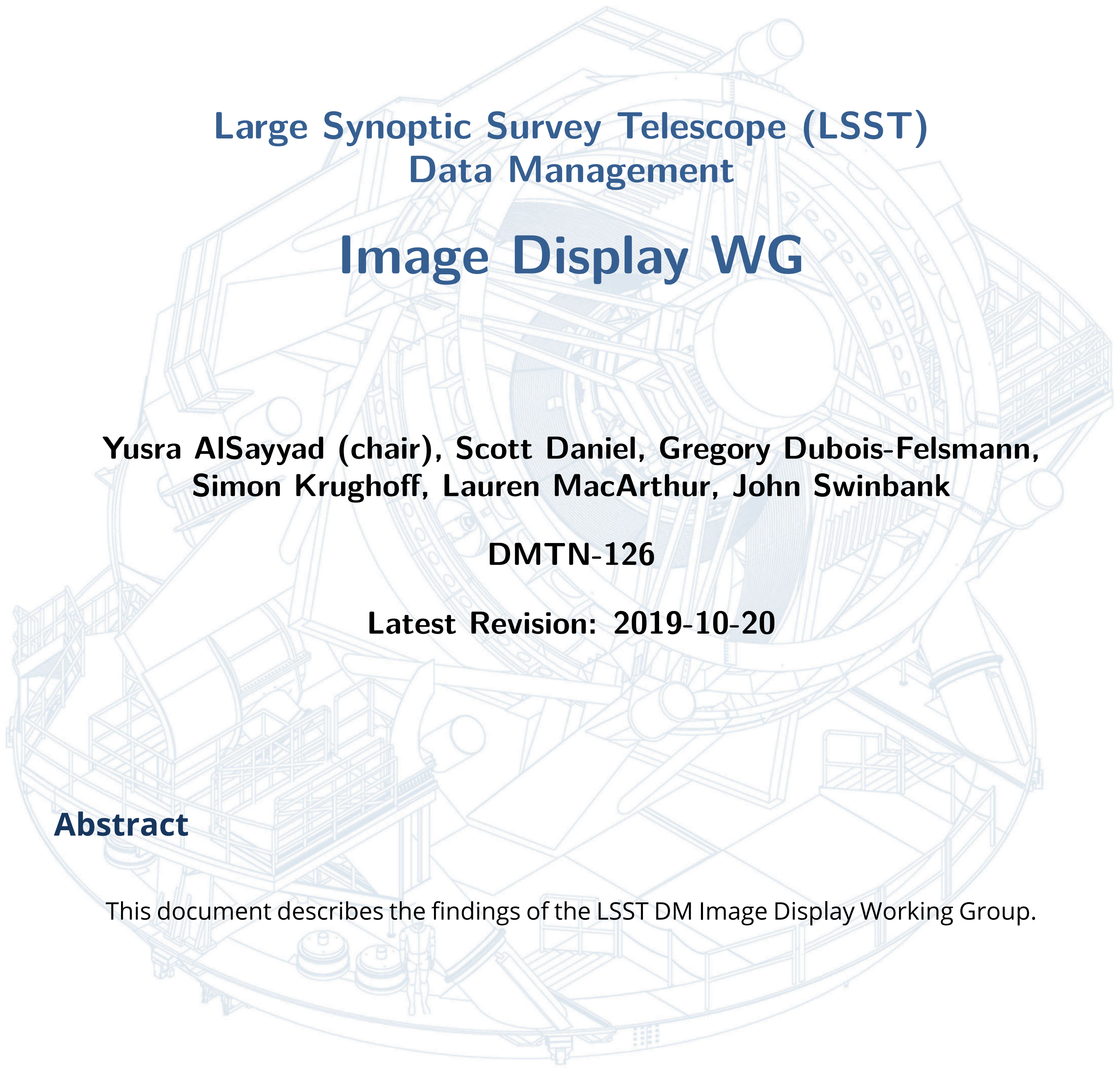

Large Synoptic Survey Telescope (LSST)
Data Management

Image Display WG

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

DMTN-126

Latest Revision: 2019-10-20

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

[YA: TO DO: Simon] There are eight environments in which a DM member would view images.

