FITSWebQL: an interactive preview system for very large FITS data cubes

Christopher ZAPART*, Yuji SHIRASAKI, Masatoshi OHISHI, Yoshihiko MIZUMOTO, Wataru KAWASAKI, Tsuyoshi KOBAYASHI, George KOSUGI, Eisuke MORITA, Akira YOSHINO, Yohei HAYASHI

(Received 2020 May 7; accepted 2020 Aug. 14)

Abstract

We have upgraded the JVO ALMA WebQL web service - available through the JVO ALMA FITS archive - to include legacy data from other telescopes, for example Nobeyama NRO45M in Japan. The updated server software has been renamed FITSWebQL. In addition, a standalone desktop version, independent of the Japanese Virtual Observatory, that runs on local end-user computers (laptops, desktops) and supports Linux, macOS and Windows 10 Linux Subsystem (Bash on Windows) is also available for download from https://github.com/jvo203/fits_web_ql.

The FITSWebQL server enables viewing of even over 100 GB-large FITS files in a web browser running on a PC with a limited amount of RAM. Users can interactively zoom-in to selected areas of interest with the corresponding frequency spectrum being calculated on the server in near real-time. Hence at a glance users gain access to large easy-to-see images as well as the corresponding frequency spectra that are visible both at the same time. The client (a browser) is a JavaScript application built on WebSockets, HTML5, WebGL, SVG and WebAssembly.

The new FITSWebQL lays foundations towards supporting an interactive preview of terabyte-class FITS files. In addition to handling larger files, FITSWebQL also improves support for real-time spectrum updates through tackling the latency problem in two ways. First, it introduces an *adaptive frame rate* control: monitoring the network latency and local web browser responsiveness, and reducing the frames-per-second (FPS) rate as and when necessary. Secondly, our software tracks in real time end user's mouse movements with the Kalman Filter, which is then used to predict the future target mouse position after taking into account the network latency and server-side computation time. Hence we speculatively deliver ahead-of-time real-time spectrum data for positions where the user is likely to be looking at. The new version also allows users to view multiple FITS files simultaneously either in a special RGB composite mode (presently limited to NRO45M FUGIN only), where each dataset is assigned one RGB channel to form a single colour image, or as separate image tiles. Spectra from multiple FITS cubes are shown together too.

The paper describes the main features of FITSWebQL and its technical architecture. We also touch on some of the recent developments, such as a switch from C/C++ to Rust (see https://www.rust-lang.org/) for improved stability, better memory management and “fearless concurrency”, displaying FITS data cubes in the form of interactive on-demand video streams in a web browser or efforts to incorporate machine learning for an improved user experience.

Since version 3 FITSWebQL has had a robust choice of different colourmaps, automatic integration with the Splatalogue molecular database, fixed and variable (auto-scaled) Y-Axis, synthesised beam overlay, manual reference frequency/source velocity corrections as well as a 3D viewing mode and client-side contouring handled in JavaScript.

Key words: ALMA, NRO45M, WebQL, radio-astronomy, optical astronomy, FITS data cubes

* Corresponding author
Email address: chris.zapart@nao.ac.jp (C. Zapart)

1. Introduction

The ALMA WebQL service (Allen et al., September 2013; Eguchi et al., 2013) operated by the Japanese Virtual Observatory (JVO) first went into operation towards the end of 2012. Since that initial release by Satoshi Eguchi the ALMA project has been releasing ever larger FITS files, prompting the JVO team to develop a revised second edition (Zapart et al., 2019a) of the ALMA WebQL service that subsequently went live in March 2016 using a vastly upgraded hardware infrastructure. Since then, in order to keep up with ever growing FITS file sizes coming out of the ALMA observatory and also to offer improved functionality, newer versions have been released on a regular basis. For example, the version 3 — released in 2017 — introduced an experimental 3D view of FITS data cubes. Another innovation introduced in the version 3 was a switch from HTTP to WebSockets for a bi-directional communication between the web browser and the ALMA WebQL server located in Japan. This reduces the network latency, helping deliver smoother real-time spectrum updates compared to the previous version 2. The earlier 2nd edition introduced on an experimental basis real-time spectrum updates (disabled by default) that only worked well over a local network connection within the NAOJ Mitaka campus in Japan. In the version 3, by default we enabled real-time spectrum updates for all users worldwide. To facilitate smooth updating of a spectrum following user mouse movements, we continuously monitor the network latency and local web browser responsiveness, reducing the frames-per-second (FPS) rate as and when necessary. Secondly, end-users' mouse movements are also tracked in real time with help of the Kalman Filter, which is then used to predict a future target mouse position after taking into account network latency and computation time. In the last resort, in case of a severe network deterioration users still have the option to disable real-time updates from the menu. Since version 3 users have been able to make various on-the-fly image adjustments such as tuning the noise sensitivity, trying out different global tone mapping functions (i.e. logarithmic or logistic instead of linear) or displaying contours, all handled locally in a web browser via JavaScript. In order to facilitate fast on-demand viewing of large ALMA datasets, we cache FITS files locally using NVMe PCI Express Solid State Drives housed in the ALMA FITSWebQL server.

Released in the second half of 2018 and completely re-written from scratch in the Rust programming language (Zapart et al., 2019b), the current version 4 features real-time video streaming of individual frequency channels from FITS data cubes. Since the 4.1.6 revision (released in February 2019) FITSWebQL has had built-in support for ZFP compression (Lindstrom, 2014) in order to reduce the server storage requirements for cached FITS files. Readers can access the service from the JVO Portal found at https://jvo.nao.ac.jp/portal/top-page.do. The latest version 4 of the software (which also includes a standalone desktop edition) is freely available from the following GitHub repository: https://github.com/jvo203/fits_web_ql.

An unchanging motivation behind this web service is being able to provide a FITS file preview (quick-look) and cut-out capability through a web browser. The service allows end-users to view over 100 GB-large FITS files in a web browser without ever having to download the underlying FITS files. After previewing FITS files users may choose to download interesting datasets either in whole or to stream a partial region-of-interest (cut-out) from the JVO server to their own computers.

This paper is not only a significantly extended version of the proceedings contribution (Zapart et al., 2019b) but it is also an invaluable source of detailed information on the conceptual/technical design and implementation, which cannot be provided within a short proceedings paper.

2. Main Features

This section takes a reader on a brief tour of FITSWebQL's main features.

2.1 Image overlay

The design of the FITSWebQL client web site strives to be simple and intuitive. The goal is to reduce the amount of user mouse clicks, to get out of the user's way. In order to fully maximise the available screen area, users are initially presented with a large — browser-window wide — spectrum plot overlay-ed on top of a large FITS image, as shown in Figure 1. Moving a mouse anywhere over the main FITS image brings up an additional overlay (either a circle or a rectangle) displaying a zoom-in view of the area around the mouse pointer. Using a mouse scroll wheel users can enlarge or shrink the zoomed viewing area. While a mouse is being moved over the main image, the corresponding spectrum is re-calculated in real-time on the server and sent via WebSockets to the browser for

