contains two numbers: the first number is the stamp's index in the list of all stamps, and the second one is the frame number from which that stamp was taken.

[11] See: psp_fig1.gif
[12] See: psp_fig2.gif
[13] See: psp_fig3.gif
[14] See: psp_fig4.gif
[15] See: psp_fig5.gif
[16] See: psp_fig5.gif
[17] See: psp_fig6.gif
[18] See: psp_fig7.gif

– stamps rejected for the PSF determination[19]. The lower left corner of each stamp contains two numbers: the first number is the stamp's index in the list of all stamps, and the second one is a hexadecimal code explaining why was that stamp rejected (it shows which of the flags used for individual rejection are set). Decoding can be done by using Photo command print_flags_info. For example, `print_flags_info 0x802` produces: `The following flags are set: BADSKY, CR`. Note that stamps are never rejected because of CR (cosmic ray) flag alone.

– stars + asymmetry[20] For each rejected star (provided that it has determined profile and is not saturated) two panels are shown: the left one is the original stamp, and the right one is the residual left after the image rotation for 180 degrees and subtracting it from itself. The four numbers are the stamp's index in the list of all stamps, row and column offset of its peak from the pixel center, and the rejection flag (the same one as in the preceeding figure).

– average PSF[21] obtained by adding all selected stamps for a frame indicated in the lower left corner.

– stamps used for PSF determination[22]. Accepted stamps are framed in yellow, while the clipped stamps are framed in red octagons. The three numbers in the lower left corner are the stamp's index in the list of all stamps, the frame number from which that stamp was taken, and the clipping rejection code. Accepted stamps have the rejection code set to 0, while the rejected stamps can be clipped because of: 1 - sigma1 from a double Gaussian best fit, 2 - sigma2 from a double Gaussian best fit, 4 - b (the amplitude ratio) from a double Gaussian best fit (these flags are cummulative in sense that they can produce any value between 1 and 7); 8 - clipping on shape (the Stokes parameters Q and U); 9 - clipping on the deviation from a median in each cell; 10 - clipping on best-fit model PSF (not belonging to the best-fit model), 11 - clipping on best-fit model PSF (the best-fit model produces large chi2); 16+i - rejected in the i-th iteration of chi2 clipping because of large chi2.

[19]See: psp_fig9.gif
[20]See: psp_fig10.gif
[21]See: psp_fig11.gif
[22]See: psp_fig12.gif

- shape and size rejection statistics[23]. The left panel shows the distribution of Q and U values (points) and the rejection boundaries (white circle). The data corresponding to the rejected stamps are shown as red circles. The right panel shows an analogous plot for b and sigma2 values from a double Gaussian best fit.
- radial profiles[24] for all stamps (left panel) and only for the selected stamps (right panel). The white/red points are data for each stamp, and the red/green lines are the mean profile corresponding to that frame. The blue lines show the mean profile +- 1 sigma
- rejection statistics[25]. For each frame, the left panel shows the numbers of stamps, good stars, those rejected on cell clipping and those rejected on chi2 clipping. The right panel shows the latter three quantities as the percentages of the numbers of stamps.

The main diagnostics plot[26]: there are 12 panels which show the behavior of various relevant quantities on a frame by frame basis. The name of each plotted quantity is given on top of the corresponding panel, as well as the units (if applicable). These quantities include (moving row by row from left to right, starting in the upper left corner and ending in the lower right corner):

- the number of counts corresponding to a 20th magnitude object at zero airmass, for each patch, as a function of time when the patch was observed. A message "No mtPatch available" on top of the panel is printed when there are no available patches.
- the sky background level (in counts).
- left and right bias drift (they are the same if there is only one amplifier per chip)
- the number of selected PSF stars per frame (points are connected by line), and the number of points in the resulting PSF profile.
- the widths of 2 Gaussian components in a best-fit model PSF: triangles - sigma1, dots (sigma2). The solid line shows the width for a single Gaussian fit.

[23]See: psp_fig13.gif
[24]See: psp_fig14.gif
[25]See: psp_fig15.gif
[26]See: psp_fig16.gif

- the amplitude ratio of 2 Gaussian components in a best-fit model PSF.
- the strength of the power-law wing part in a best-fit model PSF, scaled by the amplitude of the first Gaussian component, and multiplied by 1000.
- the index of the power-law wing part in a best-fit model PSF.
- the aperture correction defined as the ratio of counts in an aperture whose size is given by `star1_aperature` specified in *psParam.par* file, and counts in the adopted PSF profile.
- the third and fifth radial values of adopted PSF profile, the latter multiplied by 100.
- mean cell deviation from the adopted PSF profile, evaluated and averaged for each selected star.
- the number of counts corresponding to a 20th magnitude object at zero airmass, for each frame.

Top of this file
Back to Photo Overview

Please send any comments or suggestions to Zeljko Ivezic[27]

ZI, December 1997

---

[27] mailto:ivezic@astro.princeton.edu

Chapter 3

The Postage Stamp Pipeline Flow Charts

- Determination of bias, sky and flat field
- Characterization of the point spread function
- Preliminary astrometric and photometric calibration

Determination of bias, sky and flat field

```
Initialise Modules, Read Plan/Parameter Files
Foreach filter {
    Read 2D (Nframes x Ncolumns) bias structure from idB*.fit file
    Read data quartiles for all frames from scFang*.fit files
    Calculate bias drift, flat-field and sky for all frames
    Foreach frame {
        Smooth and interpolate bias drift, flat-field and (sky)
        Write flat-field to file psFF*.fit
    }
    Cleanup
}
```

Characterization of the point spread function

```
Initialise Modules, Read Plan/Parameter Files
Foreach filter {
    Determine mean (over all frames) gains and dark variances
}
Retrieve information on 200x200 stamps
Foreach filter {
    Foreach frame {
        Read the flat-field vector (from a psFF*.fit file)
        Read 65x65 stamps (from a scFang*.fit file)
        Extract profiles for 65x65 stamps
        If available { Read 200x200 stamps
                       Extract profiles for 200x200 stamps
        }
    }
    Reject objects with any of individual flags set on
}
/* these two loops over filters are separated beacuse the above is
coded at Tcl level, and the following one in C */
Foreach filter {
    Foreach frame {
        Form a subsample of objects
        Clip the subsample on the deviation from a median in each
cell
        Clip the subsample on the extracted profile shape
        Clip the subsample on the extracted profile size
        Clip the subsample by using a best-fit model PSF
        Clip the subsample until it is uniform
        Determine the mean PSF profile and the best-fit model PSF
        Determine composite profile
        Add bright objects' profiles to the composite profile
        Calculate photometric corrections (aperture corrrection etc.)
    }
}
Cleanup
```

Preliminary astrometric and photometric calibration

```
Read astrometric transformation coefficients for each frame and filter
  (from asTrans*.fit file)
Calculate time and airmass for each frame and filter
  (by using information from scFang*.fit files headers)
Read zero points, extinction coefficients and color terms for each
filter
  (from exPhotom*.par file)
Read positions and magnitudes (in each filter) of objects detected
in the MT pipeline
  (from kaCalObj*.fit files, aka patch files)
Read color terms for each filter from psParam.par file
  (to be used in log(flux20) = a + color_term * color_index)
Read counts in each filter for a 20th mag object
  (from an old calibration file, psCT*.fit)
Setup extinction arrays for interpolation
  (extinction coefficient as a function of time and color)
Foreach frame {
    If No MT patches
        Use old calibration(time) from psCT*.fit
    Else {
        Foreach patch file {
            Match PS stars with MT stars
            Determine calibration(time) for this patch file
        }
    }
    Interpolate calibration(time) to each frame
}
```

Top of this file
Back to PSP
Back to Photo Overview

Please send any comments or suggestions to Zeljko Ivezic[1]

ZI, December 1997

[1] mailto:ivezic@astro.princeton.edu

Chapter 4

Overview of the Frames Pipeline

4.1 Table of Contents

4.2 Flow Through the Frames Pipeline

The order in which the modules are executed in the Frames Pipeline is:

`Start`

```
Initialise Modules, Read Plan/Parameter Files
Foreach frame {
    Read Known Objects
    Foreach color {
        Read Frame, and add on overlap with next frame
        Correct Frame
        Fix Cosmic Rays
    }
    Foreach color {
        Find Bright Objects
    }
    Merge Bright_Objects
    Foreach Bright_Object {
        Measure Object
    }
    Subtract Star Wings
    Foreach binning-scale {
        Bin Frame
        Foreach color {
            Estimate and Subtract Local Sky
            Find Faint Objects
        }
       Merge Faint Objects, adding them to object list
    }
    Write Corrected Frame, Mask, and Binned Image
    foreach object {
        Measure Object
        Write parameters and atlas image
        Deblend
        Foreach child {
            Measure Object
            Write parameters and atlas image
        }
    }
    Write remaining outputs
}
Cleanup
End
```

4.3 Input to the Frames Pipeline

In the list below, we give representative values for several parameters (for example, overlap region size) that are not fixed and may change in practice; they are indicated by an *italic* font.

