SPIN Internship Summer 2016: LSST Pipeline Management

Samuel Piehl

August 3, 2016

Abstract

The Large Synoptic Survey Telescope is a telescope currently being built in Chile, which plans to be one of the largest collectors of astronomical. Using current technologies in software engineering and scientific computing, the data and images from LSST can be processed largely automatically [2]. However, two problems arise when attempting to automate these processes while making the system accessible and comprehensible to the larger astronomical community outside of the LSST group: quick access to metadata and ability to run large data sets through the pipeine.

This report presents intermediate solutions to these problems and suggests areas for future development. Two scripts to be integrated into the LSST data processing pipeline are proposed. The first, an addition to `showVisitSkyMap.py` in the LSST skymap tools (`https://github.com/lsst/skymap`), adds an option to display charge-coupled device (CCD) sensor serial numbers on a visualization of the constructed skymap, allowing astronomers to assess and easily access areas of interest, as well as allow for future code developers to identify potential errors. The second is an original script, `summarizePatches.py`, which creates a file that summarizes the current data being processed for use in future processing steps. This takes a step towards permitting the processing pipeline to easily and automatically process large amounts of data.

Introduction

The Large Synoptic Survey Telescope (LSST) plans to be one of the largest astronomical projects ever constructed, producing about 15 TB per night of image data. Currently, the telescope is being constructed on a peak of Cerro Pachn in the Chilean Andes, and is expected to begin operations and data collection in 2022 [1]. All data will be made public, and so will be available to the larger astronomical community, allowing for investigation by many scientists into the structure and composition of the Solar System, the Milky Way, and permit further study of other galaxies and the nature of dark matter and energy [2].

More in depth discussions of the LSST camera and skymap, summarized in this section, are described in the LSST book [2]. The base sensor of the LSST camera is a charge-coupled device (CCD), the camera itself being composed of 189 4K $\times$ 4K CCD sensors. The camera is partitioned into 3 $\times$ 3 arrays of CCDs, called rafts. Each raft is independent and identical to the others, and can function as its own camera. Additionally, there are 4 wavefront sensors located on the corners of the camera, which help to keep the camera pointed accurately. This camera construction is outlined in Figure 1 - where each blue square outlines a single CCD, and each red square outlines a raft. The figure also shows wavefront sensors and guide sensors, which help the camera and telescope in keeping a precise location.

Each image that is taken from the LSST can be projected onto a skymap in order to discuss its location in the sky. These skymaps are rectangular tessellations of the larger survey area (or tract) for each image, and are charted using astronomical units such as right ascension and declination. Each element of the tessellation is referred to as a patch, which is the base unit of the skymap. When an exposure of the LSST camera is made, the CCDs and rafts do not line up exactly with sky patches, and correlation of patches to CCDs is an important detail in processing exposure data.

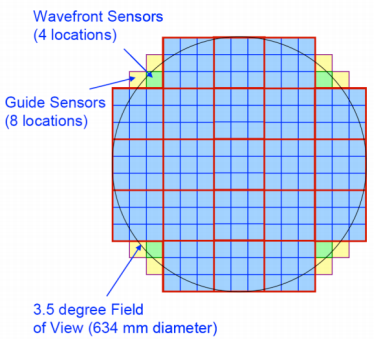

Figure 1: Overview of the LSST camera, image taken from [2]

After image exposures are taken from the LSST camera, the images enter a processing pipeline that implements astronomical and technical tasks such as removing instrumentation bias, determining a World-Coordinate System (WCS), constructing point-spread functions, and identifying astronomical sources. Currently, most processing tasks in the LSST software stack are written in Python and implemented as command line tasks in a Unix-like environment with the LSST software stack. For example, the `processCcd.py` task, which subtracts background, calibrates detections, fits point-spread functions, and refines the WCS mapping, among other processing tasks, is run using the command `processCcd.py /path_to_data_repository/ --rerun demo --id visit=0230069`, where `/path_to_data_repository/` is the path to the data repository one wishes to process, `--rerun demo` tells the command to output the result of processing into the directory `rerun/demo/` and `--id visit=0230069` tells the command to run the visit (or exposure) number 230069.

