MultiDrizzle: Astronomical Image Analysis Software for Python

Warren Hack, Christopher Hanley, Robert Jedrzejewski, Ivo Busko, Anton Koekemoer

Space Telescope Science Institute
Baltimore, MD

1. A lot of work has been done lately to develop Python tools for scientific analysis of data, including tools for analyzing astronomical data. MultiDrizzle has been developed as one such tool.

2. This talk will demonstrate MultiDrizzle’s capabilities, and how Python made it all possible.

3. This package was developed by Warren Hack, Christopher Hanley, Robert Jedrzejewski, Ivo Busko, and Anton Koekemoer at the Space Telescope Science Institute (STScI).
Astronomical Data Analysis

Most astronomical data analysis applications rely on either IDL or IRAF. IDL provides:
- Commercial high-level interpreted array processing language
- Simple array operations
- Large library of user-contributed tasks

IRAF, in comparison, provides:
- Freely available environment with scripting capability
- Large library of tasks for data analysis developed over the years
- It’s compiled language is a FORTRAN/C hybrid unique to IRAF
- Scripts and compiled tasks are strictly procedural
- Scripts rely on file I/O for image data and parameter values
- Little capability for error handling in scripts
- Python has recently been introduced for use with IRAF (through PyRAF)

1. Most astronomers also have access to other data analysis packages.
2. These other packages, though, are similar in many ways to IRAF in that they were developed as ‘insular’ environments specific to the type of observations to be analyzed.
3. A very few (perhaps only one) are object-oriented.
4. Most have limited scripting capabilities, at least compared to Python, while supporting native-compiled tasks written in a language specific to the environment or using interfaces specific to the environment.
STScI Calibration Pipeline

STScI automated pipeline calibration software was developed exclusively using IRAF, until the implementation of the Advanced Camera for Surveys (ACS) pipeline in 2001.

ACS pipeline calibration processing includes:
- Basic calibration software, CALACS, developed under IRAF
- Automatic combination of dithered images while removing distortion.
  - Final combined products still contained cosmic-rays

1. Instruments installed on the Hubble Space Telescope prior to the installation of ACS all used IRAF and C exclusively for their pipeline calibration software.

2. Each new instrument added more calibration steps to the automated pipeline allowing for processing of more varied types of observations and for better calibrated products.

3. Having new capabilities available for ACS allows observers to take their data in novel ways to increase the quality of the science that can be done with the data. Dithering, for example, has users take a series of images slightly offset from each other in order to not only identify features from each pointing which are not present in the other images so they can be removed, but also to allow them to be combined at a potentially higher resolution. This assumes, of course, that the software exists to support such image combination or the identification of such bad data.
1. This is a single exposure from the Advanced Camera for Surveys (ACS) Wide Field Camera, one of three separate cameras built into the ACS.

2. This particular camera consists of 2 CCD chips physically mounted next to each other in the camera. The gap represents the limit how close they could mount the chips to each other.

3. Most of the faint star-like dots and streaks on the image are actually cosmic-ray hits on the camera as the image was being taken. A couple examples are pointed out in the magnified section.

4. Cosmic rays are energetic particles that hit the detectors adding counts.
1. The view of the image in this slide, and on subsequent slides, uses an inverse color-map where bright objects are black and the background is white. This allows fainter features to be more easily seen.

2. These images have been calibrated to remove effects due to the camera itself, such as differences in sensitivity from one pixel to the next (referred to as ‘flat-fielding’) and electronic noise generated by the chips (known as dark and bias corrections).

3. The overall faint gray background (‘haze’) outlines the extent of each chip and represents the overall brightness of the sky and residual electronic noise (dark and bias counts) generated by the detector itself as seen during the exposure.

4. Again, most of the bright points that appear to be stars are really cosmic rays.
Image Combination

Multiple images can be used to produce a final calibrated product from the pipeline. Each input, though, will have:
- Cosmic rays
- Bad/hot pixels and defects

The final product will be made from:
- Multiple observations of the same portion of the sky
- Observations may be offset slightly from each other

Why not just shift and add the images?

1. The final calibrated product from the pipeline can be created by combining multiple input images.
2. Each input will not only contribute cosmic-ray artifacts, but they may also be taken at slightly different pointings.
3. Combining the images taken at different pointings requires some alignment to be done between the inputs images before they can be added together.
4. The simplest method to align offset pictures would be a simple ‘shift-and-add’ technique, such as the one used in the pipeline for images from HST’s NICMOS (near-IR) camera. Why not do that here?
1. **PLEASE NOTE:** As the 2 images blink, we see the effects of the distortion as stars near the edge seem to jump back and forth while the star in the circle remains steady.