Before it can run, the Frames Pipeline requires various inputs; for details see Correct Frame's discussion of inputs. For each frame, it needs:

- CCD hardware specs for each chip, including its readout noise, gain, list of any defects, etc.
- raw frames for each color, which are a different size than corrected frames: they lack overlap regions between frames, and hence have a smaller number of rows (1354 vs. *1489*; 10%); and they have extra columns from the overclock region on both ends (*2128* vs. 2048).
- a bias vector appropriate for each chip.
- a flatfield vector for each chip.
- a CALIB1 structure for each frame, which includes the filter name, sky value and slope, bias values for each amplifier, approximate flux calibration (in the form of "the number of DN which equal magnitude *20.0*"), a model for the point-spread-function, and the coordinate transformation to convert pixel coordinates in this frame to great-circle coordinates (frames uses these to find the transformation from one band to another, so *which* great circle coordinates is not important).

The first item on this list must be kept up-to-date by the Survey's Imaging Scientist (or other person who carefully monitors the CCD camera). The second item is supplied directly from the Data Acquisition system. All the remaining items are produced by the Postage-Stamp Pipeline.

Chapter 5

The Correct Frames Module

There are two parts of the correct frames module; the production of flat fields and biases in the postage stamp pipeline, and the correction of raw data in the frames pipeline.

5.1 Correcting Raw Frames

- Inputs
- Algorithms
- Outputs

5.1.1 Frames Pipeline Inputs

The inputs to the correct frames module are:

- The raw pixel data
- A 1-D bias vector
- A 1-D inverted flat-field vector
- A CCDPAR structure
- The bias drift for both left and right amplifiers

The raw pixel data

The correct frames module can accept two forms of raw data, either complete frames (including overscan region), or subregions (which must *not* have overscan regions attached); the latter are used by the PSP to flat field postage stamps.

A 1-D bias vector

The bias vector must be the full width of the chip with overscan regions removed. It's scaled up by some value, `TSCALE`, that's specified in the header.

A 1-D inverted flat-field vector

The flat vector is the full width of the chip with overscan regions removed. It is scaled as described in the section on PSP algorithms.

A CCDPAR structure

The following elements of the CCDPAR structure are used:

- The location of the overclock regions
- How the data is split between amplifiers
- The amplifiers' readnoise (in electrons) and gain (electrons/DN)
- The list of bad columns (*double-column defects should be specified as a pair of single-column defects; they are processed correctly despite this requirement*)

The left and right bias drifts

How much the bias level of the right and left amplifiers has drifted relative to the bias vector.

5.1.2 Frames Pipeline Algorithms

First the entire frame is searched for saturated pixels, which are grouped into `OBJMASK`s in the `MASK_SATUR` mask. Because a saturated pixel may have bled into its neighbour above or below it, but the resulting pixel value may be

below 65535, an additional pixel at above and below each saturated column is added to this mask.

Each pixel that is not saturated is then bias subtracted (allowing for bias drifts) and divided by the flat field (or, more precisely, multiplied by the inverse flat field). The soft_bias is then added back to keep all pixel values positive.

Because of the way that the flat field is normalised, a low-sensitivity pixel that isn't saturated in the raw frame can have a value too large to fit into 16 bits after flatfielding. If this is the case, the nominal `saturation_level` for the frame is lowered. Unless something nasty has happened (e.g. the bias drift is wildly wrong) the adopted saturation level should still be close to 65000.

The final stage is to interpolate over all bad pixels, including those labelled as saturated. The algorithm used is described elsewhere. In deriving the interpolation coefficients we assume that the signal-to-noise is infinite, and that the PSF is a sigma=1pixel Gaussian (i.e. $\sim$ 1" FWHM). The correction code is machine generated, to it wouldn't be *too* hard to relax these restrictions if it proved necessary.

All interpolated pixels are added to the `MASK_INTERP` mask.

Interpolation is only carried out using good pixels from the same row; it would be better to use a region with larger support, but the book-keeping involved was too frightening to contemplate (remember; we have to handle bleed trails as well as intrinsic bad columns, and they can easily be adjacent or separated by a single column or row).

No provision is made for non-linearity of the CCDs. XXX

5.1.3 Frames Pipeline Outputs

Outputs are:

- The corrected data (which *must* have a mask)
- The corrected data's mask

Furthermore, once the C code to correct the frame has been run, the TCL routine `regStatsFromQuartiles` is used to determine the global sky level (and its standard deviation). *These values should come from PSP, but there seem to be problems. XXX*

Chapter 6

Finding and Removing Cosmic Rays

6.1 Finding Cosmic Rays

We look for cosmic rays using a sequence of tests, each more discriminating (and more time-consuming) than the preceeding. Initially the frame is searched for *pixels* that appear to be part of a cosmic ray event; then all of these contaminated pixels are assembled into cosmic ray events, and the pixels around these are examined for evidence of contamination, using a lower standard of proof.

In the following descriptions, `C` is the candidate cosmic ray pixel, and other pixels are referred to as (e.g.) `NW`.

```
   -1     0    1
 1 NW     N    NE
 0  W     C    E
-1 SW     S    SE
```

Searching for contaminated pixels follows the following sequence:

- If the code is compiled suitably, pixels that are labelled as interpolated, saturated, or not checked are ignored. This is usually not done, as it makes it hard to find CRs next to bad columns and bleed trails (Exercise for the reader: consider two equally bright CR pixels, one above the other, adjacent to a bad column. Why is the CR finder unable to find them?).

- Then compare C with the four mean values of pairs of neighbouring pixels, `(W--E)`, `(N--S)`, `(SW--NE)`, `(NW--SE)`; if `C` exceeds any one of these by more than the `min_sigma*sigma_sky` the pixel is taken to be a cosmic ray candidate.
- Next, see if the data is within the band limit, i.e. we know that no astronomical source can be sharper than the PSF, so the ratio of a pixel's value to that of a pixel a distance `d` away cannot exceed the ratio of the central value of a star to the value of a pixel at the same distance, `d`.

  Allowing for noise, this condition becomes

  `(C - F*N(C) - sky) > F1*PSF(d)*(mean - F*N(mean) - sky)`

  where PSF(d) is the value of the PSF at a distance d, mean is the average of two pixels each a distance d away, and N(p) is p's standard deviation. `F` is an input parameter (called `cond3_fac`), with default value 3.0; `F1` is a fiddle factor for the noise (it's called `cond3_fac2` and has default value 0.6), used to deter photo from labelling the centres of bright stars as cosmic rays.

  This inequality is evaluated for the four pairs N–S (d==1), E–W (d==1), SW–NE (d = sqrt(2)), and NW–SE (d = sqrt(2)), and if any satisfy it, we've got a cosmic ray.

After a contaminated pixel has been detected, its value is temporarily replaced by the lowest of the means determined in the last step listed above, in order to make it easier to find further pixels contaminated by the same cosmic ray.