Simon's table.

Simon's chart in words.

We focused on the lsst-dev and Nublado environments, which are partially implemented.

DM requested these essential features. Features = 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

[YA: As a <type of user>, I want <some feature> so that <some reason>. Is feature request obvious/self-evident? (Yes/No) If no, seek use cases. Can you do it already on lsst-dev? If yes, how? If not, how much work would it be to make it possible? Can you do it already on LSP? If yes, how? If not, how much work would it be to make it possible?]

4.1 All-sky viz

[YA: TO DO: Lauren] Full sky viz for the area covered per night. Overlay on full sky viz. Accumulated N of visits on the full sky. Wished for during ops rehearsal. Where the fields fall in the sky (galaxy, solar system, RA/dec) (current: hscMap for HSC, doesn't exist for LSST) HscMap https://hscmap.mtk.nao.ac.jp/hscMap4

Visualization of all-sky "stuff" (a/k/a statistics, e.g., MAF metrics) This is the display of quantities that are evaluated across the entire (LSST-observed) sky and are desirable to be able to explore at a range of scales from all-sky down to the finest-grained level at which they are defined. KB: This becomes more important with LSSTCam and particularly the SV surveys when we start having larger contiguous regions of sky coverage.

Instrument footprint on all sky (or all-sky) images This could mean 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.

BS: 1 (nominal footprint with some catalog of bright sources i.e. "finding chart") The first of these is implemented already Second - requires work on the table format of for the data. Moving to an obsCore interface. Functionality overlaps portal and firefly

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 (**?**) 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

[YA: TO DO: Yusra]

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)

Showing a particular instance used by DESC for DC2 r2.1i. Can label image data with ?problems? (e.g. tree rings) that apply at particular positions. These are then stored to file (either just as text, or maybe a SQLite database), although it's not clear in exactly what representation. The ?masks? shown are the actually LSST mask, rendered as opaque colour on top of the image. It's not clear exactly which mask planes are used. Nice UI. Clearly a special-purpose tool. Simple enough that it is appealing to use for appropriate tasks. ?Could use this as a simple GalaxyZoo?. How would you stand this up for use by the DM team, e.g. for looking at diffim outputs? Discussed how one might deploy it on e.g. LDF Kubernetes Commons at the T/CAM standup. And authentication options. Not appropriate for use 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.

4.4 Individual Image Inspection

[YA: The title of this feature is vague. Is this the use case supporting maintaining the ability to open images in ds9? Also note that users are most proficient with DS9.] 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

[YA: TO DO: Yusra]

- blinking (animation/movie of images
- side-by-side
- Locking two frames by WCS or pixels coordinates
- Locking two frames by scale and limits
- Pan/Zoom

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. *[YA: Whenever photo editing apps that let you compare the before and after of a photographs with a movable curtain, and I always wish they just let me 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.
- 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.

4.6 Overlay Maps/Masks

[YA: Thank you Scott for the draft. TO DO: Gregory edits]

Commissioning, AP, and SST specifically requested the ability to layer images on top of each other 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.

Overlaying maps onto images (currently use afwDisplay Toggle individual mask planes and choose colors

SST: Individual mask planes (including: across a full focal plane; for coadds) ”Individual” apparently meaning that one or more mask planes to be displayed could be selected programmatically or by UI, with flexible assignment of colors and transparency. Princeton says: need full focal plane defect overlays at the summit mask integration (Firefly does this well) Obvious nans (e.g. in another color)

Colorized 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). Note that mask-overlay display could be treated as an instance of such a capability, but for tracking purposes we’ll treat it separately as mask overlays (in paragraph above) which benefit from specialized UIs/APIs for extracting and naming bits from packed bit masks.