2. This demonstrates how optical/geomteric distortion allows the images to be aligned on one point in the image (like the star in the center), but then objects farther from that point get progressively offset from each other. So, no matter what star you get to match, every other object in the image’s will be offset and never align due to the distortion.

3. You may also have non-integer shifts that would also pose a problem for shift and add techniques.
So how do you combine images then?
What is MultiDrizzle?

Prototype version was developed as a Python script using IRAF tasks in Jan 2004 to:
- Combine images taken at slightly different pointings
- Remove fixed image defects/detector gaps
- Remove image distortion
- Remove cosmic rays and other transient events

Images from basic calibration software, CALACS, serves as input for MultiDrizzle.

MultiDrizzle provides the ability for the user to process their data to take advantage of the new observation strategies, such as dithering, to improve the scientific return of their data.
MultiDrizzle Procedure

Initially implemented as a Python script calling IRAF tasks to perform the steps:

1. Mask known defects.
2. Subtract the sky level from each detector.
3. Undistort (‘drizzle’) the input images onto separate, registered outputs.
4. Combine the drizzled images to create a “median” image.
5. Distort back the median image to the geometry of the input images.
6. Identify cosmic rays by comparing the distorted median image to the input images and mask them.
7. Combine (‘drizzle’) all masked input images into the same output to create the cleaned, sub-sampled, mosaic image.

Prototype originally distributed for use by external community starting in January 2004.

1. The algorithm shown here reflects the basic steps necessary for this process to adequately detect, flag, then remove cosmic-rays and other transient defects from the images.

2. The original prototype described here relied exclusively on IRAF tasks to perform these operations.

3. The implementation of this process as an IRAF script, rather than a compiled task, required that all image processing operations result in image I/O, with multiple processing operations possible for each individual step in the algorithm. This resulted in a tremendous amount of image I/O and the creation of numerous intermediate image products which needed to be removed upon completion of the task.
1. The geometric distortion that prevented the shift-and-add technique from working has been removed from this image.

2. This image has been centered on the same spot as all other images it will be combined with. In addition, it has the same orientation and pixel scale (distance on the sky from one pixel in the image to another).

3. These images have already been ‘shifted’ and corrected, and can now be ‘added’ as desired.

4. Cosmic rays are still present, though.

5. This image could be created a finer or coarser output grid, depending on what was desired.
1. 6 ACS WFC images went into creating this median image.
2. Due to their different pointings, 3 images are in the top left, 3 are on the lower right.
3. Noise is higher in the gap and edge regions because fewer images overlapped there.
1. In order to mark the bad pixels or cosmic-rays in the original input images, the median image needs to be sliced and have the distortion added back in to create comparison images.

2. The comparison images match the original input images positions exactly, with the exception that all bad pixels and cosmic-rays are gone.

3. The magnified sections of show the differences.

4. The differences will be used to create a mask for the original image flagging all the bad pixels and cosmic-rays that were removed in the median image.
Using our sky subtracted images, static pixel mask, cosmic ray masks as input we create our distortion corrected output frame from all the original input images.

Image is photometrically flat, distortion free, and cleaned of all (?) cosmic-rays and bad-pixels right out of the pipeline.

1. The original images, with the masks created in the last step, are then drizzled one more time to create a single output image.
2. This output image has been drizzled so that North is up.
3. This alignment can be useful for comparison with images of the same area of the sky taken by other telescopes or satellites.
4. The final product is being displayed here using false colors to highlight the details in the objects.
Why re-write this **Python script** based on IRAF tasks if it was already being used by astronomers?

The version of MultiDrizzle described so far had originally been developed as a Python script based on IRAF tasks.

The redesigned version does exactly the same image processing steps, except as a native Python task relying on some C extensions and ‘drizzle’ as a Fortran task.
MultiDrizzle Automatic Pipeline Requirements

- Minimize File I/O
  - *This is a problem for IRAF scripts since all array-like calculations typically require file I/O.*
- Error trapping
  - IRAF scripts have no ability to trap errors. When an IRAF script crashes, usually no meaningful diagnostic information is returned.