When the whole frame has been processed, contiguous contaminated pixels are merged into single cosmic rays. If the peak pixel is below some minimum (`cr_min_e`) number of electrons above the sky level, the candidate cosmic ray is rejected, otherwise it's added to the `MASK_CR` mask.

Next, all cosmic ray pixels are removed.

All the pixels adjacent to each cosmic ray are then examined, using the above set of tests, but with the `min_sigma` and `cond3_fac` values relaxed by 50%. Any contaminated pixels are added to their 'parent' cosmic ray.

Finally, if the `keep` flag is set, the original values of all cosmic ray pixels are reinstated; otherwise the `MASK_INTERP` bit is set for all interpolated pixels

6.2 Removing Cosmic Rays

Once all contaminated pixels have been found (as described above), they are interpolated over using a four-point interpolation scheme as described elsewhere (e.g. horizontally we calculate

```
-0.2737*I[-2] + 0.7737*I[-1] + 0.7737*I[1] - 0.2737*I[2]
```

as a possible replacement for `I[0]`). We interpolate in all four directions (e.g. SW–NE, W–E) and choose the minimum value, after rejecting any involving bad (`MASK_INTERP` or `MASK_NOTCHECKED`) pixels. If none of the interpolated values is acceptable, the pixel is replaced by the sky value.

Because we have chosen the minimum of a number of estimates, the resulting value is biased a little low. We correct for this by assuming that there were only two uncontaminated values, and thus add 0.56*(`sky sigma`).

6.3 Restrictions on CR code

- Cosmic ray contaminated pixels in the first and last row or column are more likely to be missed.
- No more than 10000 cosmic ray contaminated pixels can be found in any one frame.

Chapter 7

Bright Objects

7.1 Find Bright Objects

There is no fundamental difference between the Find Bright and Find Faint Objects modules; indeed the same object finder is used by both. The only difference is that Find Bright Objects:

- Works on an *un*-smoothed image, with the single global sky value determined by correct frames.
- Uses a brighter threshold, namely `fbo_thresholds*(sky sigma)` where `sky sigma` is again determined by correct frames.
- Runs before we've made a local determination of the sky level. This means that we haven't subtracted (much) power from the wings of extended objects.
- Runs before the wings of bright stars have been subtracted

Once the bright objects have been found, the list of peaks in each object is culled, removing all peaks that are less significant than 3 `sigma` (`sigma`, the pixel standard deviation, is determined locally).

7.2 Subtract Star Wings

Here are a few notes for when I write this section:

- Only saturated objects are currently subtracted

- Objects with centres off the frame are *not* handled
- We should discuss how to find centre/psfCounts for saturated objects
- Ghost removal belongs here

Chapter 8

Faint Objects

8.1 Estimate and Subtract Local Sky

- Spatially median smooth the sky, clipping at 2.3sigma. This is done in NxN boxes, centred every (N/2,N/2) (parameter `ffo_median_size`).
- Interpolate between the sky points (using a variant of Bresenham's algorithm), subtract from the frame, dither, and add back the constant 'software bias'.

8.2 Find Faint Objects

The steps taken are:

- Convolve with a Gaussian with FWHM equal to the narrower of the components in the PSF. The filter size is `ffo_psf_filt_size*ffo_psf_filt_size`.

  A 'fringe' of pixels `ffo_psf_filt_size/2` is left unsmoothed around the edge of the frame; these are marked `NOTCHECKED`.
- Find all connected pixels above a set of thresholds (`ffo_thresholds`) in the smoothed frame (currently only the lowest threshold is used). Thresholds are in units of the smoothed-sky sigma, which is estimated from the interquartile range.
- Grow the objects by 6 (XXX) pixels in each direction. This is required as we detect objects based on pixels above a threshold, but we know that objects are in fact larger. Specifically, if an object is detected in

only one pixel we *know* that it in fact covers at least an entire seeing disk, and this stage of growing detected objects allows for this.

Once the objects have been found, the list of peaks in each object is culled, removing all peaks that are less significant than 3 `sigma` (`sigma`, the pixel standard deviation, is determined locally).

8.3 Bin Frame

Bin the frame by some power of two in both row and column direction. The origin of the binned frame in each band is chosen taking into account chip-to-chip offsets, so that the binned pixels in the different bands correspond to the same portion of the sky. The data is scaled up by the square root of the area of the binned pixel, so that no significant information is lost.

Binning is generally applied to data which has had an object finder run on it, and in which all detected objects have been replaced by the background level.

An object detected in a binned frame has the `OBJECT1_BINNED` flag set; it is not currently possible to know on what scale the field was binned.

Chapter 9

The Merge Colors Module

Merge Colors combines `OBJECT1`s (detections of objects in a single passband) into `OBJC`s, which contain information from all colors.

The basic idea is to align the different colors with each other properly, then consider the OBJECT1s from all colors simultaneously; any objects which overlap – contain the same area on the sky – are combined into a single OBJC.

We create an OBJC for each OBJECT1 in the first color, and create a `master_mask` for the new OBJC. This mask is marked to indicate all the pixels which belong to the OBJC. Then, we consider each of the other colors in turn. For each OBJECT1, we check to see if it overlaps any of the existing master masks; if so, we then *grow* the master mask so that it contains the union of itself and the OBJECT1.

After iterating through all the colors, we are left with a set of OBJCs whose master masks indicate their total extent. Each OBJECT1 has its centre set to the brightest `PEAK` in any of the OBJECT1s that were merged to create it, and the centre of the OBJC is set (in the canonical band's coordinate system) to the centre of the whichever of the merged OBJCs had the highest peak in the canonical band.

After merging, each OBJECT1 carries with it all the peaks detected in any of its constituents, with the proviso that peaks that are too close to each other are merged together. Furthermore, the OBJC is given a list of all peaks in *any* of its OBJECT1s, properly transformed to the canonical frame and tagged with the band in which they were detected.

Chapter 10

The Measure Objects Module

10.1 Measure Object Properties

The Measure Objects module calculates parameters for objects, and classifies them based on their values.

Each OBJC is passed to the Measure Objects module individually, so all passbands can be measured simultaneously.

Actually measuring the object's properties is simple enough to describe, although some of the algorithms are rather complex.

The first thing to check is if the object's had its wings subtracted; if so we assert that it's already been measured (presumably as a bright object) and return. *We should not remeasure* any *bright objects, as they have had power subtracted from their wings*

Then we find a good centre for each band, a step that's only important if the object being measured is the product of the deblender. Then a *canonical centre* is found. If the object is detected in the band denoted `mo_fiber_color` in the frames parameter file (usually r'), the centre in that band is the canonical centre. If it *isn't* detected in that band, the band with the largest peak counts is used. It would be better to use e.g. a fibre magnitude, but one is not available at this point.

Once the canonical centre's in hand, we process the object in that band, followed by the other bands in no particular order. For each band, we:

- Transform the canonical centre to that band's coordinate system
- Interpolate the sky and sky error to the centre of the object

- Extract a radial profile about the canonical centre; see SDSS Profile Extraction[1] and Aperture Photometry in Band Limited Data[2] for details. The radial profile is constructed from the values in cells by using a slightly clipped mean of the mean/median values; the errors are calculated using a local fit around the annulus as descibed in the referenced documents. (*the clipping is currently set to 0. XXX*).

  The object can be too large for the method we use, in which case the `OBJECT1_TOO_LARGE` bit is set (the condition is that no cell may contain more than 65535 pixels, giving an outer radius of 657 pixels); if the extracted profile reaches the edge of the frame in any sector, `OBJECT1_EDGE` is set.

  During the extraction, if the sky level were sufficiently badly estimated, the central intensity of the object could be negative. If this is so, set the `OBJECT1_BADSKY` and `OBJECT1_NOPETRO` bits, and give up on this band.

- Calculate the fibre magnitude, centred on the canonical position. This is done using the same technique as the inner parts of the radial profile. *This magnitude is supposed to be corrected to canonical seeing, but this is not yet done.*

- Calculate the PSF magnitude, and apply an aperture correction[3] calculated by the PSP. You don't usually need to use an aperture correction with PSF magnitudes — it all comes out in the wash when analysing the standards — but drift scan data is a little different[4].

