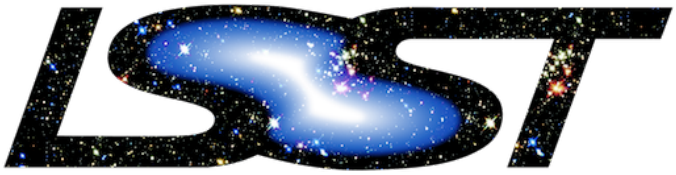

LARGE SYNOPTIC SURVEY TELESCOPE

Large Synoptic Survey Telescope (LSST)
Data Management

Image Display WG

Yusra AlSayyad (chair), Scott Daniel, Gregory Dubois-Felsmann, Simon Krughoff, Lauren A. MacArthur, John Swinbank, Chris Waters

DMTN-126

Latest Revision: 2020-02-19

Abstract

This document describes the findings of the LSST DM Image Display Working Group.

Change Record

Version	Date	Description	Owner name
1	YYYY-MM-DD	Unreleased.	Yusra AlSayyad

Document source location: `https://github.com/lsst-dm/dmtn-126`

Contents

Image Display WG

1 Introduction

The QA Working Group (QAWG) was formed in mid-calendar-2017, with a wide-ranging remit to suggest improvements to processes and tools used for quality assurance across the Data Management subsystem, following the charge defined in LDM-622. The QAWG produced an extensive report (DMTN-085), which touched on issues in the area of image display and visualization. However, due to uncertainty around project scope and the future of development of image display tools within DM, the QAWG did not issue effective recommendations in this area.

The Image Display Working Group (IDWG) was constituted in summer 2019 with the primary goal of rectifying this shortcoming, as well as issuing recommendations on image display for the purposes of commissioning (which fell outside the QAWG scope) and considering whether useful recommendations can be made about tool development in support of diagnostic work within the Camera Subsystem. The detailed terms of the Working Group are provided in LDM-702.

2 Approach

The IDWG identified three ways to approach its charge.

1. We considered the various contexts or environments in which a scientist or developer might wish to view images, and established how well these are served by current tools. For example, these environments include a developer working on a standalone laptop, or a scientist using the LSST Science Platform. This analysis is presented in §3.
2. We surveyed groups of key stakeholders across the LSST Project, including DM pipeline developers, members of the Commissioning Team, and members of the Camera Team, to understand what they regard as the most important use cases for visualization, and how well those use cases are served by existing tools. This analysis is presented in §4.
3. We identified existing tools — including those developed within DM, those produced by other subsystems within LSST, and those available in the wider community — to assess

their capabilities and relevance to the key use cases which have been identified. This analysis is presented in §5.

Finally, in §6 we present conclusions and recommendations drawn from the above.

3 Environments

There are different environments in which a DM use would view images. Different tools are appropriate for different environments.

- **Laptop:** Have complete freedom of tools, for small amounts of data copied over
- **Project dev server (`lsst-dev`)**: Freedom of tools with image data disks mounted and available. X Windows forwarding.
- **Science Platform:** No X windows, need tools for JupyterLab and web.

Therefore it is necessary to specify the environments associated with each recommendation. We focused recommendations on the lsst-dev and especially the Science Platform environments, for which the image display features are only partially implemented.

DM requested the following essential features. Features are some action users want to be able to complete or some characteristic of the image viewer. Use cases are the commissioning, science validation, pipelines writing tasks that prompt these actions. Some of these features are currently possible and easy on lsst-dev and Nublado; some are not. We distinguish between “possible” and “easy.”

4 Features and User Stories

4.1 All-Sky Visualization

High on the priority list for surveys (such as LSST) with a large footprint on the sky is a tool enabling a visualization of the (to-be) observed area on the full sky covered over some time

period (e.g. per night, survey-to-date). For example, one might like to overlay the instrument footprint of all the single-epoch images accumulated to-date or in a given time period and be able to visualize their on-sky RA/Dec positions with respect to common celestial objects/reference points (e.g. the Milky Way galaxy, the Solar System and its members, constellations, Messier objects, other surveys, etc.)

Specifically, this could include functionality to both:

- display a nominal focal plane footprint at any desired location on the sky, e.g., for use in understanding planned observations; and
- display over a coadd the outlines of the single-epoch images used to generate the coadd, or of all the single-epoch images that contain a given source.

It would also be desirable to be able to display information associated with the footprints of interest (e.g. single-epoch statistics, MAF metrics). This would include quantities that are evaluated across the entire (LSST-observed) sky and it would be desirable to have the ability to explore them at a range of scales; from all-sky down to the finest-grained level at which they are defined. It was noted that this becomes increasingly important in the context of LSSTCam and particularly the Science Validation surveys when larger contiguous regions of sky coverage start to emerge.