Methodology

The primary results of this research are in the form of Python scripts that implement specific features to help with comprehension and streamlining of the data processing pipeline. Unlike many other processing tasks along the pipeline, these are explicitly Python scripts, and not constructed command line tasks in the LSST software stack. This allows for quick initial testing and rapid development, and is accessible and reproducible to those unfamiliar with the complex LSST code base. For the scope and timeline of this project, Python scripts are the most efficacious implementation of the proposed solutions.

The scripts created in this project are open source and can be found on the LSST Github, located at `https://github.com/lsst`. Both are under the `skymap` code repository and compose the branches `tickets/DM-6903` and `tickets/DM-6991`. The changes to `showVisitSkyMap.py` are under the branch `tickets/DM-6903` and the original script `summarizePatches.py` is implemented in the branch `tickets/DM-6991`. `tickets/DM-6903` has been merged with the master branch and incorporated in the LSST code base, while `tickets/DM-6991` remains to be, for reasons outlined in the discussion and conclusion of this report.

As the LSST telescope is not operational yet, data taken from the Dark Energy Camera (DECam) on the Blanco 4-m telescope was used to test and demonstrate the functionality of the scripts. The DECam geometry differs from the LSST camera is a few crucial ways: the 62 CCD sensors are not clustered in rafts, but instead are both the fundamental and only unit that make up the camera (`http://www.ctio.noao.edu/noao/content/dark-energy-camera-decam`). However, DECam exposures suffice for testing the software stack with existing astronomical data.

Results and Deliverables

This section summarizes the construction and use of each of the scripts created. The detailed structure of all classes and packages used in construction of these scripts is not discussed here - for more detail and documentation of all classes and namespaces used in LSST, including those used here, see `https://lsst-web.ncsa.illinois.edu/doxygen/x_masterDoxyDoc/index.html`.

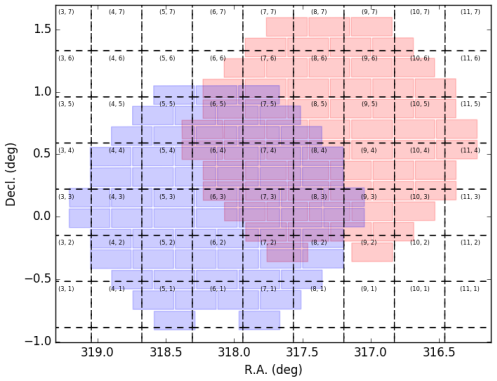

Figure 2: `showVisitSkyMap.py` script output without CCD serial numbers labelled

The script `showVisitSkyMap.py` is used to construct a visual representation and reference of the exposures and their location in a skymap after running `makeDiscreteSkyMap.py` to create skymap data. A simple command line implementation of `showVisitSkyMap.py` with the LSST stack is as follows:

```
python /path_to_skymap/examples/showVisitSkyMap.py
rerun/demo 0 0230069^0232735 --saveFile skymap.png
-p
```

This implementation gives the required inputs: the location of the processed data repository, `rerun/demo`; the tract, `0`; and the visit numbers to be displayed on the skymap, `0230069^0232735`. Additionally, two optional flags were passed as arguments: `--saveFile skymap.png`, which saves a PNG image (Figure 2) as skymap.png, and `-p`, which displays the patches (dashed lines in Figure 2) on the output skymap.

In this project, `showVisitSkyMap.py` was edited to allow for display of CCD serial numbers on the skymap image. This was implemented as a command line flag handed to the command above, `--showCcds`. The results of this new command are shows in Figure 3, which plots visits 230069 and 232735 from the DECam on a skymap with patches visible after running the command:

```
python $SKYMAP_DIR/examples/showVisitSkyMap.py
rerun/demo 0 0230069^0232735 --saveFile
skymap.png -p --showCcds.
```