  This is done as a sum over the cells in the radial profile, rather than as an integral over image. This sacrifices a little statistical efficiency, but makes the PSP's task of estimating the aperture corrections easier (more precisely, it greatly eases the PSP's memory requirements).

- Next comes the calculation of Petrosian quantities. The basic equations are given elsewhere.

  The procedure is:

[1]See: http://www.astro.princeton.edu/~rhl/photomisc/profiles.ps
[2]See: http://www.astro.princeton.edu/~rhl/photomisc/aperture.ps
[3]See: XXX
[4]See: http://www.astro.princeton.edu/~rhl/photomisc/calib.ps

- If there's only one measured point in the profile, set OBJECT1_NOPROFILE and give up
- Calculate a cumulative profile from the radial profile extracted above. If desired (which, by default, it is not) the resulting profile can be spline-smoothed, allowing for the errors in the points. This cumulative profile is then fitted by a taut spline. To reduce the profile's dynamic range, what is actually splined is the asinh of the radius against the asinh of the cumulative brightness (a cumulative light profile is, of course, employed so that these radii are well defined).
- Convert this splined cumulative profile to a surface brightness profile by differentiation (not too scary, as we are differentiating a function we just integrated so the noise is OK).
- Find the Petrosian ratio, and solve for the Petrosian radius. Reject any of these radii that are at too low a surface brightness (setting the OBJECT1_PETROFAINT bit), and keep the *largest* of the remainder. If there are none, set OBJECT1_NOPETRO; if there are many, set OBJECT1_MANYPETRO.
- If we found a Petrosian radius, estimate the error in the Petrosian radius, as discussed in Errors in Petrosian Quantities[5]. If we failed, set OBJECT1_NOPETRO_SMALL and set the Petrosian radius to it's fallback value.
- Now find the flux within some multiple of the *canonical* Petrosian radius; note that we processed the canonical band first, so this is known at this point in the processing.
- Calculate the radii containing 50% and 90% of this Petrosian flux, setting OBJECT1_MANYPETRO50 and OBJECT1_MANYPETRO90 as appropriate.
- Finally, if the edge of the frame lies within one Petrosian radius of the object's centre, set the OBJECT1_INCOMPLETE_PROFILE bit.

- If OBJECT1_NOPROFILE isn't set, calculate the objects Q and U parameters; see

  The Estimation of Objects' Ellipticities[6] for details on algorithms. The

[5]See: http://www.astro.princeton.edu/~rhl/photomisc/petrosian.ps
[6]See: http://www.astro.princeton.edu/~rhl/photomisc/ellipticity.ps

integrals over the object are carried out as described in Aperture Photometry in Band Limited Data[7].

- Next, measure the shape of a certain isophote. *The algorithm described here needs some work, but I think that the basic idea is sound.*

  In each sector of the extracted cell profile, estimate the *first* radius at which the desired surface brightness is reached; this is done using the cumulative spline technique described above. If the object's centre is fainter than the desired isophote, set the `OBJECT1_ELLIPFAINT` bit and give up.

  Next fit a five-Fourier series (i.e. up to the cos(2 theta) and sin(2 theta) terms) to these radii, and convert the resulting coefficients into a centre, a major and minor axis length, and a position angle. This is currently done by finding the centre of area of the curve defined by the Fourier series, and calling it the centre. The second moments of the curve about this point are then used to determine the axes and orientation of a best-fit ellipse. Finally, adjust the axes so that the area of the ellipse equals the area of the sectors within the isophote. *It is not clear that this is a satisfactory procedure; TBD*

- The gradient terms are easily found by applying the same procedure at slightly brighter and fainter isophotes; the error terms are TBD.

- Next we fit a PSF, an exponential disk, and a deVaucouleurs model to the cell profiles. Details to be written up when RHL gets a chance. XXX.

- The texture is calculated, but the algorithm is not final. Details to be provided when the code is finished.

- A crude classification is then applied; this is probably not good enough even for level 0. *This is scary. I think that the parameters being measured are good, and suitable for the task of both star-galaxy separation and (maybe) galaxy classification. It'll probably take the application of some brute-force technique such as neural nets, decision trees, or a Bayesian classifier [all of which are equivalent] to decide which combination of parameters is most effective.*

- Finally, some flag bits are copied to the OBJC, namely: `OBJECT1_EDGE`, `OBJECT1_BLENDED`, `OBJECT1_CHILD`, `OBJECT1_NOPETRO`, `OBJECT1_MANYPETRO`,

[7]See: http://www.astro.princeton.edu/~rhl/photomisc/aperture.ps

OBJECT1_INTERP, OBJECT1_CR, OBJECT1_SATUR, OBJECT1_NOTCHECKED, and OBJECT1_BINNED. If the OBJC is blended, the OBJECT1_BLENDED bit is set in the OBJECT1 that we've just processed.

If the object is blended, the deblender is run and this entire procedure is repeated for each child.

10.2 Classification using a Decision Tree

Having finished measurement, we are now ready to classify. Classification is performed using a decision tree, which is stored internally to PHOTO as a TREENODE structure (with branches connecting it to other TREENODEs). Some human must have created a decision tree appropriate for the data, and placed it (currently, in the form of an ASCII file) into the PHOTODATA directory, before the pipeline is run. It will not be possible to modify the tree for each frame, or even for each run — we will probably end up using a single tree for at least an entire season, if not for the entire survey.

As of May, 1995, we have the ability to create decision trees relatively quickly. However, we have *not* spent time to try to create trees which yield high accuracy; that will be done later, when we have real test data.

Chapter 11

Photo's Deblender

- Overview of Deblender
- Photo's Deblending Algorithm
- Book Keeping for the Deblender

11.1 Overview of Deblender

The photo deblender is based on the peaks found in each object. The basic idea is that we take the list of all peaks found within the parent, in any band (a peak need only appear in one band to be considered); each of these peaks is taken to define the centre of a possible child.

For each child, in each band we create a *template*, an approximate light distribution for the child (for a star, this template would be simply the PSF). We then assume that in each band

```
parent = sum_i (weight_i template_i)
```

and solve for the weights. With these in hand we can then assign the flux associated with each child in each pixel according to the value of `weight_i template_i`, and our task is complete.

Note that, because the children are associated with the master list of peaks detected in *any* band, the resulting children are defined consistently in each band so it makes sense to (e.g.) ask questions about their colours.

11.2 Photo's Deblending Algorithm

Photo's deblender works by looking at the list of `PEAK`s in an `OBJC`, and treats each as the centre of a child. This list was produced by Measure Objects. The deblender proceeds as follows:

- When an OBJC is passed to the deblender, the first thing that's checked is if it has too many peaks (set as `child_max` in the *fpParams.par*); if it does the `OBJECT1_NODEBLEND` bit is set, and the deblender gives up.
- Next, each child (i.e. peak) is examined in each band. If, in that band, the peak is consistent with being a PSF the `OBJECT1_DEBLENDED_AS_PSF` bit in `flags` is set, the star is subtracted, and bits are set in an `OBJMASK` that indicate which pixels were modified.
- Then photo goes through each child creating a *template*, that is:
  - If the object is too large for the deblender to handle (basically, more rows or columns than half a frame) the `OBJECT1_TOO_LARGE` bit is set and photo gives up on that child.
  - If the object reaches the edge of the frame in any band, the `OBJECT1_EDGE` bit is set, and photo gives up on that child.
  - If the `OBJECT1_DEBLENDED_AS_PSF` bit is set, the template is taken to be a PSF, centred at the centre of the child. Otherwise, photo goes through the object looking at pairs of pixels `(rowc + y, colc + x)` and `(rowc - y, colc - x)`, where `(rowc, colc)` is the centre of the child. The template is made up of the *smaller* of the two pixel values, corrected for bias by adding `0.56(sky sigma)`.