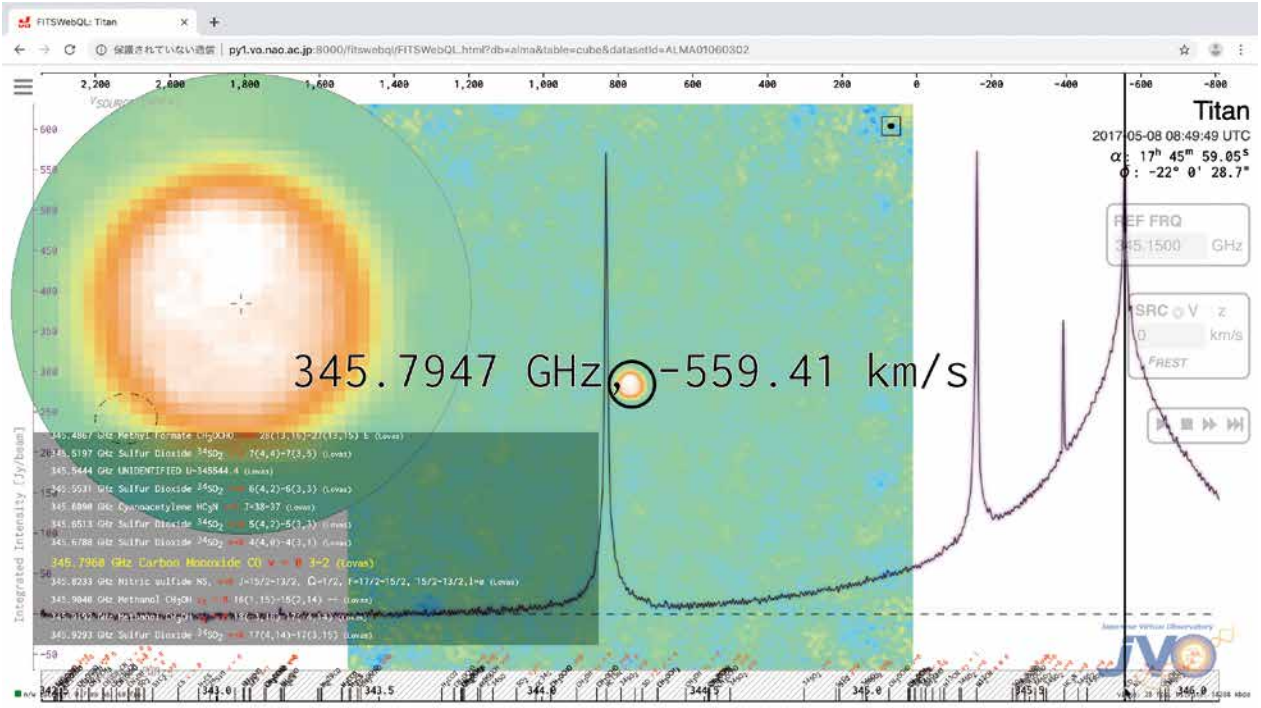

Figure 1: An integrated image/spectrum overlay with spectral lines pulled from Splatalogue. Object: Titan

display. Users can immediately notice that the Y-axis is being re-scaled constantly to accommodate the changing min/max range of the updated spectra. At any time by pressing a keyboard key "s" a user can freeze the Y-axis range to its present state. Users can re-enable the default "auto-scale" behaviour at will from the FITSWebQL menu[1] (*Preferences/autoscale y-axis*).

At this point it is worth mentioning an official ALMA project for displaying ALMA datasets remotely in a web browser: the CARTA viewer[2]. For a long time available only as a standalone desktop application, only recently — after having its development team changed — has it gained a remote FITS file viewing capability via a web browser. In contrast, for the most time FITSWebQL has since its inception been the first and only product to offer a remote viewing service. There is a single fundamental difference between CARTA and FITSWebQL: a "per-cube" and "per-plane" FITS image display. In FITSWebQL the main image is calculated as an intensity integrated over all 2D planes present in the 3D data cube (a "per-cube" image). This differs from the recently renewed CARTA viewer which by default displays an image of the first plane only[3]. In addition, as far as the first author is aware a long-standing image display and visualisation tool for astronomical data — SAOImage DS9[4] — also only displays "per-plane" images for radio-astronomy FITS data cubes. Displaying per-plane images is faster as even in the case of terabyte-class FITS files one needs to read-in from disk only the first 2D plane before being able to show an image to a user. In contrast, our approach — although more time consuming — can be considered "more complete" as it presents users with images constructed using entire data sets. To this end we have optimised FITSWebQL from the very beginning to read the entire FITS file as fast as possible in order to present users with an image based on the entire data. Specifically, FITS files are read in parallel using multiple CPU cores utilising SIMD extensions (i.e. AVX-512) as much as possible when making the necessary endianness and then half-float conversions. The second time a particular FITS file is being accessed we only read the FITS header from the original FITS file. The data cube itself will be read in parallel from an internal FITSWebQL binary data cache that uses either 16-bit half-floats (for a 2:1 compression ratio) or ZFP compressed floating-point arrays (Lindstrom, 2014) (a 4:1 compression ratio).

[1] No mouse clicks are required, the menu will appear automatically by hovering a mouse anywhere near the top of the web page.

[2] https://cartavis.github.io/

[3] Changing the default setting will make CARTA calculate a "per-cube" histogram used to display a single frequency channel image. However, based on our ad-hoc tests the CARTA viewer seems to take much longer than FITSWebQL to produce an all-data histogram and its corresponding image/spectrum combination. To the best knowledge of the first author there is no specific setting to make CARTA produce a "per-cube" image as a summation of all frequency channels.

[4] https://sites.google.com/cfa.harvard.edu/saoimageds9/

When a user hovers a mouse over the X-frequency axis located at the bottom of the screen, the JVO server sends in real-time images corresponding to individual 2D planes to the client web browser. These images are encoded in the form of a monochrome HEVC video stream, decoded in the browser using WebAssembly, the colourmap is then applied and the final image is overlaid over the original per-cube image. A user can also make a partial cut-out along the X-axis with a mouse (a frequency sub-region selection) and an image corresponding to a partial FITS cube will be sent from the server together with an updated spectrum (also re-calculated server-side).

One of the benefits of handling FITS files without relying on external libraries (i.e. cfitsio or Starlink AST) is that the relevant image/spectrum statistics can be built up in parallel "on the fly" whilst reading from disk the FITS data. In addition, perhaps a lesser-known feature of FITSWebQL is its ability to display FITS files loaded from external URL resources (for example URLs of FITS files residing on the Hubble Space Telescope web site). Whilst the FITSWebQL slowly downloads an external FITS file, after an optional GZIP decompression the incoming data chunks are already being converted into the half-float format, statistics gathered and the final image/spectrum is being built-up incrementally.

2.2 FITS header handling

Another benefit of performing a custom FITS I/O is the ability to adapt the processing logic based on detected telescopes. For example, in case of data coming from the Nobeyama NRO45M telescope we specifically look for a "*LINE*" or "*J_LINE*" header keyword, which then enables us to display this additional line information (i.e. ^{12}CO). This is illustrated in Figures 6 and 8.

FITSWebQL features three internal operating modes based on the type of a detected telescope: optical, radio-astronomy and X-Ray. We first attempt to detect the telescope based on the "*TELESCOP*" keyword by looking for strings such as "*alma*", "*ska*", "*vla*" or "*nro45m*". Failing that we also search the remaining header keywords for any other hints, for example "*ASTRO-F*", "*HSCPIPE*", "*SUPM*", "*MCSM*", "*suzaku*", "*hitomi*" or "*x-ray*". Relying on the hints gained from the FITS header FITSWebQL modifies the way the FITS cube data is loaded (for example by ignoring "−1" pixel values in case of X-Ray data) and displayed (i.e. switching to a "*logistic*" tone mapping function for "*ASTRO-F*" data or preferring to use a "*ratio*" tone mapping in optical astronomy).

2.3 Splatalogue integration

Beginning with ALMAWebQL v2.1 we introduced a spectral lines overlay (NIST Recommended Rest Frequencies by Frank J. Lovas (Lovas, 2009 Revision)) positioned over the bottom frequency axis. Since FITSWebQL v3 we have gone further by offering a seamless integration with version 3 of the full Splatalogue database (Markwick-Kemper et al., 2006), which also contains the NIST rest frequencies amongst other sources.

