VisiOmatic 3:
Remote image visualization with new Python-based features

Emmanuel Bertin,[1,2] and Conrad Holmberg[1]

[1] *Canada-France-Hawaii Telescope, USA;* bertin@cfht.hawaii.edu

[2] *AIM / CEA / CNRS / Université Paris-Saclay / Université Paris Cité, F-91191 Gif-sur-Yvette, France*

Abstract. The VisiOmatic package is a comprehensive remote visualization system designed for large, multispectral astronomical image datasets. We present the third version of the package, primarily developed for advanced "quicklook" image visualization of imaging data at CFHT, but versatile enough for a wide range of applications, including use as a stand-alone viewer. The server-side code has been completely rewritten in Python, introducing new features such as support for hyperspectral datacubes and multi-extension files from mosaic cameras, just-in-time caching of FITS data, and temporal image sequence playback. `VisiOmatic` 3 is released under the MIT license.

1. Introduction

The increasing scale of astronomical datasets and reactivity requirements from modern research programs have highlighted the need for tools that enable efficient remote access and interactive visualization. As observatories generate increasingly large and detailed images, traditional desktop-based software encounters challenges related to accessibility, scalability, and real-time usability, particularly in distributed research environments.

Web-based platforms provide a practical solution to these limitations, allowing researchers to explore and analyze data remotely through standard browsers. VisiOmatic[1] (Bertin et al. 2015, 2019) was designed to be embedded in such platforms, offering a framework for visualizing large astronomical images directly in the browser. The third version of VisiOmatic, developed at the Canada-France-Hawaii Telescope (CFHT), includes a complete rewrite of the server code in Python and introduces several new features that are particularly beneficial for observatories, such as providing PIs with advanced quick-look capabilities for exposures shortly after acquisition.

2. Technical Overview

As a web application, VisiOmatic consists of a client component and a server component.

[1] `https://www.visiomatic.org`

The VisiOmatic web client is built on the Leaflet[2] JavaScript mini-framework. The client interface is fully asynchronous and entirely customizable through module options, themes, and Cascading Style Sheets (CSS). It is compatible with touchscreen devices such as those running iOS and Android. The client code relies solely on ECMAScript 2016+ and HTML5. Version 3 of the web interface introduces several new features, including the ability to display multi-extension FITS files from mosaic cameras (Fig. 1) and animation and real-time color compositing of hyperspectral datacubes.

