Short manual of the *multiking_addACS* procedure Version 1.2

Maurizio Paolillo

November 18, 2010

Abstract

This document describes the use of the IDL *MultiKing* procedures to generate and add simulated PSF or Globular Clusters to HST ACS observations, simulating the effect of dithering on the final drizzled and stacked image. The script can be easily modified to add any type of object.

NEW:

- *Version 1.2-*

  a. bug fixed: the code was not checking the drizzling kernel used by the reference image, and was thus using whatever kernel the user had defined in its drizzle preference file. The fix allows the code to drizzle with the same kernel used in the reference image.

  b. The documentation was updated to reflect the fact that files in use by TinyTim are not required anymore, since TinyTim is implemented in the code but not actually used (see below for further details) and also discouraged for it's limitations.

  c. the code now is able to generate PSFs of with the total magnitude requested by the user, as was alrady possible for GCs (In previous versions PSFs were always normalized to 1).Tthe use of the package to

generate PSFs, based on the Anderson templates, is better described.

- *Version 1.1*- fixed a bug which did not allow the code to properly handle simulated sources overlapping edges. This happens when reading from an input list, since randomly created sources avoid the field edges by design.
  The code now also allows to include Poissonian errors in the mask files due to simulated sources, for final drizzling. This is intended to be used with the ERR weighting option of *Multidrizzle* [1], which computes correct error maps including all error sources.

1 Introduction

This task is based on the IDL procedure *multiking_addACS.pro*, aimed at generating and add simulated PSF or Globular Clusters to HST ACS .flt images. The final images can be combined with the Multidrizzle package [1] as regular ACS images.

The combination of several .flt files in one final drizzled image can be automatized through the shell *multiking_wrapper.csh* which allows to simulate a list of objects in given positions using *multiking_addACS.pro*, add them to a set of dithered observations and combine (drizzle) the individual observations into the final image. Examples for both cases are provided below.

We strongly encourage users to check the Limitations of the script in §3 using trying the code, and contact the developer if specific help is needed.

Required files:

- *multiking_wrapper.csh*: wrapper script to simulate a list of objects in several dithered observations and combine them in a final drizzled image. Uses *multiking_addACS.pro*.
- *multiking_addACS.pro*: IDL procedure which creates multiple GCs and adds them to a blank or real HST ACS .flt image
- *kingmod_addacs.pro*: IDL procedure which creates the individual noisy and PSF-convolved GC images to be passed to *multiking_addACS.pro*
- *get_dithered_coords.csh*: c-shell to invoke the pyraf tran task to convert between distorted (flt) and undistorted (dth) coordinate grid
- original *_flt.fits ACS files
- final dithered image for astrometric reference

- *_idc.fits and *_dxy.fits distorsion files from the archive, *coeff?.dat files produced during the drizzling phase to be used by the pyraf tran routine
- *final_mask?.fits files produced during the drizzling phase if adding to original image, to allow to apply the same masking
- OPTIONAL: *makeHSTpsf.csh*: c-shell to create HST PSF with TinyTim with the appropriate spectrum and detector position, if you decide to edit the source code to bring back TinyTim as PSF generator
- OPTIONAL: *elliptical_spec.dat*: spectral template for TinyTim in ASCII format (see TinyTim help for more information), if you decide to edit the source code to bring back TinyTim as PSF generator

Required software:

- *IDL*: (I tested using version 6.0 on a Linux system)
- *Pyraf* and the **Dither** package: including the *tran* routine within the Dither package to convert .flt pixel coordinates to world coordinates, and the *drizzle* task to stack the dithered observations.
- OPTIONAL: *TinyTim* HST PSF simulator (I tested using version 6.1) can be used if you don't want to use the Anderson PSF libraries [2]. However this requires to manually uncomment part of the IDL procedure and is discouraged since it is not optimized for charge diffusion effects and subpixel sampling.

Output:

1. simulated HST ACS image, named according to the IDL input
2. simulated GC parameter list, named according to the IDL input
3. log file of the makeHSTpsf.csh and TinyTim output: makeHSTpsf.log

4. OPTIONAL: if the option to add Poisson noise to mask files is chosen, the modified mask files *_flt?_mask1.fits and *_flt?_mask2.fits are produced for each input flt file.