The Figure 1 illustrates integration with the Splatalogue where upon moving/hovering a mouse over the X-axis a dark pop-up window in the bottom left-hand side[5] displays details about spectral lines in the vicinity of the mouse pointer. Furthermore, in the Figure 2 the reader can see how from the top "Splatalogue" menu one can select sources for spectral lines as well as adjust the line intensity cut-off in order to show/hide weaker spectral lines.

For convenience we have also introduced a direct shortcut to the Splatalogue web page: by pressing the "ENTER" key with a mouse pointer positioned over the X-axis users can open up a new tab containing a Splatalogue web page centred around the currently selected frequency.

2.4 Image manipulation

The "Image" menu (see the Figure 3) offers a typical choice of several tone mapping functions that are used to convert the dynamic-range 32-bit floating-point intensity pixels into the 8-bit range of a computer display. From the same menu users can also change colourmap preferences. The menu is completely interactive. Based on a currently selected tone mapping function, users can adjust with a mouse the image noise sensitivity and/or the black/white[6] histogram cut-off levels.

2.5 Contour lines

Image contours can be enabled via a *View/contour lines* menu item as shown in the Figure 4. Contouring

[5] The spectral lines pop-up window will move to the right side when a mouse moves to the left area so as not to obscure the relevant spectrum view.

[6] Most tone mapping functions imply the use of black/white levels. The exception is the *logistic* setting for which users can adjust the histogram median level instead.

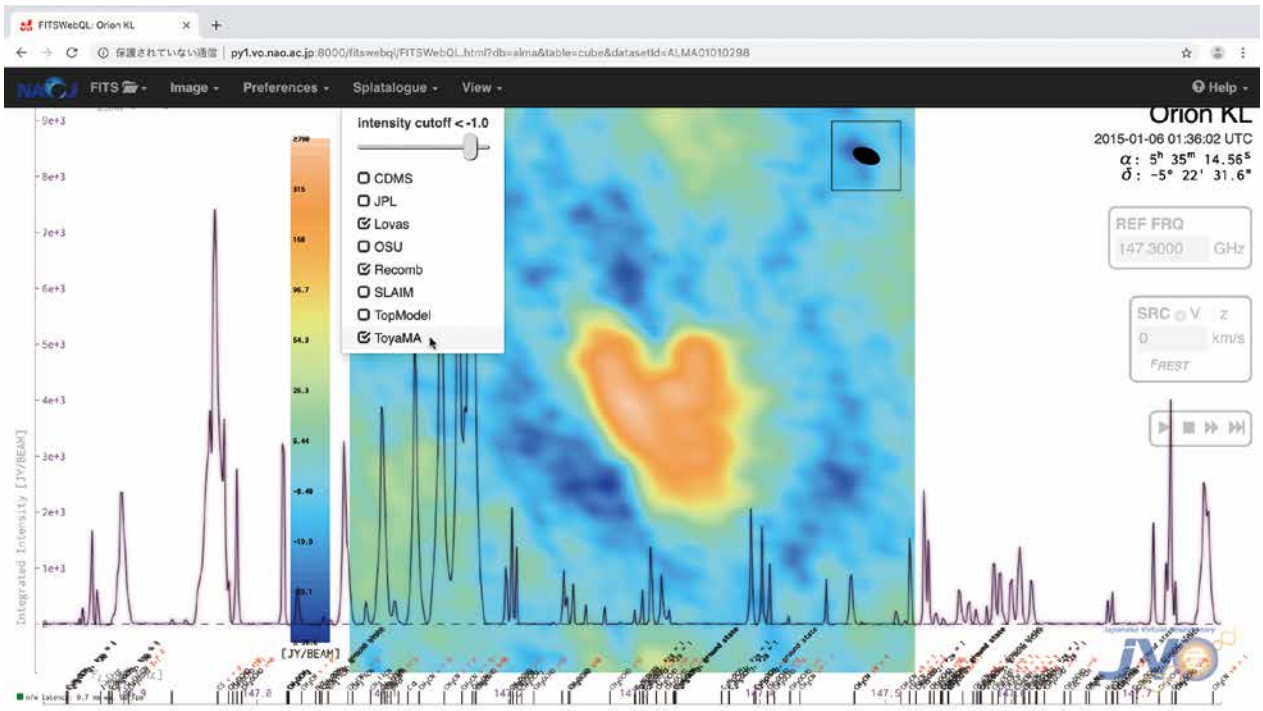

Figure 2: Full Splatalogue v3 integration. Object: Orion KL

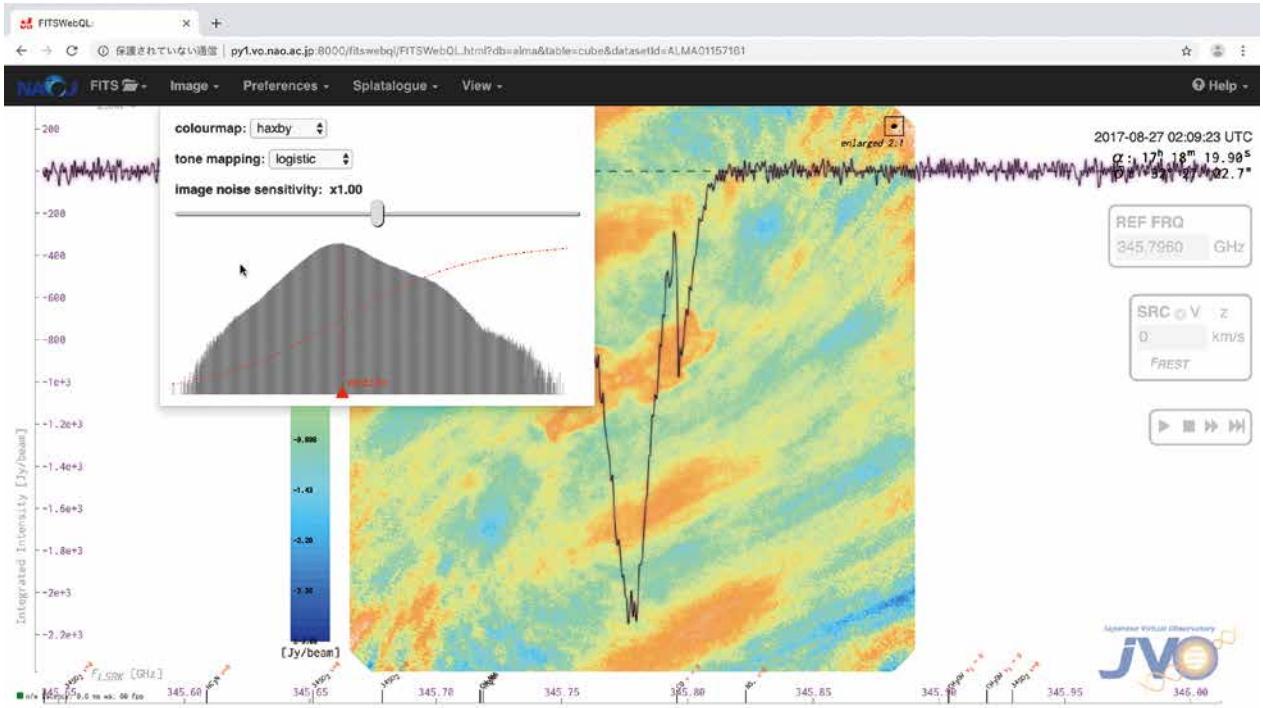

Figure 3: An interactive image manipulation menu. Object: Cotton Candy Nebula

itself is performed on the client side in the browser using a JavaScript version of the Marching Squares algorithm (Maple, 2003). Taking advantage of the Web Worker API, we have offloaded the calculation of multiple contour levels to multiple parallel web workers. The *View* menu also contains many other items allowing the user to control various aspects of the User Interface. Perhaps the most interesting item might be the *3D surface*, which displays a screen-wide three-dimensional view of the main intensity image (see the Figure 5). With a mouse users can zoom-in and out and/or rotate the 3D surface. This feature is also implemented client-side with help of the WebGL-