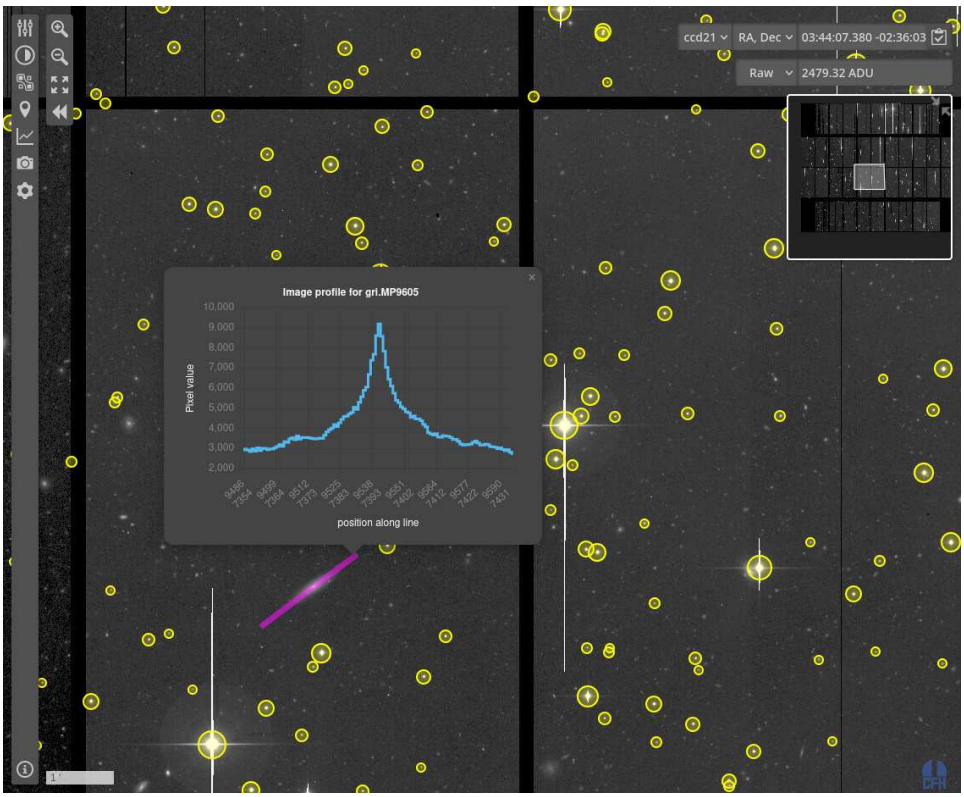

Figure 1. Screenshot of a mosaic exposure from the CFHT MegaPrime instrument, with overlays of the Gaia DR3 catalog and an image profile, displayed using the VisiOmatic web client in a dark theme configuration.

The server component of version 3 is written in Python and communicates with clients via the FastAPI web framework[3], which implements the Asynchronous Server Gateway Interface (ASGI) specification (Fig. 2). This replaces the legacy IIPImage-astro Fast Common Gateway Interface (FCGI) C++ code of earlier versions.

The server component operates as a web service that encodes large, high-resolution images on-the-fly and delivers them as compressed "tiles" in image formats natively supported by web browsers. The server code processes science-grade hyperspectral data stored in floating-point format, and perform operations such as amplitude rescal-

[2] `https://leafletjs.com`

[3] `https://fastapi.tiangolo.com`

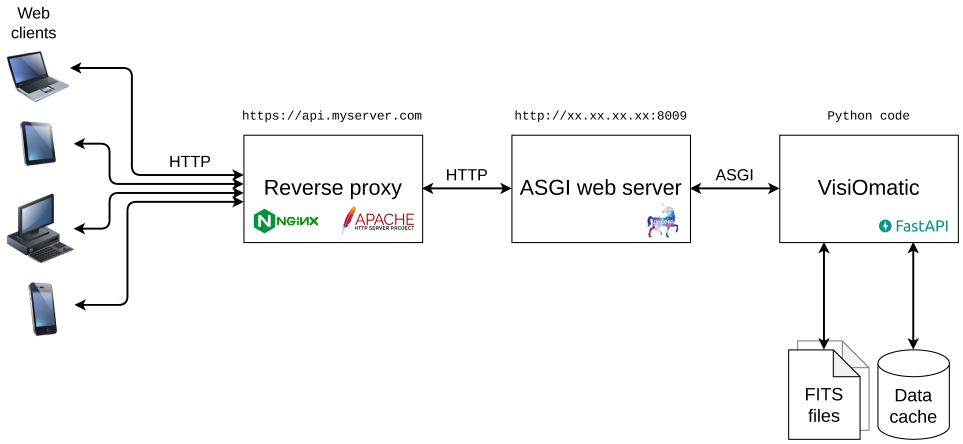

Figure 2. The VisiOmatic service can be accessed through a reverse proxy or directly via the built-in Uvicorn ASGI HTTP server for local use.

ing and channel mixing before transmitting the resulting image to the client. The web API is RESTful and adheres to the OpenAPI specification[4].

Previous versions of VisiOmatic required converting original image data files into a tiled, multi-resolution TIFF format. In contrast, the new version works directly with FITS images, including data cubes and multi-extension FITS (MEF) files, using a just-in-time caching strategy. When a client queries data from a given image for the first time, the FITS image pixels are rebinned at various resolutions, tiled, cached on disk, and mapped to memory. Subsequent queries access the cached data, unless the FITS file has changed or the cache entry has been deleted to accomodate more recent queries, following a Least-Recently Used (LRU) cache replacement policy.