    For pixels whose 'mirror' lies outside the parent object's OBJMASK, the value of the template is taken to be zero. If one of the pixels lies in an area where a PSF has been subtracted (see above), the other pixel's value is used. If both pixels were PSF contaminated, use the average of the two. In all of these cases, no bias correction is applied.
  - Next, we run an object finder on the smoothed template; the smoothing length and threshold are those used in Find Faint Objects. If the template is not detected at this stage it is presumably because the peak that corresponds to the child being processed was too faint to have been detected as an isolated object;

the OBJECT1_NOTDETECTED bit is set and photo gives up on that OBJECT1.

  - Once we've found a template in each band, we check that it was detected in at least one of them; if not photo gives up on that child.
  - Then minimal templates are created in all bands where we failed to find one; these are simply multiples of the PSF; the OBJECT1_DEBLENDED_AS_PSF bit is set to indicate this.
  - Finally, all parts of templates that do not lie within the OBJECT1s OBJMASK are set to zero.

- With templates available for all children (in all colours), we solve for their weights. This is done using (unweighted) least-squares. The Normal Equations are solved by eigen value decomposition, which allows us to discover if the problem is ill-posed (i.e. the matrix is singular).

  We achieve this by going through all the children in each band looking for the smallest eigenvalue; if it is too small, that child is rejected, and the parent has the OBJECT1_DEBLEND_PRUNED bit set (both for the OBJC and also for all the OBJECT1s). The least squares procedure is then repeated, until a solution is reached. If this solution consists of only a single child, the parent's OBJECT1_BLENDED bit is turned off, and the deblender stops.

- With the weights for each template in hand, and after setting any negative weights to zero, for each pixel in the parent we evaluate `sum == sum_i (weight_i * template_i)`. If this is at least 10, that pixel in the i-th deblended child has value `weight_i * template_i`; otherwise we re-evaluate `sum` using the *smoothed* templates. If the resulting `sum` is greater than 2, the pixels are given values of `weight_i * smoothed_template_i` otherwise we share the flux equally among all the children. XXX

At the end of this procedure, each child knows the values of all its pixels that lie within the OBJMASK defining the parent, and it is ready to be passed to Measure Objects.

11.3 Book Keeping for the Deblender

Once an object is deblended, it is represented as an `OBJC` as usual, but with a non-`NULL` `children` field which points at one of its children; in addition its `OBJECT1_BLENDED` bit is set, and `nchild` is one or more.

Each child has a non-`NULL` `parent` field which points to its parent, and may have a `sibbs` field too. It has its `OBJECT1_CHILD` bit set.

Chapter 12

Miscellaneous Operations on Objects

- Finding Objects
- Rejecting peaks that don't meet certain criteria ('culling')
- Merging Together Peaks Found on Separate PEAK Lists
- Finding a Good Centre for a (possibly extended) object
- Growing an object by adding pixels to its periphery

12.1 Finding Objects

Photo's object finder looks for all connected pixels above a set of thresholds (pixels which touch only at a corner are considered to be connected), resulting in a set of `OBJECT`s. If an estimate of the sky level is provided, also fit the centres of all peaks present within each object (both the primary peak and any lower secondary peaks). These are stored in a data type called a `PEAK`. At this point, the peak's centres are given in their own band's coordinate system.

12.2 Rejecting peaks that don't meet certain criteria ('culling')

Cull some of a list of `PEAK`s given some thresholds. Specifically:

- Delete those with a peak value < threshold.
- Each peak (except the first) has a saddle point joining it to a higher peak; if the value at that saddle point is greater or equal to `(peak - delta)`, cull the peak.

The brightest peak in an object is never deleted.

`delta` may be expressed in terms of the local pixel standard deviation.

12.3 Merging Together Peaks Found on Separate PEAK Lists

There are two distinct ways in which lists of PEAKs are merged, when we are merging `OBJECT1`s in the *same* band, and when we are merging the lists of PEAKs detected in *different* bands.

Various tolerances are used to decide when peaks are identical XXX

12.4 Finding a Good Centre

Photo centres objects by taking the brightest pixel, and fitting a Gaussian[1] of suitable width. If the width of the object (as determined by the fit) is too large in row and/or column, the object is rebinned (in row and/or column) and the fit repeated. When the smoothing scale matches the object, the resulting centre and error is returned. The rebinning can either return the mean or median of the pixels that are to be binned together; currently the mean is used.

For a star this should produce an approximately optimal centre; the code is that used elsewhere in SDSS software. For more extended objects there's no particular reason to expect that a Gaussian should describe the profile well, but it probably doesn't really matter.

[1]See: http://www.astro.princeton.edu/~rhl/photomisc/centroiding.ps

12.5 Growing an object by adding pixels to its periphery

Growing an object involves adding pixels to its outside; we need to do this when we've detected an object above some threshold, but we know that it also must have pixels below that threshold. Let us consider the problem of growing by `N` pixels.

The correct way to do this is to add a single pixel to the end of each row of detected pixels, add an extra row above and below each row, figure out which pixels are now present many times in the object and reduce it to some canonical form, and then repeat the operation `N` times.

Unfortunately, this is expensive and we have instead used a one-step procedure: Add `N` pixels to each end of each row, and `N - 1` to the rows above and below, `N - 2` to the next two, and so on. Finally, deal with overlaps.

Probably we should just switch to the 'right' way of doing it, but this hack works well enough

Chapter 13

Outputs from the Frames Pipeline

13.1 Fieldstat

The *fieldstat* structure holds a short summary of information on a field-by-field basis. The output file has a name like

`fieldstat-run-ccdcol-field.fit`

This is a FITS binary table, containing one row. The row holds a number of columns. The following are contained once:

- number of passbands in this field
- status of processing (e.g. aborted)
- field number
- total number of objects (after merging those found in all passbands)
- total number of stars
- total number of galaxies

and then, for each passband, there are individual entries for:

- saturation_level
- sky value for each frame
- sigpix value for each frame

- number of cosmic rays in each frame
- number of bright objects in each frame
- number of faint objects in each frame
- median colour of objects in frame
- median Stokes' parameters, U and Q, for objects in the frame

A typical size for this file is about 12 kB, mostly due to the inefficiency of the FITS standard.

13.2 Binned Image

The *binned image* file contains a 4x4 binned image of the corrected frame. Its filename is something like

```
BIN-run-ccdcol-filter-field.fit
```

XXX describe binning, and super pixels in atlas images

This is a 16-bit integer FITS image. The "sky" value should be set to the value of the `soft_bias` parameter, which as 1,000 at the time of writing.

A typical size for this file is 380 kB.

13.3 Image Masks

The mask file contains a `SPANMASK` of the corrected frame, showing pixels which belong to objects, pixels which have been saturated, etc. Its filename is something like

```
M-run-ccdcol-filter-field.fit
```

A typical size for this file is ??? MB.

13.4 Corrected Image

The *corrected* image contains the full representation of a corrected frame. Its filename is something like

```
C-run-ccdcol-filter-field.fit
```

This file is a 16-bit integer FITS image.

A typical size for this file is 11.9 MB.

13.5 OBJC binary tables

See the separate section Measured Object Parameters. The rows in this file are identical to those in the atlas image file, and the `id` number of the object is equal to its row number.

Files of this type can range in size quite a lot.

Chapter 14

Photo's Fundamental Datatypes

ATLAS_IMAGE
: The portion of a frame containing the pixels that have been determined to belong to an object. (Pixels that have too small a S/N to be found individually will *not* appear in the ATLAS_IMAGE. On the other hand the individual pixel values are not very useful, and their smoothed properties may be recovered from the 4x4 binned image produced by the frames pipeline.

MASK
: An 8-bit region, with one byte corresponding to each pixel in a region. Photo doesn't use MASKs, preferring to use `SPANMASK`s for reasons given below.

OBJC
: An object in all five filters, containing the `OBJECT1`s from each band.

OBJC_IO
: An `OBJC` reorganised to be written to a fits binary file.

OBJECT
: A collection of pixels above threshold, with associated PEAK information.