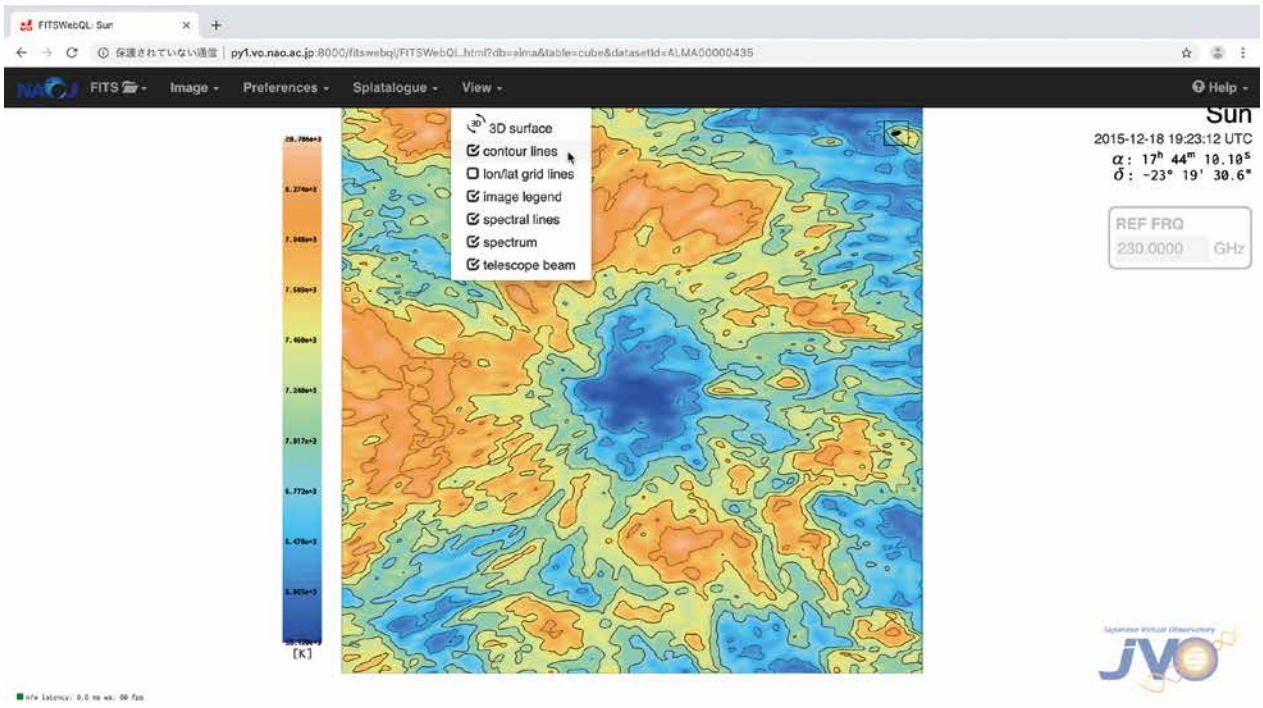

Figure 4: Client-side contouring using the Marching Squares algorithm. Object: the Sun

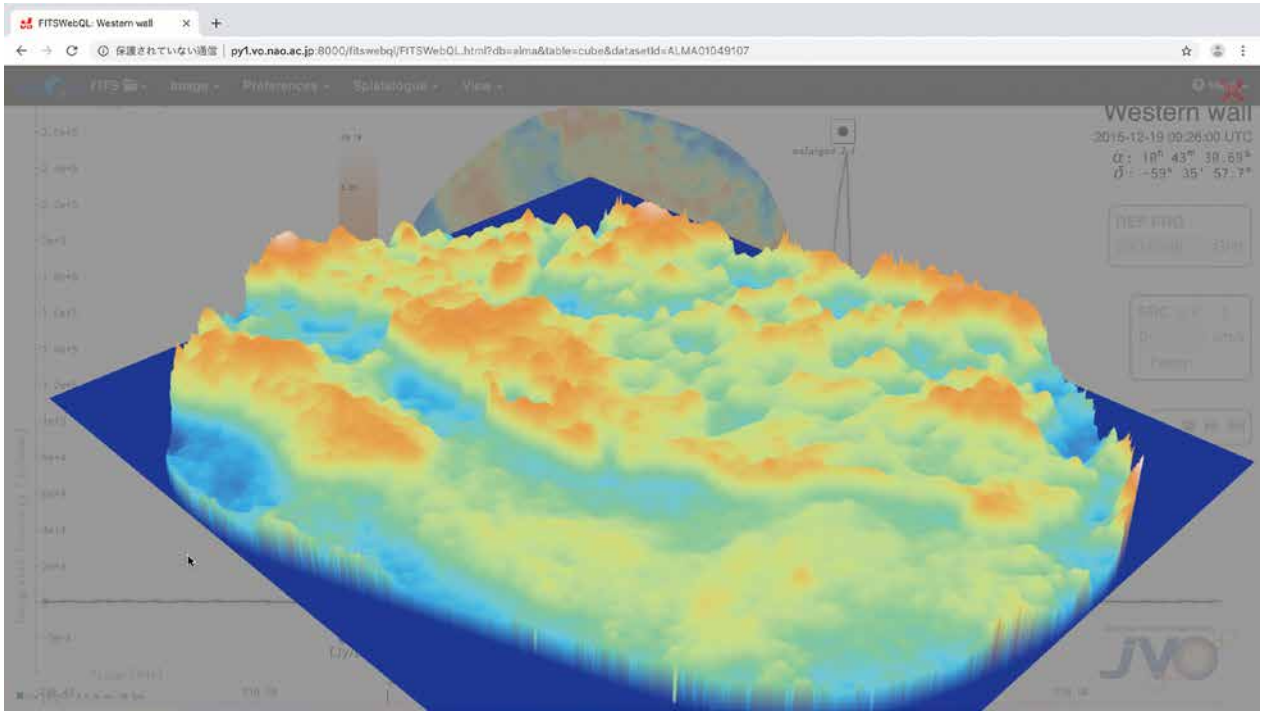

Figure 5: An interactive 3D view with Three.js WebGL. Object: Western Wall

accelerated Three.js JavaScript library.

Keen users are encouraged to explore all the remaining menu options. There is also a brief *Help* menu available in the right hand side.

2.6 Multiple datasets

For over two years now — starting with version 3 — FITSWebQL has supported viewing more than one dataset at a time. Initially requested by the Nobeyama 45m radio-telescope team, in version 4 we have extended

the support for multiple datasets to optical astronomy too. Furthermore, two distinct viewing modes are available: an RGB composite mode (see Figures 6 and 7), reserved for up to three datasets, and an unrestricted tile view mode (see Figures 8 and 9). In particular viewing datasets for multiple spectral lines (i.e. ^{12}CO, ^{13}CO and C^{18}O) side-by-side in the tile mode is informative. The Figures 8 and 9 show how, depending on the number of datasets, the tile mode adjusts automatically the placement and size of images. One major limitation of the current implementation is the need for the FITS headers belonging to multiple datasets to be aligned perfectly in all axes (in other words world coordinates and image dimensions must match). The reason being that pixel (and not world) coordinates are being exchanged