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

[YA: 7/12. TO DO: I'm not sure who is the best person to write this: Gregory John Simon] Robert says that callbacks are important because they give the use the ability to add any feature to the image viewer. This makes the extant features available in any particular viewer less important. Yes, If you write it in a way that you can implement arbitrary code and analysis. Powerful. But I think in JS9 you'd have to write in javascript

Per-source drill down Select a source/object (in a table, over plotted on an image, in an x-y plot) and be able to link to additional visualizations or data displays related to that source/object/ KB: 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. Basically, brushing and linking integration of pixel-level image visualization with the notebook aspect.

4.7.1 Extensible displays

[YA: TO DO: Simon? John?] Everything must be scriptable. We only care that there are APIs for all these things

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

[YA: TO DO: Simon] User experience: There is a big difference between a viz 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 (ds9 is the standard for this, as far as I know).

- 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, 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 to the same sky coordinates. Stack the images and in one pane and 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)

[YA: TO DO: yusra] Camera team are heavy users of ds9’s ”smoothing.” Science Pipelines Team are also heavy users of ds9 smoothing.

4.10 Mouse over WCS/Pixel

[YA: TO DO: Lauren]

Image info. Stats on regions

Position of mouse in WCS and image coords Pixel value under mouse

4.11 Dynamic stretch

[YA: TO DO: Simon]

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

Color scale Use ”Standard” image interactions in a responsive way: scale, stretch, zoom, pan, colormap and basic computation and image analysis capabilities. There was a general desire for stretch/scale changes to be ”fast”. In the meeting we discussed at least three different interpretations of color mapping: rendering of single-channel image data with a color that depends on the flux value (or whatever other variable is represented by the pixel values); rendering of single-channel image data with a hue or a partially transparent color overlay based on an additional channel; generation of color from 3 independent channels. For future purposes we’ll treat this use case as referring only to the first of the three; the other two have their own use cases below. CFC: This needs to include interactive hierarchical capabilities - e.g. display of a full FOV image consisting of 21 science rafts + 4 corner rafts in some definable spatial compression/sampling scheme (e.g. box average, max value, median filter etc..) to make the full FOV image manageable. Interactive feature need to include selecting a specific science or corner raft for more detailed display, followed by selecting a given sensor on a raft for pixel level display and computational (IRAF imexamine like) operations. Other use cases listed here can be implemented through this one. BS: 1 (ds9+imexam functionality)

4.12 Display footprints and heavy footprints

[YA: TO DO: Gregory] 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

[YA: TO DO: Gregory] 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

[YA: TO DO: Gregory] 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

[YA: TO DO: Lauren] overlay catalogs (markers with additional information beyond coordinates)

Selection of catalogs in the image viewer Ability to click on a source/pixel in an image and get associated catalog-level data Plot data about sources from a catalog on top of an image Hover over things and learn fun metadata facts e.g. anything from the visitInfo about an image or anything from an associated catalog/database about a source (intentionally left this vague because I'm sure different people want different things)

Turn on and off detection entries in tables, plots based on flags (i.e. with checkboxes) Applies to any LSST catalog entry with packed Boolean flag fields, including Source, Object, ... These flag selections are meant to participate in whatever more general brushing and linking environments are provided - e.g., to permit sources/objects with designated flags to be shown or hidden in, say, a color-color plot or in an overlay over an image.

4.15.3 Flipbooks per source

[YA: TO DO: Lauren] Thumbnails across time (lightcurves), surveys (current: matplotlib Quick cutouts of warp, to look at input images that went into a coadd, Build mosaics of many images/dataIds on the fly Collections of thumbnails Meant to be around a designated set of objects or sources (if sources, this could be a time-dependent selection of apparitions of the same object).