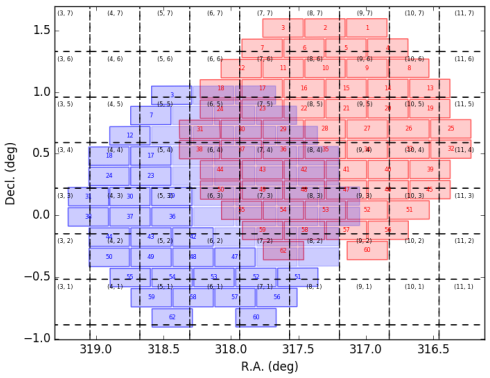

Figure 3: `showVisitSkyMap.py` script output with CCD serial numbers labelled

The implementation of this relies on classes already present in the LSST software stack. Information about visits is initially retrieved using the LSST butler, and then using classes that provide the relevant information, such as camera and CCD classes. The `tickets/DM-6309` branch of the skymap code repository, which contains these edits to `showVisitSkyMap.py`, has been merged with the LSST skymap master branch.

In order to help with large amounts of data with minimal human input with the LSST pipeline, this project created `summarizePatches.py`, to be added to the LSST skymap codes. This script relies on many of the same technical details regarding LSST classes as described for the additions to `showVisitSkyMap.py`. Currently, the LSST pipeline allows command line tasks to run large processing commands using HTCondor, which can be run using Orca orchestration software through `runOrca.py`. The `runOrca.py` command takes a specifically formatted input text file that states

which filters, patches, visits, and CCDs are to be processed. However, in commands further down the pipeline, such as `assembleCoadd.py`, require that the inputs are organized by filter in each patch while also providing visit numbers associated with the patch.

The script created in this project, `summarizePatches.py`, automates the process of sorting each visit and its CCDs into respective filters and patches. For example, an implementation of `summarizePatches.py` might look like:

```
python /path_to_skymap/examples/summarizePatches.py rerun/demo --ccdKey ccdnum
--saveOverview --saveSummaries orca.input
```

The (required) input `rerun/demo` and flag `--ccdKey ccdnum` are identical to their use in `showVisitSkyMap.py`. Depending on the context, there are two types of organization of information that is needed: information grouped by filter in each patch or all information grouped by patch. These two varieties are created with the passing of the `--saveSummaries filename` flag, which will create two files: `condensed_filename` and `expanded_filename`. In the above example, the two files created will be `condensed_orca.input` and `expanded_orca.input`, with lines as shown in Figure 4.

```
condensed_orca.input[41]:
   filter=i^g tract=0 patch=8,5
expanded_orca.input[64:65]:
   filter=i tract=0 patch=8,5 --selectId visit=232735
   filter=g tract=0 patch=8,5 --selectId visit=230069^229341
```

Figure 4: Sample lines from `summarizePatches.py` summary output

```
patchesOverview[530:547]:
   patch=(8, 5)
   {'i': [232735], 'g': [230069, 229341]}
   visit=232735
      ccd=8
      ccd=4
   visit=230069
      ccd=29
      ccd=16
      ccd=36
      ccd=22
      ccd=28
      ccd=35
      ccd=15
      ccd=21
   visit=229341
      ccd=25
      ccd=32
      ccd=39
```

Figure 5: Sample data for patch (8,5) from `summarizePatches.py` overview output

The above implementation of `summarizePatches.py` includes the option flag `--saveOverview`, which saves a human-readable summary file of which filters, visits, and CCDs compose the data in each patch named `patchesOverview`. This allows for easy navigation and interpretation of data through tools such as the Unix `grep` command to find specific visits or patches. Figure 5 shows some sample lines from the `patchesOverview` file from the above implementation, and correlates to the patch presented in Figure 4. While this file is not formatted in a way that allows for it to be fed into a future command in the pipeline, it offers a more accessible overview of each patch.

The `summarizePatches.py` script also allows users to pass flags to the command in order to process specific visits (for example `--visits 0230069^0232735`). However, much of the usefulness of the script comes from the feature that if no visits are passed to the script, it will summarize all of the data that it can find in the input repository, allowing users to quickly create these files without investigating which (or even how many) visits are involved.

Discussion

The deliverables from this project are aimed solving issues in pipeline accessibility and ability to process large amounts of data. The results offer new methods to address these issues in focused areas of the pipeline, but are rather limited in their generalizability to the rest of the code base.