1. The HST pipeline is dealing with a vast amount of data and we don’t want to saturate the I/O ports.
2. The HST pipeline needs to be as automated as possible. When errors do occur, it is critical that they be diagnosed quickly. IRAF scripts have no ability to trap errors when they occur. It makes diagnosing problems with pipeline processing of a dataset difficult, if not impossible without trying to reproduce the problem interactively on another system. This can be a time consuming process.
MultiDrizzle Automatic Pipeline Requirements (cont’d)

- Robustness
  - This **HAS** to successfully finish processing and produce a product **every** time.
- IRAF Parameter files
  - Minimize dependence on IRAF par files to avoid multiple processes colliding when updating par file for same task
- Processing time is an issue
  - The HST data processing pipeline needs to be able to process a day’s worth of data in significantly less than a day. If the program is I/O bound, we could run into problems.

Robustness
1. In our experience IRAF scripts have proven not to be reliable enough for a production environment.
2. Earlier attempts at Multidrizzle like processes using IRAF scripts proved to be un-maintainable.
3. IRAF task’s usage of parameter files causes it to use the same parameter file for multiple IRAF sessions. This prevents more than one process from using the same IRAF task at one time, whereas the pipeline needs to run multiple instances in order to keep up with the data. The prototype, therefore, would never work in the pipeline environment.
4. The smallest changes to the data or algorithm tended to break the code. This made the code hard to change.
5. Also, due to the lack of error trapping, when things broke down, they were very difficult to diagnose.

Processing
1. In addition to pipeline processing all data from the Advanced Camera (ACS), all archival requests for ACS data are also reprocessed with MultiDrizzle.
2. This can lead to 20 – 100GB of data being processed per day.
MultiDrizzle Redesign

Algorithm used by MultiDrizzle needed to be completely re-implemented to replace all IRAF task functionality.

Python provided numerous advantages over IRAF to be used in this redesign:

- Object-oriented
- Good error handling
- Fast, especially with C extensions

In addition, Python libraries and modules useful for astronomical data analysis already exist; specifically, PyFITS and numarray.
PyFITS and NUMARRAY

The MultiDrizzle I/O model is based upon the use of a couple of STScI additions to Python:

- PyFITS and numarray.
  - All computations done are on numarray objects.
  - PyFITS allows us to directly read our FITS input as numarray objects and manipulate header information.
  - Pre-existence of this code was fundamental.
  - Numarray objects allow simple array arithmetic just as in IDL; e.g., output = image1 + image2

FITS

1. FITS is a data standard that is used by most astronomers.
2. FITS : Flexible Image Transport System
3. It is a platform independent standard for image storage supporting ASCII descriptions of the data in a ‘header’ and potentially multiple binary images in the same file.
Problems solved

The use of **PyFITS** and **numarray** solved a number of problems:

- With all calculations done on **numarray** objects in memory, file I/O is minimized.
- Using PyFITS allows for a uniform handling of input during processing regardless of how the files were originally structured.
  - Every HST instrument has its own imaging format
  - We can support any new instrument as long as we can convert the science data into a **numarray** object.

1. This is not just necessary for new cameras.
2. Each camera on HST stores their imaging data in their own FITS format designed to mimic the physical camera used to take the data.
3. For example, any single ACS WFC image shown as an example earlier would be stored in a single FITS file with 2 binary images (one for each chip), along with other binary images to record related values for each chip.
4. Thanks to PyFITS and NUMARRAY, the MultiDrizzle algorithms all work the same way for each camera.
1. Here are examples of images from all the other cameras that are currently installed on HST.

2. MultiDrizzle can process all these types of images since they are stored as FITS files due to the use of PyFITS and numarray.

3. Each image would be stored in a FITS file with its own format based on the number of chips that make up the image.

4. The colors in the ACS/WFC, STIS/CCD and WFPC2 images shown here are false colors used to simply highlight the details in each image. White represents the brightest objects, with fainter objects being represented as orange getting darker as the object gets fainter.
MultiDrizzle Issue

MultiDrizzle relies on creating a median image from a set of input images

- It’s necessary to have all of the pixels of the stack for a particular location in memory at the same time.
- Two methods for handling this stack of images:
  - read them all into memory at once, or
  - process them in sections

1. A median image represents the ‘median’ average of each pixel as computed from all the input exposures. This forces a value for each pixel from each input image to be in memory (or accessible to memory) at the same time in order to perform the statistics.

2. Clipping can be used in this process to eliminate pixel values which are wildly different in some input images compared to others before computing the final median value. The wildly different values would be caused by cosmic-rays (which are generally much brighter).