Flip-books per source Understood to be temporal in nature. Display of image cutout flip-books for selected sources. Automatic definition of default cutout dimensions based on source properties (e.g., smaller for point like sources, larger for extended sources, perhaps including the full parent footprint for deblended sources). Flip books for (raw, calibrated, etc.) flux images; for source models computed for each epoch; for residuals (e.g. from difference imaging vs. templates, or image-to-model residuals, where the model could be per-epoch or time-integrated). BS: 2 (definitely needed for difference imaging diagnostics)

4.15.4 Overlay Vector Fields

[YA: TO DO: Lauren] Overlay a vector field, both for the CBP data, and for people doing astrometric matching or looking at distortions. Vector field plots (e.g., astrometric residuals,PSF moments or other technical information from the EFD) The scale factor must be adjustable. KB: matplotlib quiver plots, for example? We definitely need something along those lines for astrometric residuals and PSF characterization CFC: This seems like something that a visualization would be used to display, but the computation of the vector/quivers is external to the display tool.

5 Existing Tools

[YA: Can I imagine writing an `afwDisplay` *for this tool? Do they fulfill any use cases]*

5.1 Firefly

[YA: TO DO: Gregory]

Gregory says on Slack: It's easy to imagine adding a wide variety of new features to Firefly at relatively modest cost, especially, as I mentioned in the meeting last week, ones that mainly involve writing more 'afw'-oriented Python interfaces to existing features. The sticking point we have, and what was behind my question about acceptability to the pipeline developer community last week, 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 - Trey thinks about this kind of stuff all the time - but it may be a disproportionate amount of work compared to the perceived near-term benefit. (edited) In some cases this is behind the difficulty in addressing what may at first sight appear to be simple UI issues. Merlin asked me, for instance, if we could connect the mouse scroll wheel UI events to the zoom-in and zoom-out actions. Just doing that is (I assume - I'm not a hard-core Javascripter yet) nearly trivial, but it would probably not yield a good UX, because the system is not designed to handle the high rates of actions that would produce. So Merlin's take on the situation is that very fluid interactive response to zooming - and to rescaling - are more important to him than "features" like the mask-plane overlay controls (which he still really likes), and that as a result he's likely to prefer staying with DS9 in preference to starting to use Firefly in his daily work.

(To re-emphasize another earlier point: this is, and should be, totally OK, a choice up to individual developers, and a key motivation to have a "neutral" API like 'afw.display' in the system. We're not trying to mandate tool use, but to decide where we can get the greatest net benefit to the user community from any additional LSST-funded tool development.) (edited)

We have 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 - but this is "big" work that is probably not realistically doable within the post-descope constraints - and also is not driven strongly by Firefly's other IPAC-side applications. (edited) From the WG point of view, then, I think we need to understand how this "fluid response? issue rates in the priorities of the user communities we have. That will then help us assess where best to invest resources. (edited) (end of essay)

5.2 Ginga (+Astrowidgets

[YA: TO DO: Chris or Yusra] Things Chris learned at Python in Astronomy, plus a day's worth of experimenting. Have not yet got `display_ginga` working. Chris has written a quick (1 hour

effort) GUI for interacting with the Ginga display in a notebook; looks pretty slick! Can display mask planes, by adopting code from (but not integrating with) `display_ginga`. Add regions by clicking on the image; get the results through Python callbacks. Understands FITS headers; can access them. Which can actually be populated by metadata provided by the code; it doesn't have to be header. Not clear what the semantics of this should be. One could imagine extending afwDisplay with some generic notion of image metadata. There are handy keybindings. AstroWidgets acts largely as a wrapper around Ginga. Chris is working on a way of panning through multiple frames. Continuing development of AstroWidgets; they claim to now support all of the DS9 region types.

5.3 JS9

[YA: TO DO: Simon] Eric Mandel actively developing. Hard to do mask overlays, due to fundamental differences between how DS9 and JS9 handle multiple images in the same app. Not clear what the multi-client environment is like ? one monolithic server with many clients, or each client spawning its own server? But note that there may be constraints on this because the browser and the python interpreter would need to have access to the same environment. As far as we understand at the moment, matching the sort of mask overlay that we currently have in Firefly/DS9 isn't possible. But that may just be a matter of not fully understanding JS9's capabilities. The pyjs9 client library wraps a RESTful API exposed by the server.

