aflak: Visual Programming Environment with Quick Feedback Loop, Tuned for Multi-Spectral Astrophysical Observations

Malik Olivier Boussejra,[1] Shunya Takekawa,[2], Rikuo Uchiki,[1]
Kazuya Matsubayashi,[3] Yuriko Takeshima,[4] Makoto Uemura,[5] and
Issei Fujishiro[1]
[1]*Keio University, Yokohama, Kanagawa, Japan;* malik@boussejra.com

[2]*Nobeyama Radio Observatory, Minamimaki, Nagano, Japan*

[3]*Kyoto University, Kyoto, Japan*

[4]*Tokyo University of Technology, Hachiōji, Tokyo, Japan*

[5]*Hiroshima University, Hiroshima, Japan*

Abstract.
This paper describes a free (as in freedom), extendable graphical framework, aflak, that provides a visualization environment in particular for astrophysical data set. By leveraging visual programming techniques via an approach based on an node editor, aflak allows the busy astronomer to conduct fined-grained processing on multi-spectral data sets. While joining compute nodes together in the node editor, the final output of the transformations is smoothly displayed in a dedicated visualization window. This enables the astronomer to fine-tune all the interactive parameters of their program with a direct feedback loop.

1. Introduction

Astrophysics is a domain of knowledge where precision and reproducibility are absolute. While a single image may range from hundreds of megabytes to several gigabytes in size, the amount of granularity needed is truly minute by most visual computing standards: a single pixel in the picture of a faraway galaxy may be thousands of light-years large. Mostly, the only data astronomers are able to gather from far objects is their light. And from this light they must create, confirm or invalidate theories via the careful analysis of many case studies. Visually interacting with the data not only assists the astronomer in finding particular objects, but it also helps in the design of programs to verify the relevance of the computing by smoothly and regularly checking the output.

In this paper, we present a free and open-source software framework, aflak (Advanced Framework for Learning Astrophysical Knowledge), which is mainly aimed at dynamically analyzing multi-dimensional, astrophysical spectral data. aflak can load a data set, and provide a visual programming paradigm to apply transformations on it and visualize their outputs in real time, thus providing a fast and smooth feedback loop to astronomers. aflak, with its built-in support for FITS files and astrophysical processing, is currently especially adapted for multi-spectral astrophysical data.

2. Related Works

Astrophysics has had many viewers for FITS files. Most of these tools are free and open-source software. One of the most famous and most used viewer is SAOImage DS9 by Joye & Mandel (2003), which can open FITS files and offer basic analytic needs. Lately, QFitsView (Ott 2012) has been gaining tractions. Even some commercial endeavors, such as NightLight, have been released by Muna (2017). However, the previously mentioned tools are mainly viewers and do not offer many features for data analytics. Data analytics is mostly conducted with other tools, the oldest of which being IRAF (Tody 1986), then supplanted by PyRAF (De La Pena et al. 2001). IRAF, through PyRAF, paved their ways to Astropy, a Python library that can tackle most of the computing needs of astrophysicists (Robitaille et al. 2013) (e.g. transformation and image algebra). As Astropy internally uses NumPy, it is relatively easy to write custom analysis code in Python.

Now, as stated above, there is currently a clear separation between tools for viewing and analyzing in the astronomy ecosystem. Astrophysicists have a workflow consisting in manually analyzing data sets by applying and composing transformations on them. Only then do they export the result, e.g. as a FITS file, to see it inside a viewer. Even for Astropy, external tools (e.g. `matplotlib`) are required to view the results. `aflak`'s objective is to provide an integrated environment to both analyze and view astronomical data, with very fast iterations. While `matplotlib` may provide printing quality graphs, it is not really suitable for fast iterations on relatively big data sets.