At the time of writing, such a capability does not exist specifically for LSST, although some of the desired functionality does overlap with those provided by Firefly (see, e.g. (Dubois-Felsmann et al., 2016)). The hscMap viewing tool developed for the HSC-SSP (see §5.5, `https://hscmap.mtk.nao.ac.jp/hscMap4`) provides some of this functionality for all-sky survey footprint visualization.

4.2 Full Focal Plane Visualization

The Commissioning, Camera and Data Release Production teams expressed a requirement for visualizing the full focal plane. Key aspects of this requirement include:

- A “zoomed out” overview of the focal plane should be displayed;
- The user can select and zoom in to particular areas of interest;

- As the user zooms, data is loaded at progressively higher levels of resolution;
- Ultimately, at a high enough zoom level, the user can examine the values of individual pixels.

Note that this functionality is only of interest if it is accompanied by the ability to overplot mask planes on the data (§4.6).

Two tools are currently under development in the LSST ecosystem which may address this requirement: the Camera Image Viewer (§5.4) and ExpViewer (§5.7.1). Note, however, that both of these currently rely on compressed image data, so it may not be possible to use them to accurately recover the values of individual pixels. Furthermore, loading and transmitting full focal plane images is expensive in terms of I/O bandwidth.

Third party packages which can address image visualization at the scale of an LSST focal plane exist. However, these involve resampling images onto a particular sky tessellation. For example, the Hierarchical Progressive Survey (HiPS) standard (Fernique et al., 2017) requires that data be resampled on to the HEALPix (Górski et al., 2005). This makes it impossible to recover individual LSST pixels from the resampled data.

4.3 Mark and save positions on images

Define regions of interest.

Use Case: Crowd Sourced Image Inspection Inspect and markup images during commissioning in order to find surfaces that need black paint (analogous to the des-exp-checker).

A user story told we heard many times was of image review of DES images during comission-ing.

4.4 Individual Image Inspection

Independent of all other recommendations, there is still strong interest in the ability to, outside of any notebook or JupyterLab-like environment, examine individual raw images in detail using DS9 or some equivalent tool. Users are agnostic as to how this tool is provided (i.e.

whether it is served through a web browser or through some kind of terminal forwarding in the vein of xpra). The principal concerns are that it be able to rapidly load and manipulate pixel level data and that it be able to operate in a server-client mode so that users do not have to download terabytes of image data onto their local machines.

4.5 Compare Images

All groups polled requested the ability to directly compare two images, by locking WCS or pixel coordinates, scale and limits, and both blinking and side by side. After locking users want to be able to pan and zoom while viewing side-by-side or blinking.

Two groups also requested a "crossfade" option as "nice to have" rather than essential. Crossfade means that two images are overlaid at the same time, and the user moves a slider to shift the relative weighting of each image. Ideas's put forth include also include using a movable "curtain" UI element to wipe one image across the other. Most, if not all, uses cases are fulfilled by a simple blink.

Using DS9 on `lsst-dev` this is possible and easy (for frequent users of DS9). Using Firefly or matplotlib on Nublado, it may be possible, but it is not easy.

User stories

- As a pipelines developer, I want to be able to blink two locked images so that I can assess the effect of an algorithm modification.

4.6 Overlay Maps/Masks

Commissioning, AP, and SST specifically requested the ability to layer maps on images with varying degrees of transparency and multiple colors. This will be used to overlay the masks (detected, saturated, interpolated, etc.) from the LSST Science Pipelines as well as maps of known defects on top of images during visual inspection. The functionality should be provided through a realtime UI for use when inspecting an image "by hand" and through a programmatic API for use when inspecting an image in a JupyterLab/notebook environment. `afwDisplay`/DS9 and Firefly both currently support this functionality.

Science Pipelines users will view masks in order to inspect full focal plane defect overlays. They are already users of Firefly's capability to toggle individual mask planes and choose colors. Science Pipelines users want NaNs made obvious in another color, in order to make image processing problems obvious sooner rather than later.

The comissioning team also requested colorized *image* overlays on greyscale images (with tunable alpha)/ This refers to the rendering of single-channel image data with a hue or a partially transparent color overlay based on an additional channel (e.g., colorizing a single-channel flux image by the per-pixel variance).

4.7 Callbacks (Run arbitrary code on a pixel or region)

Users requested the ability to run arbitrary python code on a pixel or selected region.

Callbacks are important because they give the user the ability to add any feature to the image viewer. This makes the extant features available in any particular viewer less important.