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:

[1] `https://iiif.io`

- 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.

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;

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

- 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

[YA: TO DO: Lauren] Asked for feedback from Subaru/HSC to see if they have any inputs. They sent some documentation videos! http://hscmap.mtk.nao.ac.jp/hscMap4/app/help.html http://hscmap.mtk. hscmap/docs/ https://hscmap.mtk.nao.ac.jp/hscMap4/app/ But no real user feedback. The data is postprocessed before being fed to HscMap. Which helps explain the speed. “Full sky almost to pixel level”. Lupton-sanctioned colours! Can change colormap of pre-computed images, not on the original data. “The algorithm is applied post-stretch”. Interfaces with a database for catalog queries. Can add your own images and CSV format catalogs. Would be really useful to simply visualize the whole LSST sky. Can be incorporated into JupyterLab. But it’s not clear how deep this integration goes. Suspect there is no mask overlay support. Chief selling-points are speed and colours, together with ability to overplot objects from a database. We do not know what the backing format is; we know it used not to be HiPS, but that may have changed.

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

[YA: TO DO: yusra] Peter Williams / WorldWideTelescope integrated HSC data with HSC. Used a tiling scheme named TOAST; and a tool named toasty(?) Took about 15 minutes to process

a tract, but could likely be made faster with effort. Addresses the whole sky visualization use case.

[YA: TO DO: format feature vs. tool table (Yusra)]

6 Recommendations

[YA: TO DO: John, Simon, Lauren, Gregory, Yusra: please review and edit] The recommendations are based on an assessment of the requested features, supporting use cases, and existing toolset evaluated. One currently unsupported feature is large image display. Users cross 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 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 resourcing 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

Górski, K.M., Hivon, E., Banday, A.J., et al., 2005, ApJ, 622, 759 (`arXiv:astro-ph/0409513`), doi:10.1086/427976, 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
Butler	A middleware component for persisting and retrieving image datasets (raw or processed), calibration reference data, and catalogs
CAM	Control (or Cost) Account Manager
CSV	Comma Separated Values
Camera	The LSST subsystem responsible for the 3.2-gigapixel LSST camera, which will take more than 800 panoramic images of the sky every night. SLAC leads a consortium of Department of Energy laboratories to design and build the camera sensors, optics, electronics, cryostat, filters and filter exchange mechanism, and camera control system
Commissioning	A two-year phase at the end of the Construction project during which a technical team a) integrates the various technical components of the three subsystems; b) shows their compliance with ICDs and system-level requirements as detailed in the LSST Observatory System Specifications document (OSS, LSE-30); and c) performs science verification to show compliance with the survey performance specifications as detailed in the LSST Science Requirements Document (SRD, LPM-17)

DM	Data Management
DMTN	DM Technical Note
DS9	Deep Space 9 (specific astronomical data visualisation application; SAOImage)
Data Management	The LSST Subsystem responsible for the Data Management System (DMS), which will capture, store, catalog, and serve the LSST dataset to the scientific community and public. The DM team is responsible for the DMS architecture, applications, middleware, infrastructure, algorithms, and Observatory Network Design. DM is a distributed team working at LSST and partner institutions, with the DM Subsystem Manager located at LSST headquarters in Tucson
Data Release Production	An episode of (re)processing all of the accumulated LSST images, during which all output DR data products are generated. These episodes are planned to occur annually during the LSST survey, and the processing will be executed at the Archive Center. This includes Difference Imaging Analysis, generating deep Coadd Images, Source detection and association, creating Object and Solar System Object catalogs, and related metadata
EFD	Engineering and Facility Database
FITS	Flexible Image Transport System
Firefly	A framework of software components written by IPAC for building web-based user interfaces to astronomical archives, through which data may be searched and retrieved, and viewed as FITS images, catalogs, and/or plots. Firefly tools will be integrated into the Science Platform
GUI	Graphical User Interface
HEALPix	Hierarchical Equal-Area iso-Latitude Pixelisation
HSC	Hyper Suprime-Cam
IPAC	No longer an acronym; science and data center at Caltech
IRAF	Image Reduction and Analysis Facility
KB	KiloByte
LDF	LSST Data Facility
LDM	LSST Data Management (Document Handle)
LSP	LSST Science Platform
LSST	Large Synoptic Survey Telescope