3. The median value, with large enough numbers of input exposures, should eliminate all cosmic-ray hits from the final combined image.
THE BRUTE FORCE APPROACH

All input images in memory at once:
How much memory is needed?
– Worst case scenario in pipeline is 30 images
  • 192 Mbytes x 30 input images = 5760 Mbytes

Or

Far more memory than can be addressed
on a 32-bit operating system
Solution One

Memory Mapping and Iterators:

- We wrote an `imageiterator` class that allowed us to step through a stack of images using a predefined (1Mb) buffer size.
- Using the memory mapping property of `numarray` we planned only to read in the number of pixels defined by our `imageiterator` buffer size.
Solution One (cont’d)

Why doesn’t this work…

• Although only small portions of the data are read into memory at a single time, the operating system still needs to be able to address every pixel in each input image simultaneously.
  – Problem on 32 bit systems with 2Gb address space limitation

Actually pipeline is a 64 bit system where problem doesn’t exist. However, we wish to distribute this software to astronomer’s desktops which poses this 32-bit constraint.
Solution Two

Look to PyFITS developer for help...

- A new attribute was developed for the PyFITS data object called "section", which returns a user-specified slice.
  - Data sections must be contiguous
  - Sections are read directly from the file without having to memory map the entire image.
  - Using sections in our imageiterator got around the 32-bit limit
  - No other code needed to be changed to implement this

This attribute was added to PyFITS version 0.9.6 or later.
Combining the image iterator with the new PyFITS attribute "section" allows us to process large numbers of input images with minimal memory usage and file I/O.

1. The iterator returns sections sequentially from the bottom of the image to the top, as displayed in this example.

2. The sections returned by the iterator contain an integer number of rows from the image up to the buffer limit, which has a default value of 1Mb.
Redesign effort:

- **Started** in October 2003
- **Installed** in HST pipeline for ACS use on 22 Sept 2004
  - Only a small number of problems encountered so far
  - Processing demonstrated to be CPU bound, not I/O bound
- **Made available for public** download in Nov 2004 for stand-alone use either under Python or PyRAF.
- Support for other HST instrument’s imaging data more fully implemented with **patch release** to be made available soon.

The use of Python allowed this package to be developed quickly, while producing code which can be maintained easily.
Supporting Modules

A few new modules developed directly for MultiDrizzle also have more general use.

- **Imagestats**: compute complex image statistics on a numarray object at native C execution speed.
- **Numcombine**: complex image combination operations on multiple numarray objects
- **Irafglob**: interface for handling multiple forms of file input specifications.
- **Nimageiter**: generator of numarray sections for multiple identically-shaped numarray objects

These modules support numarray objects in general, not just astronomical images, and are part of a package distributed alongside MultiDrizzle.
**MultiDrizzle platforms**

Strict reliance on Python and C (for some modules) allows MultiDrizzle to be run on all platforms supported by Python.

- Can be run under Windows, but we have not yet developed a smooth way to install it due to existing Fortran code converted to C.

The same code can be used in the HST pipeline and on astronomers workstations.

1. The Fortran code used for the ‘drizzle’ task was re-packaged so that all IRAF dependencies were removed, primarily file I/O.

2. The core algorithm in Fortran was converted to C using ‘f2c’, while a Python interface C function was written for the drizzle functions used by MultiDrizzle.

3. This reliance on a Fortran-to-C converter adds a requirement for distribution which should not be necessary, especially under Windows where the f2c library needs to be compiled with the same compiler used for compiling drizzle’s C code.
Summary

MultiDrizzle demonstrates that:
- Significant astronomical data analysis can now be done under Python.
- Python can be used to develop tasks for mission-critical automated environments.
- Same code can be used for automated use and stand-alone interactive use.

Enough functionality exists now to start comparing Python with IRAF and IDL as astronomical data analysis environments.
- Development continues on MultiDrizzle and related modules to add more functionality for use under Python.
  - *astrolib*:
    http://www.scipy.org/wikis/topical_software/AstroLib
  - STScI Python:
    http://www.stsci.edu/resources/software_hardware/pyraf/stsci_python

24-Mar-2005 Warren Hack (STScI)
Credits

• Drizzle Algorithm
  – “Drizzle: A Method for the Linear Reconstruction of Undersampled Images”
  Februrary
• Anton Koekemoer (STScI)
  – Original MultiDrizzle Algorithm and Script
• Todd Miller (STScI)
  – NUMARRAY
• J.C. Hsu (STScI)
  – PyFITS

1. We want to acknowledge the very timely assistance provided by Todd Miller and J.C. Hsu.
2. Todd fixed ‘numarray’ bugs as they were found during development.
3. J.C. implemented the ‘section’ attribute in PyFITS.
4. MultiDrizzle could not have succeeded at all without their tremendous contributions.
MultiDrizzle releases and documentation can be obtained from the MultiDrizzle home page.