This would also enable simple per-source drill down. The comissioning team stated that "It would be very nice to be able to click on a source (or collection of sources) in a pixel-level image and have callback to the notebook aspect to run further analysis on those sources"

4.7.1 Extensible displays

Everything must be scriptable. Science Pipelines puts a high priority on having APIs for all functionality.

Because the set of interesting specific visualizations far exceeds what could be provided centrally, make it possible for users to define visualizations, constructed from the lower-level visualization capabilities in the system (or an external library), in a way that makes them straightforward to apply on demand.

4.8 Snappy User experience

User experience: There is a big difference between a visualization platform being able to do something and having it be easy or responsive enough to not cause an extra barrier to usage.

A few specific examples of functionality that needs to be gotten right or the tool will suffer from users finding ways to use other tools:

- **Trivial adjustment of range and zero point of image scaling**: It should be a one click (or hotkey) operation to get a good guess at the scaling range (e.g. z-scale). Dynamic scaling needs to be achieved through intuitive gestures. Example: When looking at raw frames, the bias has not yet been taken out. This means each amp needs a different scaling. When switching between amps, it must be trivial to adjust the scale parameters via the mouse so that a reasonable set of scale parameters can easily be found (the implementation in ds9 is a well known reference for this).
- **Pan and zoom must be desktop-like**: Even a small delay between a drag and the frame panning causes momentary disorientation. Similarly, if the zoom is not close to continuous and instantaneous, the user must reorient after every change in zoom. Example: It is common to be at a fairly high zoom level and want to pan around to find a specific type of object or feature. If the pan takes any fraction of a second, the user needs to figure out where they are so they know where to pan again. Similarly, one may want to zoom out to find other examples. If the zoom isn't smooth, it requires cognitive effort to figure out a) when the right zoom has been reached and b) where to pan after the zoom.
- **Mouse over values need to update with ultra-low latency**: The value of the pixel under the mouse needs to be displayed prominently. The value must update with ultra-low latency as the mouse is moved over the image. Example 1: It is common to want to explore pixel values to get an average by eye of the background. Just by running the mouse over pixels between objects, it is simple to get a sense of the background value if the mouse over value updates essentially continuously. Example 2: It is common to want to try to find the max pixel value in an object. Rather than drawing a trace, it is often sufficient to run the mouse over the pixels of the object to look for a gradient and follow it up. This is much harder to do if it takes significant time to update the mouse over value.
- **It needs to be simple to match multiple images on either image coordinates or WCS coordinates**: Given a reference image, snap all other images in other frames to the same image or sky (configurable) coordinates and zoom level with less than three mouse clicks. Once co-registered, pan and zoom should be linked in all panes. This includes the ability to cycle through the frames in the same pane, i.e. "blinking" through the frames

sequentially via a blink selection in a menu option or hotkey. I.e. not sequentially clicking through a series of tabs or list entries. Example: There are lots of moving objects in images of any depth. To find them load three epochs of the same area separated by a few days. Register all images to the first image's sky coordinates. Stack the images and in one pane cycle through them automatically with a delay of 0.5 seconds. Look for points sources that follow a linear trajectory relative to the surrounding stationary point sources.

4.9 DS9-style "smoothing" (on the fly convolutions)

Both the Camera Team and Science Pipelines Team are heavy users of DS9's "smoothing" function.

Science Pipelines users smooth images in order to see the sky backgrounds which are usually drowned out by the noise in the original images.

4.10 Mouseover Effects

The ability to have a (selectable) set of values and properties displayed on mouseover of a given position is highly desirable. Examples include (but are certainly not limited to):

- Image information: id, observation date & pointing, type, exposure time, etc.;
- Position of mouse in WCS (RA/Dec) and image (pixel) coords;
- Pixel value under mouse;
- Pixel S/N under mouse (assuming a variance plane is available).

4.11 Dynamic stretch

Camera team visualizes raw images (still with overscan regions or amp-to-amp offset). Interactive (and responsive) modification of color stretch parameters.

User story: As a camera team developer, I want to be able to change the scale limits of two side-by-side images by hand, so that I can see the overscan.

4.12 Display footprints and heavy footprints

Display of LSST data objects with natural on-image interpretations: (Heavy)Footprints, PSF models at a point, PSF variation models or measurements Footprints should be clickable to facilitate investigation into deblending computations.

Be able to viz intrinsic model (not PSF convolved) (current: matplotlib) Subtract models from images. Be able to flip from image, model, residuals. (current: matplotlib)

Pixel-level images with source models subtracted to see visualize residuals with respect to the model Could be useful for large galaxies, bright stars, evaluating deblending CFC: Could be useful for stray and scattered light analysis if the source model include ghost images from internal refections of bright sources.