Object	In LSST nomenclature this refers to an astronomical object, such as a star, galaxy, or other physical entity. E.g., comets, asteroids are also Objects but typically called a Moving Object or a Solar System Object (SSObject). One of the DRP data products is a table of Objects detected by LSST which can be static, or change brightness or position with time
PSF	Point Spread Function
QA	Quality Assurance
QAWG	QA Strategy Working Group
RA	Right Ascension
SAMP	Simple Application Messaging Protocol
SLAC	SLAC National Accelerator Laboratory (formerly Stanford Linear Accelereator Center; SLAC is now no longer an acronym)
SST	Subsystem Science Team
SV	Science Validation
Science Pipelines	The library of software components and the algorithms and processing pipelines assembled from them that are being developed by DM to generate science-ready data products from LSST images. The Pipelines may be executed at scale as part of LSST Prompt or Data Release processing, or pieces of them may be used in a standalone mode or executed through the LSST Science Platform. The Science Pipelines are one component of the LSST Software Stack
Science Platform	A set of integrated web applications and services deployed at the LSST Data Access Centers (DACs) through which the scientific community will access, visualize, and perform next-to-the-data analysis of the LSST data products
Source	A single detection of an astrophysical object in an image, the characteristics for which are stored in the Source Catalog of the DRP database. The association of Sources that are non-moving lead to Objects; the association of moving Sources leads to Solar System Objects. (Note that in non-LSST usage ”source” is often used for what LSST calls an Object.)
Subsystem	A set of elements comprising a system within the larger LSST system that is responsible for a key technical deliverable of the project
T/CAM	Technical/Control (or Cost) Account Manager
UI	User Interface
UX	User Experience
WCS	World Coordinate System

WG	Working Group
afw	LSST's pipeline library code and primitives including images and tables
algorithm	A computational implementation of a calculation or some method of processing
background	In an image, the background consists of contributions from the sky (e.g., clouds or scattered moonlight), and from the telescope and camera optics, which must be distinguished from the astrophysical background. The sky and instrumental backgrounds are characterized and removed by the LSST processing software using a low-order spatial function whose coefficients are recorded in the image metadata
camera	An imaging device mounted at a telescope focal plane, composed of optics, a shutter, a set of filters, and one or more sensors arranged in a focal plane array
drill down	Move from a higher level aggregation of data to its inputs. For example, given data describing a tract, to drill down to constituent patches and then to objects. Also refers to the act of identifying an issue in a high-level summary of the data (e.g. an aberrant metric value) and interactively investigating its inputs to find the source of the problem
epoch	Sky coordinate reference frame, e.g., J2000. Alternatively refers to a single observation (usually photometric, can be multi-band) of a variable source
flux	Shorthand for radiative flux, it is a measure of the transport of radiant energy per unit area per unit time. In astronomy this is usually expressed in cgs units: erg/cm2/s
footprint	See 'source footprint', 'instrumental footprint', or 'survey footprint', 'Footprint' is a Python class representing a source footprint
metadata	General term for data about data, e.g., attributes of astronomical objects (e.g. images, sources, astroObjects, etc.) that are characteristics of the objects themselves, and facilitate the organization, preservation, and query of data sets. (E.g., a FITS header contains metadata)
pipeline	A configured sequence of software tasks (Stages) to process data and generate data products. Example: Association Pipeline
tract	A portion of sky, a spherical convex polygon, within the LSST all-sky tessellation (sky map). Each tract is subdivided into sky patches

warp	(noun) The pixels from a single CCD Exposure that overlap a given coadd patch, trimmed and resampled into the patch's coordinate system; in other words, an image that has been astrometrically registered to the common coordinate system of a tract