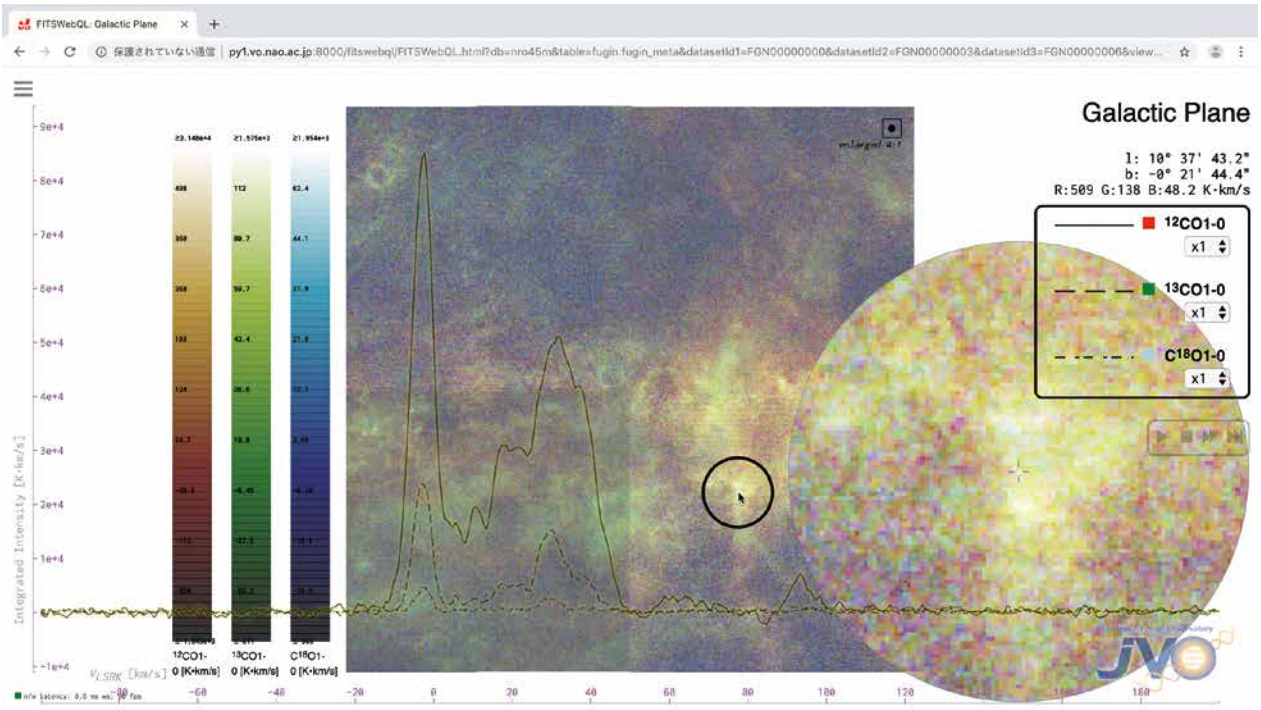

Figure 6: RGB composite mode for up to three datasets (radio astronomy, NRO45m). Object: Galactic Plane

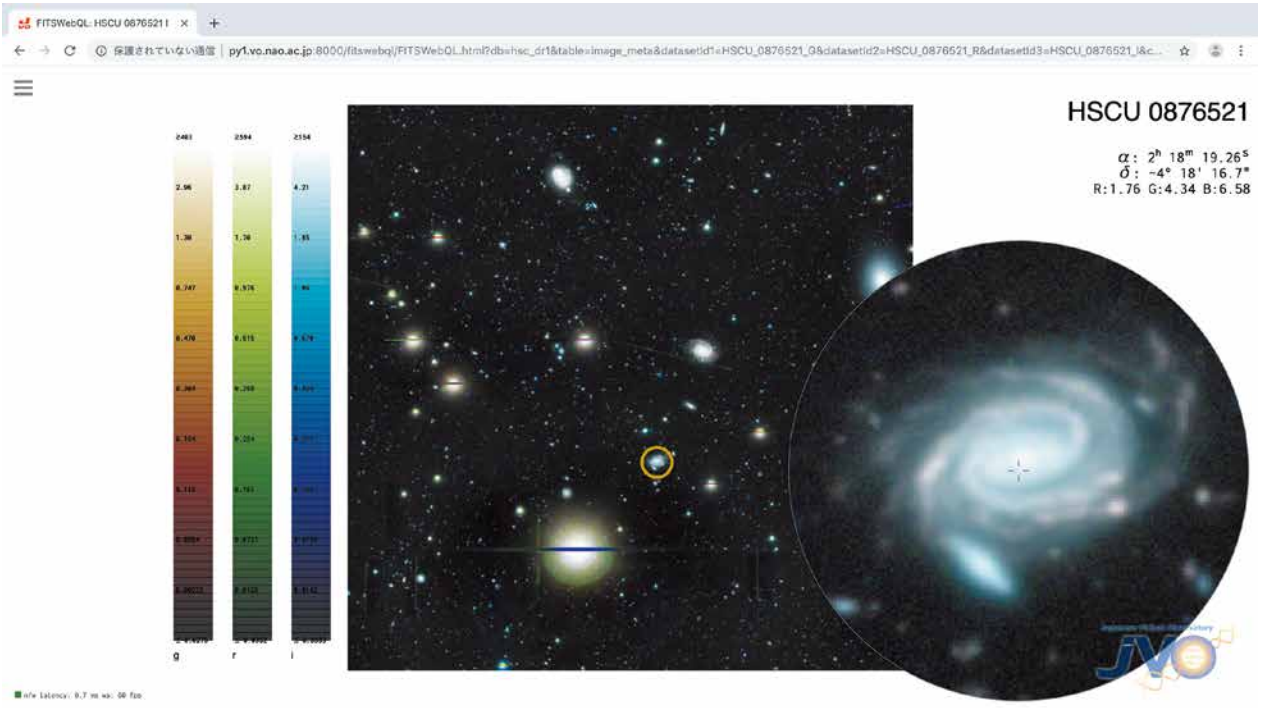

Figure 7: RGB composite mode for up to three datasets (optical astronomy, Subaru HSC).