The outlined implementation of `showVisitSkyMap.py` has been merged with the LSST code base as the branch `tickets/DM-6903`, and will be used by future researchers and developers in

their use of the LSST stack. It succeeds in making a specific piece of metadata readily accessible to users of the LSST pipeline by placing the CCD serial numbers on CCD outlines in skymap images, and makes the data both visual and understandable to those outside of the LSST team. The edits fit naturally within the existing structure of the code - the option to show the CCD serial numbers is called in the same way as all other options for `showVisitSkyMap.py`. Additionally, the flag itself makes sense to outside users; `--showCccds` shows the CCD numbers on the skymap.

However, the scope of the edits to `showVisitSkyMap.py` is limited. The implementation of the `--showCcds` option is highly specific to this script and takes advantage of the current structure and information used in creation of the skymap; these changes are not generalizable to an arbitrary piece of metadata that a user might wish to access. For future work, it could be advantageous to developers and users of LSST software to create a command line tool that can retrieve requested metadata in a given repository.

The `summarizePatches.py` script is aimed at helping run a large number of visits through the pipeline with little to no need for a user to have information regarding the specifics of each visit - an important step for automation of the pipeline. Additionally, it does so in a way that fits with other implementation of pipeline utilities using scripts and takes advantage of existing classes in the LSST stack. However, it is a script and not a command line task, which limits its overall effectiveness. Implementing `summarizePatches.py` as a command line task would get rid of hacks currently in the script that allow for parsing of visit numbers. Also, using a command line task would fit more with the flow of the pipeline - with most of the major tasks implemented as such. It is possible that this script can be used as a tool for current developers, and be merged into a command line task in the future.

Overall, scripts and practices like `summarizePatches.py` can streamline the pipeline and make its easier to use. Further work in this should continue to make scripts that allow for more automation of the pipeline with little need for the users of LSST software to help guide data processing. When LSST begins to produce 15 TB of data every week, the only way to process this large amount of data will be through automated programs that are similar to and generalize `summarizePatches.py`.

Overall, this project succeeded in proposing and implementing targeted fixes to issues that currently exist in the LSST software. As development of the LSST code base continues in the years before the telescope is operational, the results of this project will assist future developers and astronomers, as well as offer areas for improvement and accessibility of LSST software.

Acknowledgements

I would like to thank Hsin-Fang Chiang and the LSST team who have mentored and guided me through my internship this summer. This report is written to fulfill the National Center for Supercomputing Applications' Students Pushing Innovation (SPIN) internship requirements, and has my permission to be made public by the SPIN program.

References

[1] M. Jurić, J. Kantor, K. Lim, R. H. Lupton, G. Dubois-Felsmann, T. Jenness, T. S. Axelrod, J. Aleksić, R. A. Allsman, Y. AlSayyad, J. Alt, R. Armstrong, J. Basney, A. C. Becker, J. Becla, S. J. Bickerton, R. Biswas, J. Bosch, D. Boutigny, M. Carrasco Kind, D. R. Ciardi, A. J. Connolly, S. F. Daniel, G. E. Daues, F. Economou, H.-F. Chiang, A. Fausti, M. Fisher-Levine, D. M. Freemon, P. Gee, P. Gris, F. Hernandez, J. Hoblitt, Ž. Ivezić, F. Jammes, D. Jevremović, R. L. Jones, J. Bryce Kalmbach, V. P. Kasliwal, K. S. Krughoff, D. Lang, J. Lurie, N. B. Lust,

F. Mullally, L. A. MacArthur, P. Melchior, J. Moeyens, D. L. Nidever, R. Owen, J. K Parejko, J. M. Peterson, D. Petravick, S. R. Pietrowicz, P. A. Price, D. J. Reiss, R. A. Shaw, J. Sick, C. T. Slater, M. A. Strauss, I. S. Sullivan, J. D. Swinbank, S. Van Dyk, V. Vujčić, A. Withers, P. Yoachim, and f. t. LSST Project. The LSST Data Management System. *ArXiv e-prints*, December 2015.

[2] LSST Science Collaboration, P. A. Abell, J. Allison, S. F. Anderson, J. R. Andrew, J. R. P. Angel, L. Armus, D. Arnett, S. J. Asztalos, T. S. Axelrod, and et al. LSST Science Book, Version 2.0. *ArXiv e-prints*, December 2009.