4.13 Overplot circles/ellipses/footprints

Suggested pixel-level image visualization requirements Ability to overlay sources, along with their model ellipses; must be able to use multiple projections, e.g., instrument, sky; must be able to pan and zoom and adjust color scale quickly; must be able to overlay masks; must be able to see map value and coordinates at mouse location; a ”magnifying glass” / zoom-in inset similar to ds9 would be nice to have CFC: This should be merged with use case 7. (LPG +1) Overlay sources from other surveys (current: hscMap for HSC, ???? for LSST) Overlay source catalog with brushing and linking (current: Firefly. Though not everyone knows how to use this.)

4.14 Miscellaneous

- header viewer
- pdf, png, jpeg output
- Viewer in separate window (not just separate tab)

4.15 Miscellaneous requests not specific to images

These are catalog visualization requests that augment image display, but aren't specific to displaying images in a technical sense.

4.15.1 Brushing and linking

Across images, histograms, and scatterplots (on the board these pairs of those were called out: image-hist, image-scatter, hist-scatter, but hist-hist, scatter-scatter, and so on also make sense). Support pairwise scatterplots (e.g., displays of the matrix of all x-y plots derivable from a set of N variables). KB: Bokeh / holoviews / datashader offers much of this capability, for example CFC: PyVis tools does this. KB: Brushing and linking between matplotlib plots and python objects in a notebook is definitely important. In addition, it would be really useful to be able to examine an image (e.g., in Firefly), overlay catalog objects, select objects of interest, and then be able to access that selection from a notebook. In other words, I am asking for a linkage between pixel-level images, object catalogs, and the notebook aspect.

4.15.2 Catalog Integration

An obvious desire for a viewer associated with images with which any type of object catalog can be associated (e.g. catalogs specifically created for the image through image processing pipelines, external reference catalogs, etc.) would be the ability to overlay the catalogs on the image itself. There should be a two-way ability to click on a source in an image or catalog and get associated catalog-level data or zoom to associated position on the image, respectively. Hovering the mouse on a source should pop up useful metadata about that source.

Sub-selections of data points should be possible via mouse-clicking tools (e.g. single and shift clicking, rectangular and other geometric shapes, "roping") or an SQL-like querry to the database serving the catalog. There should be numerous operations that can be executed on any seleciton of data points:

- produce scatter plots and histograms of any columns in the catalog;
- produce quiver plots when field vector values are provided as columns in the catalog (particularly useful in assessing calibration data, astrometric residuals, PSF moment residuals, engineering facilities measurements, etc.);

- basic statistical calculations of the sample properties;
- highlighting of sources with certain catalog-based characteristics (e.g. S/N threshold, flags set, etc.);
- ability to save/reload any sub-selection as an external catalog.

4.15.3 Per-Source/Position Flipbooks

In most cases when viewing and assessing a given image, there also exists myriad image/template types corresponding to the image of interest (and each other) in a spatial sense. An obvious example would be the set of individual input images contributing to a given coadd image. The ability to produce and display on-the-fly various flavors of a "flipbook" consisting of cutouts centered on a given source or RA/Dec from all the images related in some way to the source/position of inerest would be a great asset. Such a flipbook would be particularly useful in the context of time-variable phenomena such as supernova (lightcurves), moving objects (trajectories), and image quality variations (e.g. seeing variability, cosmic-ray and other unmasked/interpolated cosmetic defects, high levels of extinction, etc.) Per-source flipbooks are understood to be temporal in nature. An automated (but also set-able) definition of cutout dimensions could be based on source properties (e.g. based on size or detection footprint, possibly of the parent for blended sources).

One could also imagine the creation of flipbooks for various categories of flux images (raw, post-ISR, calibrated, etc.) to visualize the effects of the various calibration phases. Flipbooks of source models computed for each epoch would also be informative. In the aid of model fitting characterization and difference imaging diagnoses in particular, flipbooks of image and model residuals (individual and/or coadd image minus templates, image-to-model residuals, where the model could be per-epoch or time-integrated) would be an invaluable quality assessment tool.

5 Existing Tools

5.1 Firefly

Firefly has been the default fits viewer for the notebook environment.

It has more features than any other tool evaluated.

It already has a robust `lsst.afw.display` and is used in pipelines tutorials.