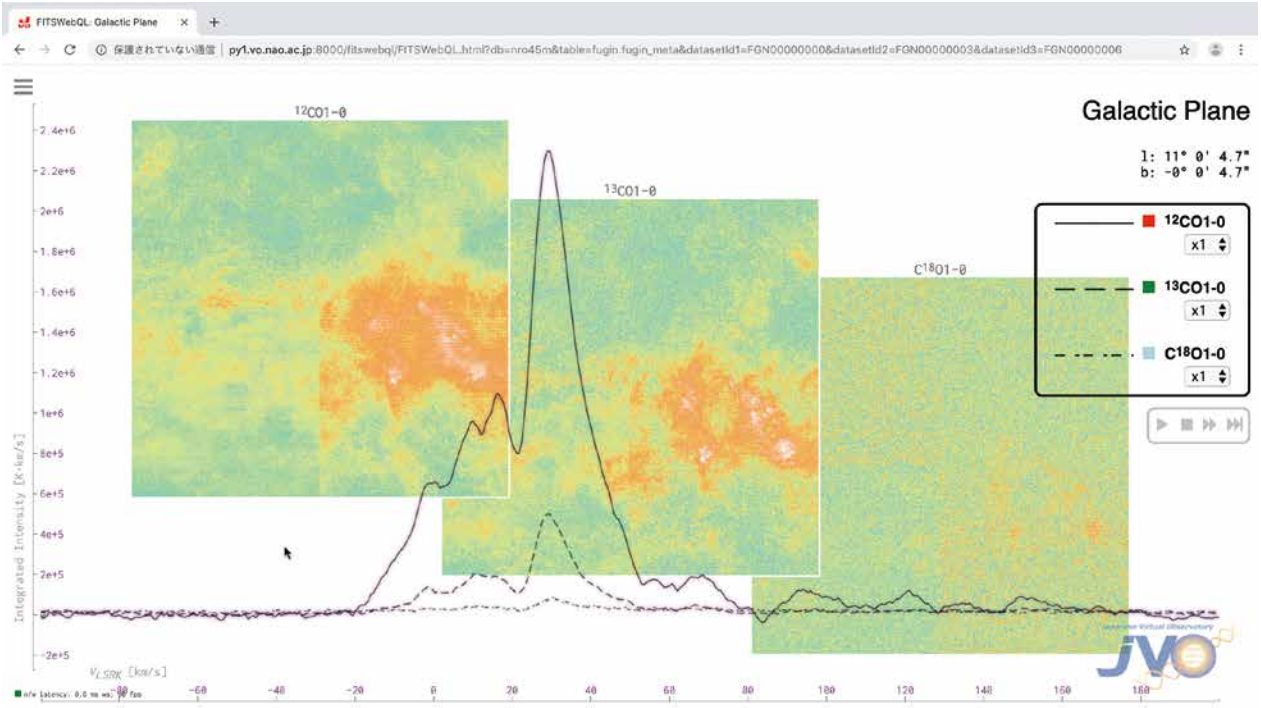

Figure 8: Tile-view mode (radio astronomy, NRO45m). Object: Galactic Plane

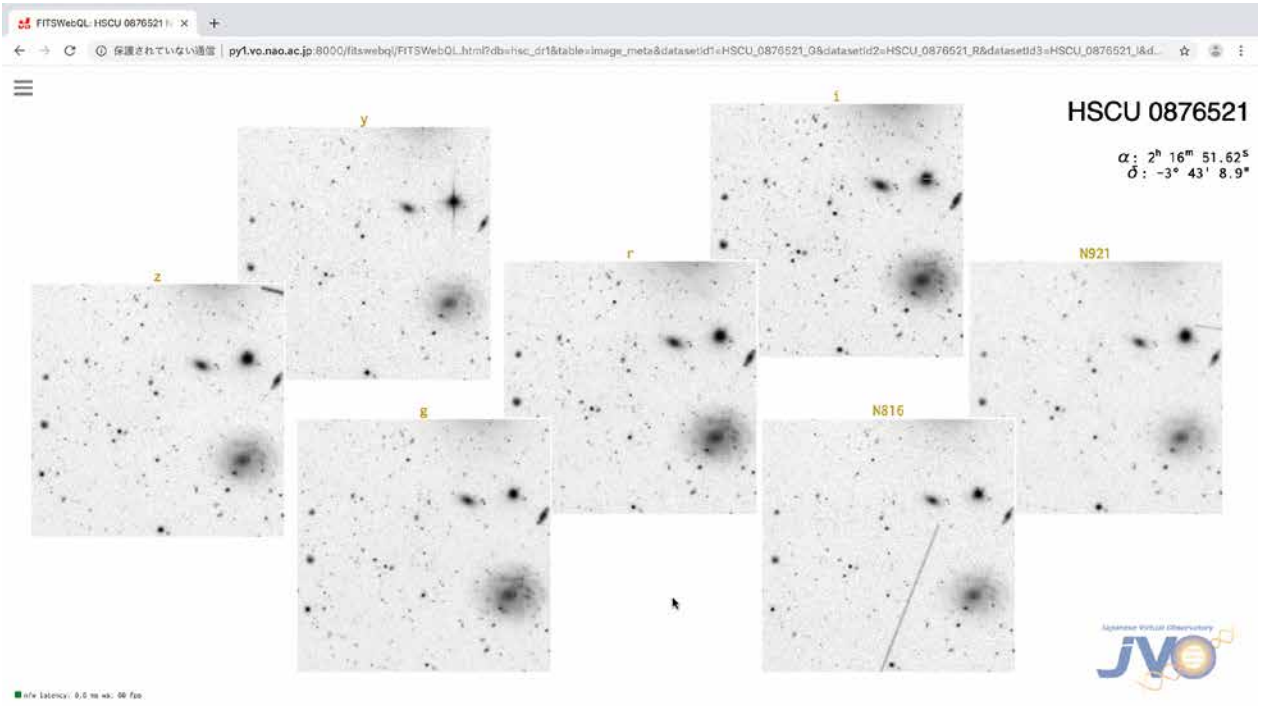

Figure 9: Tile-view mode (optical astronomy, Subaru HSC). In this example users can view simultaneously seven FITS datasets (each one corresponding to a dierent filter), with the image zooming in/out and panning being coordinated across all the tiles.

during a communication between the browser client and the FITSWebQL server. Both the client and the server currently expect that world coordinates should correspond to the same physical pixels across multiple datasets. We will try to remove this restriction in future versions, allowing users to view partially or even non-overlapping datasets.

3. Technical Architecture

The new version 4 initially started as a small feasibility study to find how easy it would be to re-implement the server part of C/C++ FITSWebQL v3 in Rust. There are good reasons for switching from C/C++ to a new systems

programming language such as Rust as it brings significant benefits, for example memory safety (*no memory leaks*), thread safety (*no data races*), better (smoother) multithreading compared with OpenMP in C/C++ and a complete lack of segmentation faults (*no crashes*) due to inherent safety measures built into the Rust language (see the Table 1 for a comparison between C/C++ and Rust). It is certainly possible to write bug-free programs in C/C++ that are free of memory leaks and do not crash. However, from a programmer's standpoint Rust makes accomplishing these tasks easier, all without sacrificing performance. In addition, Rust has an integrated HTTP/WebSockets networking library: actix-web that compares favourably with the previously used disjointed mix of C libmicrohttpd and C++ μWebSockets.

With the Rust port under-way work we had also been working on adding video streaming capability to the main v3 C/C++ code base. However, once all the bottlenecks in Rust have been identified and dealt with, another benefit has come to light: the original C/C++ code base had become rather complex and adding new functionality has turned into an error-prone process running the risk of introducing memory leaks and bugs. Hence we decided to complete the switch-over from C/C++ to Rust and add streaming video functionality to the new Rust version 4. The Figures 10–11 (the Kalman Filter part will be explained in a separate subsection) present a client-server architecture of the new Rust version.

A two-way communication between the client (a web browser) and the Rust server occurs over WebSockets. The WebSockets technology halves the network latency and is more efficient in handling small messages compared to traditional AJAX HTTP requests. On the server side, the Rust language binds together various C/C++ libraries for which there is no high-performance 100 % pure-Rust implementation available. In particular, the computation-intensive parts are SIMD-parallelised using the Intel SPMD Program Compiler[7]. Unfortunately the *no-crash* guarantees do not extend to non-Rust external libraries which may leak memory and may contain segmentation fault-causing defects. Programmers need to be very careful when choosing which C/C++ libraries to call from Rust.

Table 1: C/C++ versus Rust feature comparison.

C/C++	Rust
mutable by default (a **const** keyword is needed to prevent accidental data manipulation)	immutable by default (all variables are constants), an opt-in (let **mut** x = …) is needed to enable subsequent writes
variables can be written to by another thread without any synchronisation	threads/functions take ownership of variables (only one owner at a time can write)
by default a lax compiler, beware of unexpected compiler bugs	the Rust compiler (borrow checker) is extremely strict; initially it may take a long time (mental gymnastics) to get a code to compile
the compiler does not catch any common memory bugs, a programmer needs to maintain a high state of alertness at all times, external memory-checking tools like **valgrind** are needed	the strict compiler helps prevent many common programming mistakes, dangling pointers etc., resulting in safer programs containing fewer bugs
a fast auto-vectorised code either with the paid-orfor Intel C/C++/Fortran compiler or a free Intel SPMD Program Compiler **https://ispc.github.io/**	the default auto-vectorisation can be hit-or miss; easy integration with the Intel SPMD Program Compiler via an ispc-rs Rust crate (package): **https://github.com/Twinklebear/ispc-rs**
error/exception handling an after-thought; it is easy to skip error checks during prototyping and then omit/forget to add proper error handling during production	forces a programmer to decide how to handle errors at every step, resulting in more reliable programs
excellent Parallel STL with C++17/20, easy parallelism with OpenMP	excellent data parallelism library **Rayon** **https://github.com/rayon-rs/rayon**
using OpenCL for **GPGPU** may be a bit cumbersome (a lot of low-level plumbing)	low-level OpenCL complexity is hidden from the end-user in an easy-to-follow Rust ocl crate (package): **https://github.com/cogciprocate/ocl**
WebAssembly with Emscripten: **https://emscripten.org/**	native WebAssembly (Wasm) support: **https://github.com/raphamorim/wasm-and-rust**