5. OPTIONAL: simulated PSF at each GC position are produced as files: king_*_*_*_psfconv.fits, but deleted on successful completion of the procedure. To preserve temporary PSF-convolved GC fits images comment this line in *multiking_addACS.pro*:
`spawn, 'set chck = ‘ls | grep _psfconv.fits‘ ; if ( $#chck > 0 ) rm -f king_*_psfconv.fits'`

6. OPTIONAL: temporary TinyTim file: tmp.par (see discussion on *Tinytim* above)

2 Usage

2.1 Simulating and drizzling with *multiking_wrapper.csh*

To generate a simulated drizzled ACS image starting from a list of object positions and a set of ACS observations (including the final dithered file) run the *multiking_wrapper.csh* script followed by the input parameters and files:

Input parameters explanation:

1) File containing list of .flt files on which you want to add simulations
2) Reference drizzled image for astrometry (usually obtained with .flt files as in #1)
3) Rootname of output drizzled image with simulated objects (simulated .flt files will be named accordingly)
4) Simulated .flt should be created as orig.flt+simul [a] or just blank+simul [n]?
5) Add Poisson noise of simulated sources to original weight maps to produce correct error maps in final drizzling [y/n]? (to be used for ERR weighting in multidrizzling)
6) Create GC [gc] or stellar [psf] objects?
7) Generate random sample within assigned range [p] or read obj.params from file [r]
8) Output filename with simulated objects parameters
9) If (7)=p: number of objects to simulate;
   If (7)=r: File with parameters of objects to simulate. Note that input parameter file requires specific format as included template ander-

son_coords_wcs_center.lst, i.e. identical to the output file with simulated objects parameters of #8 (see 2.3 for more details)

10-15) If (7)=p: range [min max] of core radius (pixels), conc.index and mag of objects to simulate

Examples:

1. Generate a blank drizzled image with a grid of PSFs at specific coordinates. This example allows to create a PSF library using Anderson closest templates [2] and further accounting for your specific dithering pattern. Note that the code just uses the nearest Andersen template instead of interpolating first[1]:
   > source multiking_wrapper.csh flt.lst N1399_center_drz_sci.fits N1399_psf_center n n psf r psf_simlist.lst anderson_coords_wcs_center.lst | tee multiking_wrapper.log

2. Add 10 randomly simulated GCs to an actual ACS dithered image, further including Poisson errors of simulated sources in weight maps:
   > source multiking_wrapper.csh flt.lst N1399_center_drz_sci.fits N1399_GC_center a y gc p GC_simlist.lst 10 0.01 5.0 0.1 20 21 27 | tee multiking_wrapper.log

3. Add 10 simulated GCs to an actual ACS dithered image, reading the positions and parameters from an input file (following the input format described above), further including Poisson errors of simulated sources in weight maps:

[1] A refined approach requires the user to generate the Anderson grid at the original Anderson positions, and then interpolate them; this can be done by transforming the Anderson pixel coordinates (see provided library files) to RA, Dec coordinate of your image through the IRAF *tran* task, and then using these as a list of input coordinates to the multiking procedure.

> source multiking_wrapper.csh flt.lst N1399_center_drz_sci.fits N1399_GC_center a y gc r GC_simlist_center.lst GC_input_list.lst — tee multiking_wrapper.log

2.2 Simulating individual images with *multiking_addACS.pro*

To produce a single ACS image with simulated GC using *multiking_addACS.pro*:

> idl
IDL> .run multiking_addACS.pro

then provide the required parameters. Standard templates are provided in the following examples.

Examples:

1. Simulate 4 GCs and produce a blank image with simulated objects, without modifying weight masks:

   IDL> .run multiking_addACS.pro
   Reference ACS flt image? j8zq07a1q_flt.fits
   Final dithered reference ACS image (_drz_sci file)? N1399_sw_drz_sci.fits
   Output simulated ACS image with GC? ACS_GC.fits
   Add simulation to original image or create new image? [a/n] n
   Perform simulations or read GC parameters from file? [p/r] p
   Output file with GC parameters for reference image? simulated_GC.lst
   Number of GC to simulate (> 1)? 4
   Core radius range in pixels [min max]? 0.05 0.2
   Concentration index range [min max]? 10 100
   Total V magnitude range [min max]? 20 22
   Modify original weight masks to include sim.source Poisson noise (assumes ERR weighting in multidrizzling) [y/n]? n

2. Read 4 GCs positions and add simulated objects to original ACS image, including Poisson errors of simulated sources in weight masks:

   IDL> .run multiking_addACS.pro
   Reference ACS flt image? j8zq07zvq_flt.fits
   Final dithered reference ACS image (_drz_sci file)? N1399_sw_drz_sci.fits
   Output simulated ACS image with GC? ACSplusGC.fits
   Add simulation to original image or create new image? [a/n] a
   Perform simulations or read GC parameters from file? [p/r] r
   Input file with simulated GC parameters? simulated_GC.lst
   Output file with GC parameters for reference image? simulated_GC2.lst
   Modify original weight masks to include sim.source Poisson noise (assumes ERR weighting in multidrizzling) [y/n]? y

2.3 Source parameters file

To simulate a custom set of sources (either GCs or PSFs) the user must provide positions and parameters of such sources. The procedure uses a fixed-format template, that follows the structure of the included file anderson_coords_wcs_center.lst which can be modified for this purpose. Such file has 10 lines of header and 12 columns of parameters. Not all parameters have to be updated by the user, since the code will update some of them in the running stage, but all columns must be present with the correct format.
The parameters are:

N : Source number, only used for tracking purposes

RA,Dec : Source position relative to the astrometric solution used in the

final drizzled image. User is required to provide this to simulate at specific positions.

X,Y : Pixel position of source centroid. These values change for each .flt file, and are thus computed by the procedure based on the final drizzled image; the user is not required to provide them, but 2 columns of floating point data must be present here to satisfy the reading format (in the template anderson_coords_wcs_center.lst file for instance, for simplicity the Ra, Dec columns are repeated here).

r_0, Conc.index, Tot.mag. : structural parameters of the GCs to simulate, identical to those of points 10-15) above.

X,Y cent.in pixel : subpixel centroid position. These values change for each .flt file, and are thus computed by the procedure based on the final drizzled image; the user is not required to provide them, but 2 columns of floating point data must be present here to satisfy the reading format (in the template anderson_coords_wcs_center.lst file for instance, for simplicity a fixed value of 50. is used here).

WFC chip : WFC chip on which the source will fall. This is used by the script *get_dithered_coord.csh* to run the IRAF *tran* task on the correct chip. Note that the script assumes that each source falls always on the same chip in all .flt file (see 3 for more details). The user is required to provide this parameter.

PSF file : Anderson PSF library file for each position. Since the script uses the closest PSF in the Anderson library in terms of *.flt pixel positions, this file depends on the **X, Y** values described above. A string must be present here, but again the user does not have to provide this since the script will do it later (in the template anderson_coords_wcs_center.lst file for instance, for simplicity a fixed string "aaa" is used here).

3 Limitations and warnings (please read!)

- The *multiking_wrapper* script has been designed to automatize the whole simulation and dithering process for images with small ditherings, where the simulated sources fall always on the same WFC chip. Otherwise the *tran* task may not reproduce the correct coordinates, yielding incorrect results. As a workaround for such cases, the user may run the IDL *multiking_addACS* procedure on each image separately with different position list files, accounting for the correct chip, and then drizzle manually the images together as done in the final part of the *multiking_wrapper* script.
- The procedure assumes that there are sources on both ACS chips, so when providing a list of sources include at least a source for each of them.
- Photometric filter is read from input image but counts/flux conversion is hardcoded in the procedure (assuming F606W band) and must be changed manually in *kingmod_addacs.pro* if filter changes. Also Anderson library currently included in the package is for F606. Need to change this as well if you want to use a different band.
- The scripts have been developed for specific purposes and thus the paths and several parameters are not generalized but still hardcoded into the software.
- The accuracy of the code depends on the accuracy of the Anderson PSF library [2] (or of *TinyTim* if you edited the source code to use it) in simulating the actual ACS PSF, depending on which is used to generate the simulated PSF. Current version uses Anderson library [2], but the code includes a section to use *TinyTim* (must be edited manually)
- initial X,Y pos within pixel is hardcoded in the procedure to be at

the center of a pixel when performing simulations, but derived from Ra,Dec if reading positions from file.

- Due to ACS conventions det1 is upper one, det2 is lower one, but the fits extensions in the final image follow the opposite rule.

References

[1] Koekemoer, A. M., Fruchter, A. S., Hook, R., Hack, W. 2002 HST Calibration Workshop, 337

[2] Anderson, J., & King, I. R. 2006, Instrument Science Report ACS 2006-01, 34 pages, 1