It's easy to imagine adding a wide variety of new features to Firefly at relatively modest cost, especially, ones that mainly involve writing more 'afw'-oriented Python interfaces to existing features. The sticking point is that there are some performance-related issues that are inherent to the client-server architecture, and that would require significant engineering to address (e.g., by moving more of the data and computation to the client, or by doing more cacheing of likely-to-be-requested data). That engineering is not inconceivably difficult but it may be a disproportionate amount of work compared to the perceived near-term benefit.

In some cases this is behind the difficulty in addressing what may at first sight appear to be simple UI issues. However, implementing some UI features would not yield a good user experience, because the system is not designed to handle the high rates of actions that would produce. One feature for example, connecting the mouse scroll wheel UI events to the zoom-in and zoom-out actions, would be trivial to implement but would yield a slower experience.

Developers had been gradually making changes to improve the responsiveness of the UI by moving work from the server to the client - for instance, rotations are now done client-side and are really fast. However, these big system changes are not driven strongly by Firefly's other IPAC-side applications, and continuing this work that is probably not realistically doable within the post-descope constraints.

5.2 Ginga (+Astrowidgets)

Ginga and Astrowidgets both attempt to supply toolkits for image display, instead of a full-featured program like DS9. The standalone ginga program is referred to in the documentation as a "reference viewer," which displays an example interface that can be built. The API does not have communication abilities outside the scope of the program, so there is no DS9/XPA like interface to either the reference viewer or to any notebook based viewer. The stack includes 'lsst.afw.display.ginga', which in a notebook environment can communicate with the viewer in that environment, but the lack of inter-process communication prevents 'lsst.afw.display.ginga' from functioning as a debug display for command-line tasks.

The Ginga viewer supports directly loading FITS files as well as arbitrary numpy arrays of data.

The keyboard and mouse handling is enabled when the viewer is constructed in a notebook, and is connected to simple image operations: zooming, panning, image scaling, contrast, colormaps, axes flips, and rotation. There are python API callbacks to all of these, with "get" and "set" methods. Astropy regions are supported, and likewise have "add" and "delete" methods to the internal catalog. Headers and WCS objects are supported, with 'lsst.afw.display.ginga' having a WCS translator to map between the LSST and ginga interfaces. Mask overlays are supported, although not with a simple API.

Astrowidgets is an additional layer of helper code on top of the Ginga viewer, supplying a friendlier interface to the viewer while also connecting more explicitly to the ipywidgets widgets. This interface can be used to construct a DS9-like experience with buttons, sliders, and text boxes to control the viewer. The tutorial example adds a X/Y/RA/DEC/image value box to track mouse positions in real time. Astrowidgets is in early development, so there is not much beyond the wrappers for the Ginga built in code. However, it is easy to work with, as demonstrated by adding a method to load an 'lsst.afw.exposure' object its respective WCS and mask planes. Multiple frames are not cleanly supported with either interface, but the astrowidgets methods can connect an array of images to the viewer via widgets that provide information about the current image being displayed.

5.3 JS9

JS9 (`https://js9.si.edu/` is being actively developed by Eric Mandel as a browser-based FITs viewer that reimplements the DS9 interface.

Evaluation: On initial evaluation, it was hard to tell how to do mask overlays in a manner similar to what we do with DS9. In the interim, Eric has implemented two examples for how to do this and both look viable. See the demo with an HSC image at `https://js9.si.edu/js9/js9mask.html`. JS9 can run in either a monolithic server mode with many users accessing the same endpoint or in a more local manner with each user running their own server. Eric has provided a Docker container that can be used to spin up a server locally: https://hub.docker.com/r/ericmandel/js9server Since docker plays nicely with k8s, it may be a trivial extension to the existing nublado helm charts to also run a js9 pod. There will be performance tradeoffs in terms of data transmission depending on how we use the tool. Using the Python interface should be fast to upload images since the server and prompt are in the same cluster, but shipping FITS files to the user local browser will pay the penalty of uploading the image to the browser. However, that doesn't seem to be any different than shipping an

image to any other local tool.

The `pyjs9` client library wraps a RESTful API exposed by the server. The client library at first glance seems to have the necessary functionality to implement most of the standard `lsst.afw.display` back end.

5.4 LSST Camera Image Viewer

The LSST Camera Team has developed a bespoke image viewing tool to support their activities. As such, it is primarily aimed at examining raw pixel data recorded from the camera to diagnose hardware issues. This viewer is being developed by Tony Johnson (SLAC); much of the material below is adapted from his comments. Given the rapid pace of ongoing development, some of the discussion below may soon (or already) be outdated.

The current version of the Camera Image Viewer is available on the web at `https://lsst-camera-dev.slac.stanford.edu/FITSInfo/`.