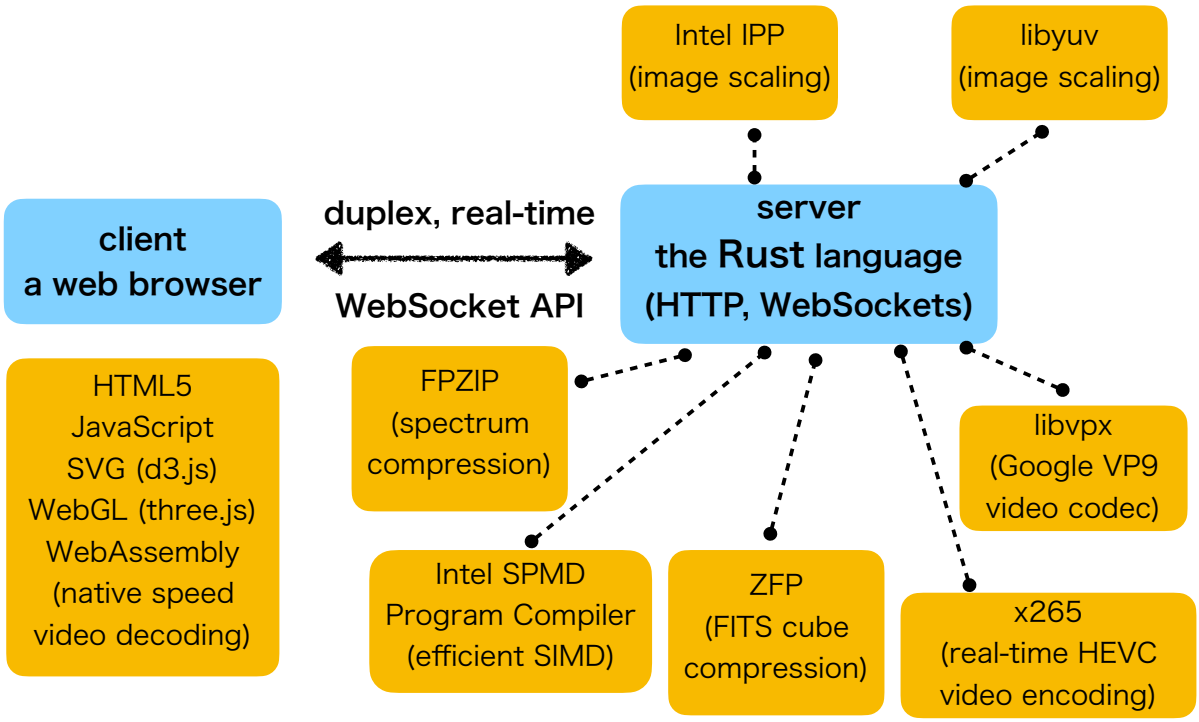

Figure 10: FITSWebQL v4 client-server architecture. This is an updated version of Figure 1 adopted from Zapart et al. (2019b).

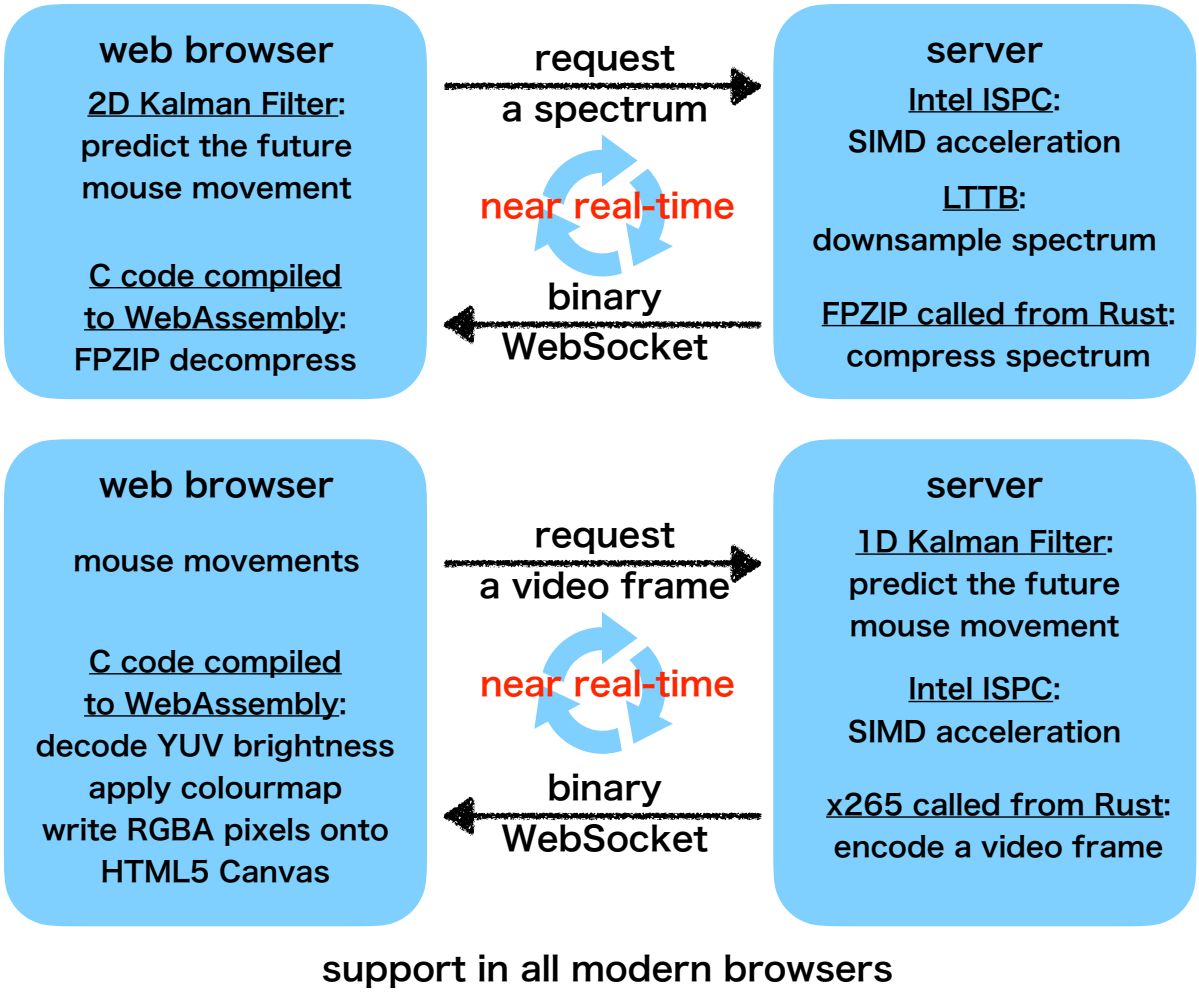

Figure 11: WebAssembly (Wasm) acceleration for a near-native speed execution in a web browser. This is an updated version of Figure 2 adopted from Zapart et al. (2019b).

On the client side we have taken advantage of the latest developments in browser technologies. Specifically we adopted WebAssembly (Wasm) that allows developers to compile C/C++ code to a binary Wasm stack machine code executed at a near-native speed inside a web browser[8]. In particular we have managed to accelerate greatly with WebAssembly the CPU-intensive parts: real-time decoding of HEVC video frames and the application of a user-specified colourmap to grey-scale video frames.