OBJECT1
: An object in one filter, complete with information about its parent region, in which pixels it was detected, its PEAKs, and all its measured properties.

OBJMASK
: An 'object' describing which pixels in an image have a certain property. Objmasks are used to describe objects (i.e. in which pixels an object is detected), and also things like cosmic rays (i.e. which pixels a cosmic ray was removed from), and interpolated pixels (all pixels which have been interpolated over). A comparison with dervish's `MASK`s is given below.

PEAK
: A single peak within an object. The list of peaks within an object is crucial to the operation of the deblender, as well as interesting in its own right. Photo takes considerable pains to keep this list correct at all times.

REGION
: Dervish's datatype supporting a rectangular collection of pixels.

SPANMASK
: A collection of `OBJMASK`s, stored as a chain. This is the equivalent of a bitplane in one of dervish's `MASK`s.

14.1 Note on Masks Structures

The standard dervish mask is a bitmask, with 8 bits for each pixel in the image. They have various disadvantages; they take up a lot of memory (a full mask is 3Mby) — not so much a concern per se, but a cache disaster; you cannot find all the saturated pixels (say) without running an object finder on the object mask; you have to check each pixel as you process it to see if it's OK. All of these concerns are addressed by the representation of each bitplane as a set of 'OBJMASK's, which are a structure consisting of {row, col1, col2} for each line-segment where the bitmask would have been set (they also contain a bounding box and other book-keeping information). These

sets, currently CHAINs, are then assembled into 'SPANMASK's, which are arrays of the chains of OBJMASKs, one for each bitplane in the old shiva masks.

Note that as our bitplanes are sparse, much less memory is involved in these data structures; you can ask for a list of all cosmic rays (it's a (possibly empty) chain of OBJMASKs); and you can ask for all the pixels in this object that aren't in the NOTCHECKED chain, and then simply loop over the spans in question, or ask for the pixels in this object that are saturated.

The downside is that a SPANMASK doesn't match a fits binary table very well. An chain of OBJMASKs isn't too bad — one row for each OBJMASK, with the {r, c1, c2} triples in the heap as properly-byte-swapped-shorts — but to put a whole SPANMASK in requires that all of the OBJMASK data appear in the heap, and then there's a nightmare about byte order, as you have to know if the heap data is 1, 2, or 4 byte units.

Chapter 15

Measured Object Parameters

15.1 File Formats

1. Corrected Frames will be written to fits files containing images and masks, where the format of the corrected frames files will be:

   - an HDU (*Header Data Unit*, here basically an empty table) giving e.g. the frame number
   - 5*{
     - An `XTENSION IMAGE` giving the corrected image. *As dervish doesn't (yet?) support IMAGE extensions, the corrected frames are currently written one-per-file*
     - A set of (currently 8) `XTENSION BINTABLE`s, each containing one of the 'bitplanes' (e.g. `MASK_INTERP`); in these the `OBJMASK` triples will be in the heap.

   }
   Note that this hierarchical structure enables us to correctly describe the data in heap, which could not be achieved if we used only a single binary table expressing the full `SPANMASK`.

2. `OBJC`s will be split into three structures for output; the main motivation for this is to write fields in each of the ncolor `OBJECT1`s together (e.g. rowc[5] not 5 rowc columns). The structs are:

   - `OBJC_IO` The main tabular data. This table has no data in heap.

- `ATLAS_IMAGE` The masks and regions making up the atlas images. The structure of this binary table will be complex, but there will be a function to read a given row into a defined structure in memory. We shall provide a stand-alone programme that links this function and breaks atlas image files out into (many) fits binary images (or maybe one with IMAGE extensions; TBD). As of December 1997 this stand-alone programme is not written, although photo is able to (and does) read atlas image files.
- `struct test_info` Anything that the photometric pipeline testers want saved to disk. This product is *not* written out during production runs and it is not a saved product of the survey. An example might be the positions of all the peaks detected in each object in each band.

3. Parameters describing photo's performance are also written out.

15.2 `OBJC_IO`: Photo's Main Tabular Output

First some outputs which describe the object as a whole:

`id`
: An id for the object within the frame; this id is the same as the object's row number in the table, and is used to tie the `OBJC_IO` and `ATLAS_IMAGE` tables together.

`parent`
: The `id` number of the object's parent, or -1 if it is a primary detection.

`ncolor`
: The number of colours present in this table; should always be 5 during normal operations.

`objc_type`
: An enum giving the object's overall classification. Current possibilities are:

UNK
: An object of unknown type (the default).

CR
: A cosmic ray

DEFECT
: Some indeterminate chip defect.

GALAXY
: A galaxy

GHOST
: A ghost produced by the 2.5m optics.

KNOWNOBJ
: A known object (from e.g. FIRST or ROSAT); its position and size are inputs to photo.

STAR
: A Star

TRAIL
: A satellite, aeroplane, meteorite, or asteroid trail.

SKY
: A pseudo-object; a location where no object was detected in any of the survey bands.

These categories are repeated below for each band; the algorithm used to arrive at the overall classigfication is TBD.

`catID`
: A catalog id number associated with KNOWNOBJs. We may want to revisit how this is done when some real known object catalogues are available.

`objc_flags`
: The union of certain of the flag bits set in each individual band (see discussion of `flags` below). Specifically, if the flags BLENDED, CHILD, EDGE, INTERP, MANYPETRO, NOPETRO, NOTCHECKED, or SATUR are set in any band, they are also set in `objc_flags`.

`objc_rowc, objc_colc, objc_rowcErr, objc_colcErr`
: The canonical position of the object (and 1-sigma errors), in the r' coordinate system. If an object is detected in r', this is the r' centre; otherwise it's a suitable average of the bands where the object was detected (Pixels).

The following fields are calculated for each band:

`rowc, colc, rowcErr, colcErr`
: The position of the object (and 1-sigma errors) in each band. In the case that an object is not detected in a given band (say f'), the position is taken to be the r' position offset to the f' coordinate system (if detected in r'), and a suitable average of the other bands otherwise. (Pixels). We will describe the position assigned in the r' as the *r' position*, even if the object was not in fact detected in r'.

`sky, skyErr`
: The sky level (and the 1-sigma), at the position of the object. (Counts/pixel^2).

`psfCounts, psfCountsErr`
: The PSF-flux (and the 1-sigma error), at the position of the object (Counts).

`fiberCounts, fiberCountsErr`
: The 3"-counts (and the 1-sigma error), at the r' position of the object. These counts are supposed to be corrected to a canonical seeing, but this is not currently (Dec 1997) being done.

`petroRad, petroRadErr`
: The Petrosian radius (and the 1-sigma error), measured using the r' position of the object (Pixels).

`petroCounts, petroCountsErr`
: The Petrosian counts (and the 1-sigma error) within f_3 r_{P,r'} of the r' centre. Suitable measures must be taken if the object is not detected in r' (Counts).

`petroR50, petroR50Err`
: The Petrosian 50%-light radius (and 1-sigma error) (Pixels).

`petroR50, petroR90Err`
: The Petrosian 90%-light radius (and 1-sigma error) (Pixels).

`Q, U, QErr, UErr`
: The values of <colˆ2/rˆ2 - rowˆ2/rˆ2> and <col row/rˆ2> (and their 1-sigma errors), measured within r_{P,r'}. These are estimators of (a - b)/(a + b)cos(2 phi) and (a - b)/(a + b)sin(2 phi), and are named by analogy to the usual Stokes parameters. For more details, see The Estimation of Object's Ellipticities[1].

`nprof`
: The number of points in the three succeeding measures of the radial profile, each of which refers to annuli with fixed outer radii of { 0.56, 1.69, 2.58, 4.41, 7.51, 11.58, 18.58, 28.55, 45.50, 70.51, 110.5, 172.5, 269.5, 420.5, 657.5 } pixels, that is { 0.23 0.68 1.03 1.76 3.00 4.63 7.43 11.42 18.20 28.20 44.21 69.00 107.81 168.20 263.00 } arcseconds. These radii correspond to circular apertures enclosing an integral number of pixels: { 1, 9, 21, 61, 177, 421, 1085, 2561, 6505, 15619, 38381, 93475, 228207, 555525, 1358149 }.