5.4.1 Capabilities

Given its audience and goals, it does not provide facilities which are require for astronomical analysis of the data. For example, the viewer has no concept of world coordinate system: it simply operates in focal plane pixel coordinates.

Instead, the viewer is designed to provide high-performance interactive display of images corresponding to a full focal plane or subsections thereof. The development plan prioritizes:

- A web-based interface for selecting and accessing data;
- Scaling from full focal-plane to individual amplifiers, with real-time panning and zooming;
- The ability to select, display and analyze regions by hardware device (e.g. particular CCDs, amplifiers, etc.);
- The ability to perform hardware diagnostic functions on the selected image region;
- The ability to trigger DM algorithms to run on the selected region;

- The ability to track and overlay the values of diagnostic measurements with time.

At time of writing, not all of this functionality is currently available.

5.4.2 Implementation details

The tool consists of two separate pieces (the "browser" and the "viewer"). The former is a relatively simple web front end to the Camera image database which enables the user to select data of interest. The latter is more technically interesting, as it must cope with the technical challenge of making available extremely large quantities of image data to the user in a fast, interactive way.

The viewer is based on the International Image Interoperability Framework[1] (IIIF), which defines a standardized API for making image data available on the web at scale. A primary advantage of this is that existing open-source software is used as the basis of the system. In particular, the Camera Image Viewer is based on:

- Cantaloupe[2], a server-side system for reading image data from storage and providing it to clients using IIIF standards;
- OpenSeadragon[3], a browser-based Javascript library for receiving and displaying image data.

Both of these tools provide for extensive customizability. In particular, Cantaloupe has been extended to read data from the FITS files delivered by the camera, while OpenSeadragon is being extended to provide the various user-facing interaction and analysis facilities which are required.

It is worth noting that, although data is stored on the server in FITS format, it is converted to JPEG for transmission to the client. It is therefore lossily compressed: the original pixel values are not directly available in the image stream.

[1] `https://iiif.io`
[2] `https://cantaloupe-project.github.io`
[3] `https://openseadragon.github.io`

Also note that — regardless of the software system used — loading and displaying full-focal-plane data at a low enough latency to provide a satisfactory interactive experience places a substantial load on the back-end storage system. This will, of course, scale with the number of clients accessing the system, but one should expect to make a significant investment in hardware underlying the image viewer.

5.4.3 Application to DM use cases

The Camera Image Viewer provides an impressively high-performance interactive visualization of the full focal plane in a way that is not easily available through other tooling. As such, it is of obvious value to the DM Subsystem.

However, the are a number of important limitations:

- The viewer does not have any concept of astronomical coordinate systems or of reprojecting data onto the celestial sphere;
- The level of analytic functionality currently available is limited, although more is in development;
- Integration with DM standard display interfaces (`afw.display`, etc) seems challenging;
- The current system is tightly coupled to the raw FITS images being produced by the camera: it would require substantial work to access processed data from a Butler repository.

Deploying the system as it now stands at the Data Facility would require some investment in both hardware, to provide the low-latency data access required, and in development, to make it possible to retrieve and display data from Butler repositories rather than just camera raw FITS files. However, making this investment would provide DM with a critical capability that is not easily available from elsewhere.

5.5 hscMap

hscMap is an image viewer developed at the National Astronomical Observatory of Japan (NAOJ) with a focus on facilitating the exploration of images from the Hyper Suprime-Cam Subaru Strategic Survey (HSC-SSP). The survey aims at a total areal coverage of $\approx$1400 deg^2

of multiband (grizy + 4 narrow-bands) imaging to three depth levels in fields spanning a large fraction of the sky visible from Mauna Kea. As such, a primary motivating factor was the ability to easily and responsively zoom around and into large areas of the full sky down to the near-pixel level. A quick tour of the latest release (`https://hscmap.mtk.nao.ac.jp/hscMap4/app`) which is currently serving the ≈300 deg^2 imaging data from the second public data release (PDR2; `https://hsc-release.mtk.nao.ac.jp/doc/index.php/survey-2/`) confirms that this goal has been achieved. Another salient feature that immediately stands out is the ability to combine and view 3-bands in "true SDSS color" (Lupton et al., 2004) with adjustable scaling (in addition to the basic RGB coloring). A current limitation is a lack of detailed (and searchable) documentation providing a deeper understanding of the mechanics and underlying data structures required. We reached out to the developers for feedback and were directed to what essentially amounts to video (YouTube) demos of selected functionality including: general navigation, overlays including survey specific field names and grids (tracts and patches) as well as common celestial objects, catalog (pre-made in cvs format and containing at least RA and Dec columns) overlays and basic plotting and sub-selecting via a "rope tool" on a scatter plot and SQL-style queries of a database, an image cut-out service, uploading local FITS files and cvs catalogs by simple drag and drop, display of HiPS-based tiling data served by the Strasbourg astronomical Data Center, etc., (see `http://hscmap.mtk.nao.ac.jp/hscMap4/app/help.html`). There is also support for integrating hscMap into JupyterLab notebooks (see `http://hscmap.mtk.nao.ac.jp/hscMap4/jupyterlab-hscmap/docs`)