`aflak` (`http://aflak.jp`) allows the user to compose algebraic transforms to implement new nodes using the provided visual programming interface. The user can combine elementary primitives to create their own macros. More than just allowing to organize nodes, these macros could then be exported and shared among their peers (planned feature). `aflak` provides an image algebra feature similar to that of NumPy, with which the user can play to smoothly visualize the resulting computations. In a word, `aflak` gives fine-grained control through a visual programmatic interface, but with immediate feedback thanks to the integrated data viewer. In the next section, we will present all of `aflak`'s currently implemented features.

3. `aflak`

`aflak`'s interface is presented in Figure 1. The upper layer of `aflak`'s architecture consists of a node editor engine and a plotting library to visualize the output data. The node editor engine has a compute back-end, which we called `cake`, that manages pending computational tasks in a multi-threaded manner, decoupled from the UI thread. `aflak` is built from the ground up in *Rust* (`http://www.rust-lang.org`), and is light enough to flawlessly and smoothly cope with gigabyte-sized data sets on a modern but standard laptop. *Rust* was chosen for its memory safety that does not sacrifice computing speed, and the relative ease—compared to bare *C/C++*—of running highly computational tasks on several threads. The user interface is drawn using *OpenGL* via bindings to the *Dear ImGui* Immediate Mode Graphical User Interface library, originally implemented in *C++* (`https://github.com/ocornut/imgui`). `aflak` is currently developed and tested in a *Linux* environment, though there is no reason to believe it will not work on other systems, as all technical choices are portable.

The visual programming interface is composed by a node editor, where one can author a visual program by creating/deleting nodes (of any of three types: value, transformation, and output nodes), and making connections between their input and output slots. Node can be combined to create more complex operations. Besides `aflak` can easily be extended with new nodes by loading a function implementing the binary interface recognized by `aflak`.

When an output node is created, a corresponding visualization window is spawned, showing in real time the data that is flowing into this output node. Whenever an error arises during the computing process, a clear error message will be propagated to the visualization window. Moreover, the visualization window provides usual visualization features such as advanced plotting for 1D and 2D data sets. Current graphical interface is in part inspired by the dynamic plotting implemented in PyQtGraph (`http://www.pyqtgraph.org/`). `aflak` prioritizes dynamic and interactive plotting dealing with fast varying data, contrary to what can be seen in `matplotlib` (Hunter 2007), which can output printing quality plots at the expense of speed and interactivity. New value nodes (node containing a value, not a transformation) can be created and updated from the visualization window, enabling fine-grained control over the value of the node from the output data. Finally, data output and the visual program itself can be exported for future reference to guarantee reproducibility of the study.

4. Future works and conclusion

`aflak` is a nascent project. Many features still need to be included for it to be fully usable by a broad range of astronomers. In order of importance, the wanted features are macro support and batch processing over many different but similar inputs. Implementation of more domain-specific transformations is desireable. In addition, convenience UI functions such as copy-pasting or bulk-selection of nodes are desirable. Loading of FITS data from public URLs of open data sets is considered.

Acknowledgments. This work is supported by JSPS KAKENHI Grant Numbers 17K00173 and 17H00737.

References

Bundy, K., et al. 2015, The Astrophysical Journal, 798, 7. 1412.1482

De La Pena, M., White, R., & Greenfield, P. 2001, in Astronomical Data Analysis Software and Systems X, vol. 238, 59

Hunter, J. D. 2007, Computing in Science & Engineering, 9, 90

Joye, W., & Mandel, E. 2003, in Astronomical data analysis software and systems XII, vol. 295, 489

Muna, D. 2017, Publications of the Astronomical Society of the Pacific, 129, 058003

Ott, T. 2012, Astrophysics Source Code Library

Robitaille, T. P., et al. 2013, Astronomy & Astrophysics, 558, A33

Tody, D. 1986, in Instrumentation in astronomy VI (International Society for Optics and Photonics), vol. 627, 733