`profMean`
: The mean surface brightness within the innermost `nprof` annuli, with fixed radii given above; these fluxes may be used to recover the annular counts exactly (counts/pixel).

`profMed`
: The 'median' surface brightness within the innermost `nprof` annuli, with fixed radii given above (counts/pixel).

`profErr`
: An estimate of the uncertainty in the profiles (counts/pixel); note that this is *not* the photon noise (which can be recovered from `profMean`),

[1]See: http://www.astro.princeton.edu/~rhl/photomisc/ellipticity.ps

but an estimate of the true uncertainty allowing for contamination by stars, HII regions, etc.

`iso_rowc, iso_colc, iso_a, iso_b, iso_phi, iso_rowcErr, iso_colcErr, iso_aErr, iso_bE`

The centre, major and minor axes, and position angle of a certain isophote (Pixels). These are determined from the 2-dimensional extracted profile. The `Grad` quantities are correction terms allowing us to correct for errors in the photometric calibration.

`r_deV, I_deV, ab_deV, phi_deV, r_deVErr, I_deVErr, ab_deVErr, phi_deVErr`

Parameters of the de Vaucouleurs profile that best fits the radial profile (as determined by the cell array), and errors. The `r` and `I` parameters are the effective radius and the surface brightness at that point.

`r_exp, I_exp, ab_exp, phi_exp, r_expErr, I_expErr, ab_expErr, phi_expErr`

Parameters of the exponential profile that best fits the radial profile (as determined by the cell array), and errors. The `r` and `I` parameters are the effective radius and the surface brightness at that point.

`star_L, exp_L, deV_L`

Likelihoods for the fits of the model by the PSF, an exponential disk, and a de Vaucouleurs profile. More specifically, the values quoted the probabilities of finding a value of chi^2 at least as large as that found for the model fits.

`fracPSF`

The fraction of the total light in the profile that can be assigned to a point source. *Not currently calculated.*

`texture`

A measure of the roughness of the object, based on the residuals after inverting the image and subtracting.

`flags`

Some more information about how the processing went. Note that these

values are also used in `objc_flags`, where they are used to summarise information about the object as a whole, rather than a single band. Current possibilities are:

OBJECT1_NOTDETECTED
: Object wasn't detected in this band

OBJECT1_BRIGHT
: Object was found by findBrightObjects

OBJECT1_BINNED
: Object contains pixels that were only marked as part of an object after the frame had been binned, and the object finder rerun

OBJECT1_EDGE
: Object was too close to edge of frame to be measured

OBJECT1_PEAKCENTER
: Given centre is position of peak pixel, rather than an MLE fit

OBJECT1_NOPROFILE
: The object was too small for us to be able to measure a radial profile

OBJECT1_TOO_LARGE
: The object is too large for us to measure its profile (i.e. it extends beyond a radius of approximately 263 arcseconds; see SDSS Profile Extraction[2].)

OBJECT1_BAD_RADIAL
: The radial profile extends beyond where its S/N first drops to (??) 1.

OBJECT1_NOSTOKES
: Object has no measured stokes parameters.

OBJECT1_BADSKY
: The sky level is so bad that the highest pixel in the object is *very* negative; far more so than a mere non detection. No further analysis is attempted.

[2]See: http://www.astro.princeton.edu/~rhl/photomisc/profiles.ps

OBJECT1_ELLIPFAINT
: The object's centre is fainter than the isophote whose shape is desired, so its properties are not measured.

Information about measuring Petrosian quantities:

OBJECT1_NOPETRO
: The object has no Petrosian radius

OBJECT1_MANYPETRO
: The object has more than one Petrosian radius; the largest found is adopted

OBJECT1_PETROFAINT;
: At least one possible Petrosian radius was rejected as the surface brightness at r_P was too low. If NOPETRO isn't set, an (different) acceptable Petrosian radius was found.

OBJECT1_PETRO_SMALL
: Object has no Petrosian radius; used `f5` instead.

OBJECT1_PETRO_BIG
: Not currently used

OBJECT1_MANYR50
: An object has more than one 50% light radius

OBJECT1_MANYR90
: An object has more than one 90% light radius

OBJECT1_INCOMPLETE_PROFILE
: The circle r = r_{P,r'} intersects the edge of the frame

Information about pixels contained in the object:

OBJECT1_CR
: Object contains at least one pixel flagged as belonging to a cosmic ray.

OBJECT1_INTERP
: The object contains at least one pixel that has been interpolated

OBJECT1_NOTCHECKED
: The object contains at least one pixel that is marked as having not been searched for objects

OBJECT1_SATUR
: The object contains at least one saturated pixel

OBJECT1_SUBTRACTED
: Bright wings were subtracted from this object (presumably a star).

Various flags associated with the deblender:

OBJECT1_BLENDED
: Object was found to be blended, and has children

OBJECT1_CHILD
: Object is a deblended child

OBJECT1_NODEBLEND
: No deblending was attempted, although the BLENDED flag is set

OBJECT1_DEBLENDED_AS_PSF
: The deblender treated the object as a PSF

OBJECT1_DEBLEND_PRUNED
: The deblender deleted some of the peaks that we in the parent object

`type`
: The type assigned to the object in this colour; the possibilities are described for the `objc_type` field.

15.3 Petrosian Quantities

(This discussion is stolen from and supersedes the corresponding parts of Michael Strauss' document *Galaxy Selection Algorithm for SDSS*).

Needs a rewrite! Let I(r) be (a spline fit to or other smooth representation of) the measured azimuthally averaged surface brightness profile of an object in r'. Define the Petrosian ratio R_P(r) as the ratio of the local surface brightness at radius r to the mean within r:

$$
R\_P == \frac{\int_{0.8r}^{1.25r} I(r')\, 2\pi r'\, dr' \,/\, [\pi (1.25^2 - 0.8^2) r^2]}{\int_{0}^{r} I(r')\, 2\pi r'\, dr' \,/\, [\pi r^2]}
$$

Mark all the radii r_i, i = 1,..., N where R_P falls to a specified value f_1, and for which I(r) > f_2.

If there's at least one such radius (N > 0), the largest of the r_i will be taken as the Petrosian radius r_P; if N = 0, the adopted radius will be a Kent radius one given by the solution to <I(r)> = f_3, the point where the mean surface brightness falls to some value f_3. If more than one such Kent radius exists, the smallest is adopted; if no such radius exists, there are two possibilities: that the lowest surface brightness in the object is above f_3 (in which case we adopt r_P = r_max, the largest "good" radius in the profile), or that the highest surface brightness is below f_3 (in which case we take r_P = f_5).

We shall replace this use of a Kent radius in the near future, preferring to use photo's best fit model to extrapolate the Petrosian ratio until it reaches a value f_1.

The Petrosian flux F_P is defined as the total flux as measured within a certain number of Petrosian radii:

$$
F\_P = \int_{0}^{f\_4\, r\_P} I(r')\, 2\pi r'\, dr'
$$

The Petrosian half-light r_{50} is defined by the implicit equation:

$$
\int_{0}^{r\_50} I(r')\, 2\pi r'\, dr' = 0.5\, F\_P
$$

```
|
' 0
```

The Petrosian 90% radius is defined by the implicit equation:

```
, r_90
|
| I(r') 2 pi r' dr' = 0.9 F_P
|
' 0
```

How should we set these surface brightness values? Michael Strauss suggests the following: For a Freeman disk (central surface brightness in r' of 20.85, using the B-r' colors of disks from Frei and Gunn), the Petrosian ratio falls to 1/4 at 3.21 scale lengths, and to 1/8 at 4.43 (with corresponding surface brightnesses of 24.33 and 25.66). The Kent surface brightness at these radii are 22.83 and 23.40. If we decide that f_2 should be a magnitude fainter than the value for a Freeman disk, and take f_5 to be twice the fibre radius, we arrive at the two possible sets of strawman values given in the following table:

f_1	Petrosian Ratio	1/5	1/8
f_2	Minimum Surface Brightness at r_P	25.3	26.7
f_4	Multiple of r_P for Petrosian flux	3	2
f_3	Fallback Radius	3''	3''

These numbers have changed

Chapter 16

Miscellaneous Algorithms

Here we discuss algorithms whose discussion doesn't seem to belong in any specific part of the overview.