Despite all these impressive features, we are not currently recommending further investigations into how hscMap could be used as a mainline image viewer for LSST due to a number of (un)known and complicating factors. These include the need for post processing of the image and catalog data to be served up by a database; the details and resources required are not known to us at this time. Additionally, it is not clear what the plans are for long-term development of and support for hscMap, potentially limiting the possibility for feature requests. However, the ability to simply and snappily visualize the whole LSST sky is reason enough to at least keep hscMap in mind when pursuing other toolkits (as recommended in this report).

5.6 Aladinlite

Aladinlite is a tool for viewing images on entire-sky scales that allows users to zoom down to smaller scales defined at the time the images were ingested. It enables this functionality by storing images in the HiPS (Hierarchical Progressive Survey) schema [citation to 2015A&A...578A.114F]. In the HiPS schema, images are sampled onto iteratively more refined healpixel grids. These

resamplings are stored on a central server which allows users connecting to that server to smoothly transition between levels of refinement, down to the original, finest healpixel sampling. While this seems to enable much of the functionality we have recommended, specifically the ability to transition from full focal plane inspection down to single amplifier inspection, the fact that the images must be resampled onto a pixel grid that is defined in sky coordinates means that the raw, sensor-level pixel information is ultimately lost. Aladinlite was designed to enable scientists to explore images at the entire-sky level. It was not meant to enable engineers to inspect pixel data from sensors. While Aladinlite will likely have a role to play in serving LSST image data to the scientific community, it is unlikely to be helpful during the process of commissioning the LSST hardware and software.

5.7 Miscellaneous

5.7.1 ExpViewer

ExpViewer is being developed by a team led by Luiz da Costa in Brazil to provide rapid display of image data received from the camera in the operational era. A preview version is currently available online at `http://expviewer.linea.gov.br`.

ExpViewer is built upon the same International Image Interoperability Framework as the Camera Image viewer (§5.4), and also makes use of OpenSeadragon for in-browser display. However, rather than directly reading data from FITS stored on disk, images are converted to TIFF format when they are ingested by the tool.

ExpViewer and the Camera Image Viewer have a lot in common, but, at time of writing, the Camera Image Viewer provides a wider range of functionality, while ExpViewer provides a simpler and (arguably) more attractive user interface. The WG understands that discussions are ongoing between the ExpViewer and Camera Image Viewer teams to identify commonalities, avoid duplication, and increase collaboration, but outcomes of this are currently unclear.

Given its ability to work with FITS data (albeit in the camera raw specific form) and its rapid pace of development, the WG suggests that the Camera Image Viewer provides a more attractive tool for potential adoption by DM.

5.7.2 WorldWideTelescope

The American Astronomical Society has adopted Microsoft's World Wide Telescope (WWT) It is under active development by Peter Williams et al. at the cfa

It uses a tiling scheme named TOAST. There is a tool named toasty, to convert images to the TOAST tiling scheme.

At Python in Astronomy 2019, Peter Williams displayed a full tract HSC coadd. It took about 15 minutes to process a tract, but could likely be made faster with effort. WWT has potential to addresses the whole sky visualization and survey footprint visualization use case.

A proposal has been submitting to the NSF for funding to better support visualizing full-focal-plane images and full-sky data from telescopes like the LSST.

5.7.3 des-exp-checker

This tool is not an image display tool, and outside the scope of this review. However, it was referenced in a use case. The DES developed a very simple web interface backed with a sqlite database and fits images on a file system. It was used to visually inspect images during comissioning to find systematic problems e.g. surfaces that needed black paint.

We reviewed the DESC instance of it used to review data from the DC2 r2.1i imsim run. Users can label image data with problems (e.g. tree rings) that apply at particular positions. These are then stored to a SQLite database. The masks shown are the actually LSST mask, rendered as opaque color on top of the images, though it was not clear exactly which mask planes are used. It is clearly a special-purpose tool, though is imple enough that it is appealing to use for appropriate tasks.

We could envision using it as a tool by DM to inspect images for evaluating Science Pipelines algorithms. For example, we could use it to develop a training set for classifying real vs. bogus sources in difference images. It would be too much overhead to setup for a single user, but could be used for crowd-sourcing visual inspection of data. This addresses one particular set of use cases, but that set of use cases is highlighted as important in the inputs we've collected. One might deploy it on e.g. LDF Kubernetes Commons.