3. Performance

With all image processing and compositing performed server-side, the volume of data transmitted to the browser and the computational load on the client are reduced to a minimum. However the Python server code can become a performance bottleneck when handling concurrent connections from a large number of users. Fortunately, VisiOmatic leverages Python's multiprocessing capabilities, and performance tests indicate that the current server code can deliver over a thousand *unbuffered* 256×256 JPEG tiles per second per CPU core (Fig. 3). Thus, a single, modestly sized server can support up to a thousand concurrent users for monochannel images and 10+ concurrent users for multi- or hyperspectral images.

The image caching process — performed only once when an image is first queried — achieves a throuput of approximately 100 megapixels per second on a Gen.4 SSD: a 380 megapixel exposure from the MegaPrime instrument can be cached in about 4 seconds on modern hardware equipped with fast storage.

[4] `https://spec.openapis.org/oas/latest.html`

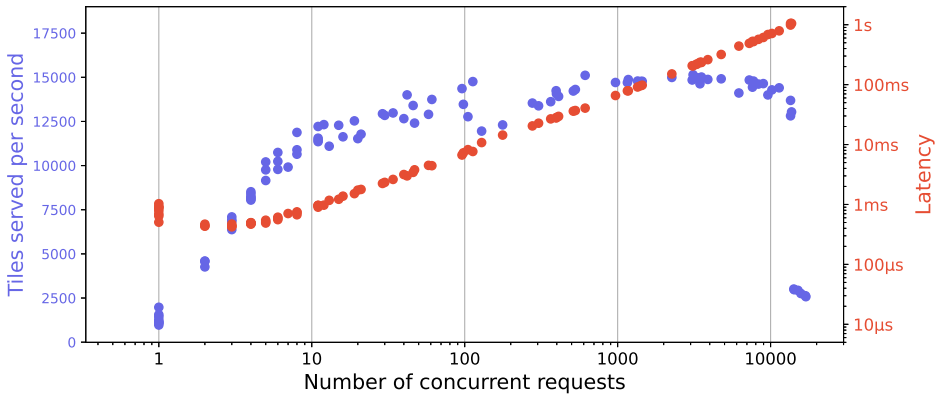

Figure 3. Tile throughput and latency as a function of concurrency for a VisiOmatic server running on a single machine equipped with a Gen.4 SSD and a 13th Gen., 12-core Intel CPU. The tiles are 256×256 JPEG images.

4. Conclusion

VisiOmatic 3 represents a significant step forward from previous versions. The transition to Python for the server code simplifies the implementation of new features for astronomers, while the just-in-time caching of FITS data enables the tool to function as a regular FITS viewer. VisiOmatic 3 has been successfully tested on the Linux, macOS and Windows operating systems and is available via the Python Package Index[5], on GitHub[6], and as a Docker image[7].

Acknowledgments. This work made use of Astropy:[8] a community-developed core Python package and an ecosystem of tools and resources for astronomy (Astropy Collaboration and Contributors 2022).

References

Astropy Collaboration and Contributors 2022, ApJ, 935, 167. 2206.14220

Bertin, E., Marmo, C., & Bouy, H. 2019, in Astronomical Data Analysis Software and Systems XXVI, edited by M. Molinaro, K. Shortridge, & F. Pasian, vol. 521 of Astronomical Society of the Pacific Conference Series, 651

Bertin, E., Pillay, R., & Marmo, C. 2015, Astronomy and Computing, 10, 43. 1403.6025

[5] https://pypi.org/project/VisiOmatic

[6] https://github.com/astromatic/visiomatic

[7] https://hub.docker.com/r/ceerad/visiomatic

[8] https://www.astropy.org