- Using Bresenham algorithm's to interpolate.
- Interpolating over bad pixels.
- The Software Bias.
- Removing the Bias due to Choosing the Minimum of two Pixel Values
- Determining a Frame's Statistics using regStatsFromQuartiles
- Fitting *Taut Splines* to Data
- Using `asinh` to Control the Dynamic Range of Data that can be Negative
- How to Estimate the Mode of a distribution

16.1 Using Bresenham algorithm's to interpolate

When doing linear interpolation, it is possible to use a variant of Bresenham's algorithm (usually used in driving bit-mapped displays). Doing so is fast, and photo makes use of this trick. Apart from speed, and rounding errors, this is equivalent to a naive linear interpolator.

16.2 Interpolation

Given a set of pixel values, `I[i]`, with one value (say `I[0]`) missing, how should we go about replacing it? Due to laziness on the part of RHL, we restrict ourselves to 1-dimensional interpolation; the adopted algorithm would trivially extend to working in 2-d, but the book-keeping is a little intimidating.

The simplest solution would be to interpolate linearily,

```
I'[0] = (I[-1] + I[1])/2,
```

but this doesn't perform well for marginally sampled data.

We know something about the auto-correlation present in our data, as it has been convolved with the PSF; we should look for interpolation schemes that minimise errors when interpolating over missing data in a signal consisting solely of the PSF.

16.2.1 Interpolation: Theoretical Considerations

As usual, there are a variety of ways of considering this problem, all of which are closely related. We give three approaches below.

- Brute Force
  If we approximate the PSF as a Gaussian, experimentation indicates that the mean error in interpolating is minimised for a filter of the form

  ```
  I'[0] = (-I[-2] + 3*I[-1] + 3*I[1] - I[2])/4
  ```

  for a reasonably wide range of FWHM (providing that the data is at least marginally sampled). This is the filter that we've adopted for the photometric pipeline.

- Arguments based directly on Nyquist's Theorem
  Consider the following argument:
  If the data were perfectly band-limited, with no power at the Nyquist limit, we could consider our signal to be the original data added to a delta function of amplitude `-I[0]`; consider Fourier transforming this — it's clear that we'll have power `-I[0]` at the Nyquist limit, which is the value that we want. The formula giving the Nyquist power is

  ```
  ... - I[-3] + I[-2] - I[-1] - I[1] + I[2] - I[3] + ...
  ```

so our estimate becomes

```
    I'[0] = ... + I[-3] - I[-2] + I[-1] + I[1] - I[2] + I[3]
+ ...
```

and the formula given above is seen as a suitably belled version of this result.

- Linear Prediction
  We would like to thank Bill Press for discussions on this topic. Note carefully that the *Linear* in Linear Prediction does not mean linear interpolation.

  The basic idea is to divide the I[i] into signal and noise, I[i] == s[i] + n[i], and then to estimate value of pixel ?, S[?], as a weighted sum:

  ```
      S[?] = Sum_i d_i I[i] == Sum_i d_i (s[i] + n[i])
  ```

  We can then attempt to mimimise the expectation value, E, of the squared residual S[?] - s[?]:

  ```
      E == <(S[?] - s[?])^2> = <(Sum_i d_i (s[i] + n[i]) - s[?])^2>
  ```

  i.e., differentiating with respect to d_i,

  ```
      <(s[i] + n[i])(S[?] - Sum_j d_j (s[j] + n[j]))> = 0
  ```

  or

  ```
      < s[i]S[?] > = Sum_j d_j < (s[i] + n[i])(s[j] + n[j]) >
                   = Sum_j d_j < s[i]s[j] > + < n[i]n[j] >
  ```

  if we assume that signal and noise are uncorrelated. If we assume that our images consists of uncorrelated PSFs, then <s[i]s[j]> == e^{-(i-j)^2/alpha^2} and <n[i]n[j]> == N_i delta_{ij} and we may invert this matrix equation to estimate the d_i. Details (and figures) are provided in Linear Prediction and Interpolation[1].

16.3 Software Bias

As we are using unsigned 16-bit integers, bias and sky subtraction could lead to negative pixels, which would be a problem; photo gets around this by adding an arbitrary constant to the data just after removing the bias (it's

[1] See: http://www.astro.princeton.edu/~rhl/photomisc/interpolate.ps

currently 1000 and is a #define in the C code); when sky is subtracted the soft_bias is added back in (and removed from the sky estimate!). As there was a hardware bias of this order in the raw frames, doing so doesn't reduce the dynamic range of the data.

16.4 Removing the Bias due to Choosing the Minimum of Two Pixel Values

It is easy enough to calculate that the expectation value of the minimum of two Gaussian variates (with zero mean and unit variance; N(0,1)) is -0.5641895835. If we ever replace a pixel value by the minimum of two estimates of its true value, we will systematically underestimate its value; we should (and do) add a correction of approximately 0.56sigma.

16.5 Determining a Frame's Statistics using regStatsFromQuartiles

The routine `regStatsFromQuartiles` constructs a histogram of pixel intensities for an image, and derives any of a wide range of parameters. See the source code for details.

16.6 Fitting *Taut Splines* to Data

A *Taut Spline* is a regular (cubic) spline with the extra constraint that there be no superfluous turning points; it's rather as if you took the piece of flexible wood that provides the term spline[2] and pulled on each end to remove unwanted wiggles.

This is achieved by inserting extra knots at cleverly chosen places, and seems to work well in practice.

[2]See: http://www.princeton.edu/OED/oed.html

16.7 Using `asinh` to Control the Dynamic Range of Data that can be Negative

It is a common practive to take logarithm of numbers that span many orders of magnitude, e.g. to display an object's radial profile. This has two main disadvantages:

- If the data is non-positive, the logarithm is badly behaved
- The errors on very small values become vary large (and 1-sigma ranges can easily be negative).

Both of these problems can be circumvented by using `asinh(x/Delta)` instead of `lg(x)`, where `Delta` is some constant with magnitude comparable to the expected error in x. This has the nice property of being (within a scale) a simple logarithm for x >> Delta, and approximately linear for x $<\sim$ Delta.

For more details, see the document[3] by Szalay, Lupton, and Gunn.

16.8 How to Estimate the Mode

The result that `mode = 3*median - 2*mean` is readily proved from a Gram-Charlier series up to the third Hermite polynomial; to the same order it's easily shown that

```
mode = median - (Q25 + Q75 - 2*median)/eta^2
```

where `eta` $\sim$ `0.673` is the 75th percentile of an `N(0,1)` Gaussian. We can then estimate the mode as

```
5.4*median - 2.2*(Q25 + Q75)
```

The variance of this estimate is approximately

```
(4.78)^2 sigma^2/N;
```

much worse then the mean (with variance `sigma^2/N`), and 3.8 times worse than the median (whose variance is `(pi/2)sigma^2/N`).

[3] See: XXX

Chapter 17

Known Bugs

This is a list of known bugs and problems, as of 2nd December 1997.

- (Bug #0). We use readnoise/gain as an estimate of the dark standard deviation, rather than measuring it (which would also allow for dark current).
  - This could be fixed using the bias frames.
- (Bug #1). PSP cannot yet use "jumbo stamps" provided in scFrame* files. These stamps are needed to characterize the PSF behavior on scales comparable to the frame size.
- (Bug #2). PSP does not interpolate sky on a frame by frame basis (as it does the flat-field vector and bias drift), and does not calculate the sky slope.

Chapter 18

References

"The Telescopic Point_Spread Function"
Racine, R. 1996, PASP 108, 699-705..