6 Recommendations

The recommendations are based on an assessment of the requested features, supporting use cases, and existing toolsets evaluated. One currently unsupported feature is large image display. Users across DM and commissioning (section 4.2) will need to see a full-focal plane or full-tract image in its entirety and be able to zoom in to the individual pixels. Of the existing tools evaluated, the LSST Camera Image Viewer was the most impressive in terms of responsiveness in the zooming. The Camera Team reports that Tony Johnson is actively developing the viewer to include additional features. Therefore adopting this will prevent duplication of effort.

However, the Camera Team reports that the viewer owes its responsiveness to a set of dedicated hardware. Therefore, we recommend that DM

Recommendation

Provide a capability for fast display of full-focal-plane images for all data ingested at the LSST Data Facility based on the LSST Camera Image Viewer.

Users across project are most proficient with DS9. Therefore, we recommend that DM

Recommendation

Make ds9 available through commissioning and early operations by either:

- *Maintaining a development server (lsst-dev equivalent) for project staff,*
- *Connecting LSP to local ds9 via SAMP or XPA or*
- *Through WebDAV*

.
The feasibility of these options should be evaluated

Both JS9 and AstroWidgets show promise for the Jupyterlab environment. JS9 is snappy, has the same interface as DS9, mouse gesture stretch, and on the fly smoothing. Ginga/AstroWidgets is snappy, extensible by python programmers, and it is easy to run arbitrary python code on selected pixels. Therefore we recommend that DM

Recommendation

Support at least one of JS9 / AstroWidgets + Ginga on Science Platform.
A bake-off is required to determine which to support

Finally, firefly farther ahead feature-wise. Firefly has a few unique features. These include its intuitive mask plane toggle and that users can open in a window on another monitor (or across the country). Therefore we recommend that DM

Recommendation

Continue to provide support for the Firefly tool, including some level of re-sourcing for servicing emergent feature requests.

A References

References

[DMTN-085], Bellm, E., Chiang, H.F., Fausti, A., et al., 2018, *QA Strategy Working Group Report*, DMTN-085, URL https://dmtn-085.lsst.io, LSST Data Management Technical Note

Dubois-Felsmann, G.P., Goldina, T., Ly, L., et al., 2016, In: American Astronomical Society Meeting Abstracts, vol. 227 of American Astronomical Society Meeting Abstracts, #348.06, doi:10.5281/zenodo.44653, ADS Link

Fernique, P., Allen, M., Boch, T., et al., 2017, HiPS - Hierarchical Progressive Survey Version 1.0, IVOA Recommendation 19 May 2017 (arXiv:1708.09704), ADS Link

Górski, K.M., Hivon, E., Banday, A.J., et al., 2005, ApJ, 622, 759 (arXiv:astro-ph/0409513), doi:10.1086/427976, ADS Link

Lupton, R., Blanton, M.R., Fekete, G., et al., 2004, PASP, 116, 133 (arXiv:astro-ph/0312483), doi:10.1086/382245, ADS Link

[LDM-702], O'Mullane, W., 2019, *Image display working group charge*, LDM-702, URL https://ls.st/LDM-702

[LDM-622], Swinbank, J., 2018, *Data Management QA Strategy Working Group Charge*, LDM-622, URL https://ls.st/LDM-622

B Abbreviations and Definitions

Acronym	Description
AP	Alert Production
API	Application Programming Interface
DEC	Declination
DM	Data Management
DMTN	DM Technical Note
DS9	Deep Space 9 (specific astronomical data visualisation application; SAOImage)
FITS	Flexible Image Transport System
HEALPix	Hierarchical Equal-Area iso-Latitude Pixelisation
HSC	Hyper Suprime-Cam
IPAC	No longer an acronym; science and data center at Caltech
ISR	Instrument Signal Removal
KB	KiloByte
LDF	LSST Data Facility
LDM	LSST Data Management (Document Handle)
LSP	LSST Science Platform
LSST	Large Synoptic Survey Telescope
NAOJ	National Astronomical Observatory of Japan
NSF	National Science Foundation
PSF	Point Spread Function
QA	Quality Assurance
RA	Right Ascension
SAMP	Simple Application Messaging Protocol
SDSS	Sloan Digital Sky Survey
SQL	Structured Query Language
SSP	Solar System Processing
SST	Subsystem Science Team
UI	User Interface

WCS	World Coordinate System
WG	Working Group
deg	degree; unit of angle