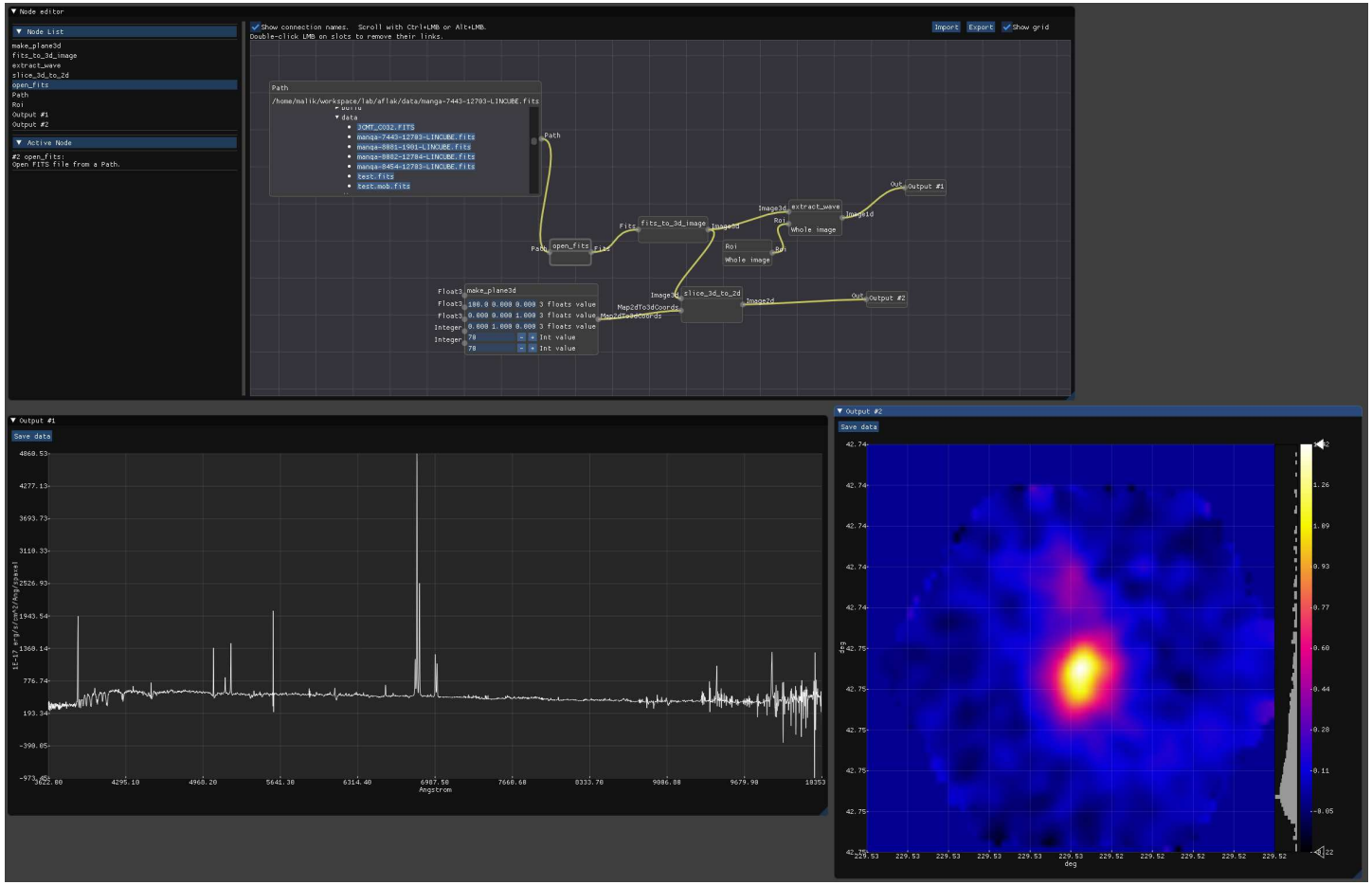

Figure 1. `aflak` showing a galaxy from the SDSS MaNGA data set by (Bundy et al. 2015). Above, you can see the node editor window. Below, you can see a window for each of the connected output nodes. Each window shows the data that flows into its dedicated output node. All parameters can be updated. All the visualized data depending on the updated parameters will get updated immediately.