3.1 VP9 vs. HEVC comparison

During the initial development we originally intended to use the HEVC (via its x265 encoder library) codec to handle real-time video streams. However, finding a suitable JavaScript and/or Wasm decoder proved impossible. The resources freely available on the Internet did not meet our requirements. They were too outdated; they did not support the latest HEVC specification. As an alternative, after exploring other codecs i.e. Cisco's Thor, initially we integrated Google's VP9 libvpx library into our project. However, due to inferior multithreading capabilities of libvpx and codec inefficiencies compared with a superior HEVC solution, we decided to return to using HEVC/x265. Due to a difficulty in finding a suitable off-the-shelf browser-based HEVC decoder, we adapted the HEVC decoding part from the FFmpeg C library and compiled it to WebAssembly for fast native execution in a web browser.

As a result of this somewhat "convoluted" development process, as of now the VP9 library is still used to compress FITS 2D images (as VP9 still key-frames) for display in a browser whilst the more capable HEVC x265 library handles real-time video streaming. The Table 2 highlights the pros and cons of the two video codec formats.

3.2 Machine learning

Since FITSWebQL version 3 we have been employing rudimentary machine learning techniques in order to improve the end-user experience. One example is the use of a simple Kalman Filter (Kalman, 1960) to predict future mouse movements performed by end users. When a user moves a mouse over the main FITS image its screen coordinates are sent through a WebSocket connection to the JVO server in Mitaka, Japan. The server then calculates a spectrum corresponding to a current viewport and sends it back to the client browser over the network. The process works very well in real-time for users whose physical locations are in a close proximity to the JVO server. However, for bandwidth-constrained users located on another continent or for those with high-latency network connections, the round trip from the user computer to the server and back may well take about 500 ms or more. During the time is takes for the messages to travel to the server and back a user might have already moved a mouse away from the original position. This might result in a potential discrepancy: a user is looking at one location on the screen but the spectrum displayed in the browser corresponds to different (previous) location. As a mitigation measure we periodically monitor the communication latency between the client and the server. Then a client-side Kalman Filter coded in JavaScript is used to predict future mouse positions after taking into account the network

[7] The open-source Intel SPMD compiler (see https://ispc.github.io) should not be confused with the paid-for Intel C/C++/Fortran compiler suite.
[8] https://webassembly.org

Table 2: A side-by-side comparison of Google's VP9 and HEVC video codecs together with their corresponding C API libraries. This is an extended version of Table 1 adopted from Zapart et al. (2019b), with the licensing information added at the bottom.

Google's VP9 (libvpx)	HEVC (x265)
libvpx library: both an encoder and decoder	x265 library: only an encoder (search the Internet for a decoder to suit your task)
slower, less ecient encoding, inferior multithreading	faster than libvpx, more ecient (bandwidth-friendly), scales well across all CPU cores
no grey-scale (an overhead of handling redundant RGB/YUV channels)	YUV 4:0:0 support (server-encode as grey-scale, add colour in the client)
an easy API, trivial to compile the decoder into WebAssembly	extreme diculty finding a suitable JavaScript decoder (DIY: FFmpeg C API compiled to WebAssembly)
a royalty-free codec (no licensing issues)	expensive codec royalties hindered an early widespread adoption

latency. Only when a mouse comes to a complete stop do we submit real (not predicted) coordinates to the JVO server. The Kalman Filter — this time server-side — is used in a similar way when a mouse is moved over the bottom frequency axis and a video stream consisting of individual frequency channel 2D images is encoded in real time on the server and streamed back to the client browser. The Figure 11 illustrates the whole process.

In a deliberate effort to present users with images of FITS datasets that are both scientifically informative as well as pleasing to the eye, we have trained a small logistic regression classifier (Murphy, 2013) to provide an initial "optimum" setting for the image tone mapping function seen in the Figure 3. As illustrated in the Figure 12 the entire image histogram (1024 bins) is converted into an empirical cumulative distribution curve containing real values between 0 and 1. These values form 1024 inputs to the logistic regression model. There are five outputs corresponding to possible choices for the tone mapping function: *linear*, *logarithmic*, *logistic*, *ratio* and *square*. Whilst not perfect the model provides an adequate default setting for the majority of astronomical images. Moreover, users not satisfied with that initial setting can always override manually the automatic guess via the *Image* menu. The Figures 13 and 14 provide an example of this feature in action. We visually compare the default linear tone mapping function applied by the SAOImage DS9 image display and visualisation tool for astronomical data (Figure 13) against the ratio tone mapping automatically selected by our tool (Figure 14). Whilst SAOImage DS9 displays with clarity the high-intensity object of interest located at the centre of the image, our FITSWebQL software can also display the surrounding low-intensity background area, thus saving users the need to change manually the colour scaling settings in order to check whether or not something interesting (for example gas flows) lies around the main object. To be fair in most cases the default linear tone mapping function applied by SAOImage DS9 does a reasonable job of conveying the information contained in astronomical images. But the existence of many exceptions to this "rule" has prompted us to automate the tone mapping selection process.

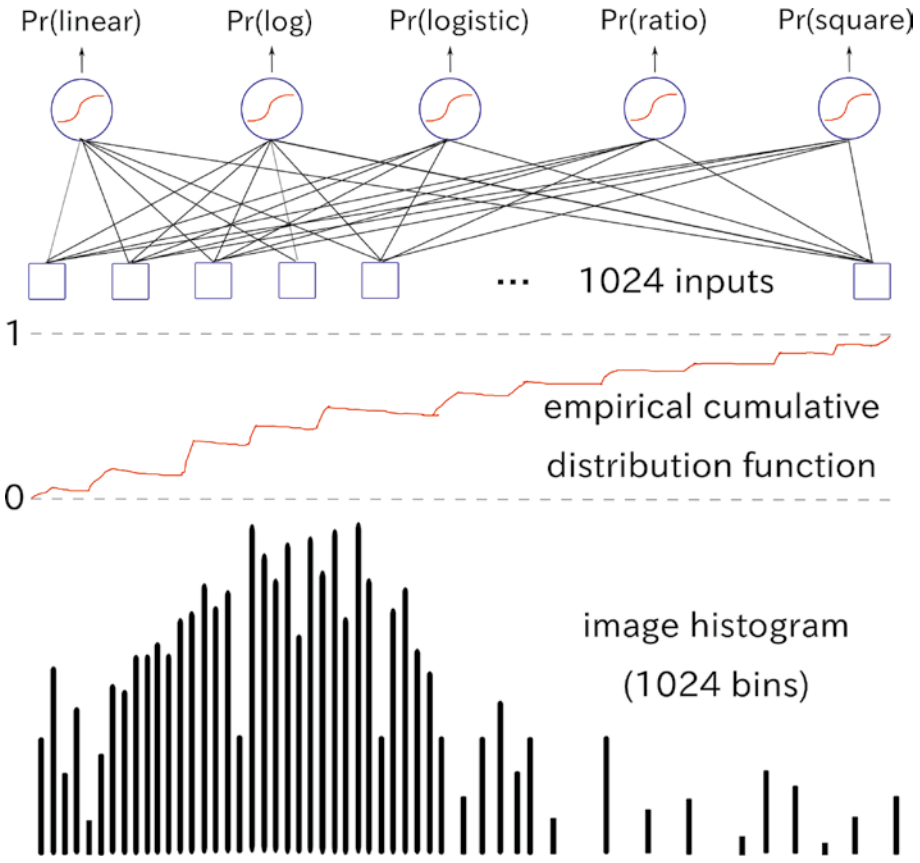

Figure 12: A simple logistic regression classifier. The probabilities of five choices for the image tone mapping function are predicted based on the empirical cumulative distribution function (cdf) derived from the image histogram. Note: the hand-drawn cdf curve is for illustration purposes only; it does not reflect a true empirical cumulative distribution function that would correspond to the histogram shown below it.

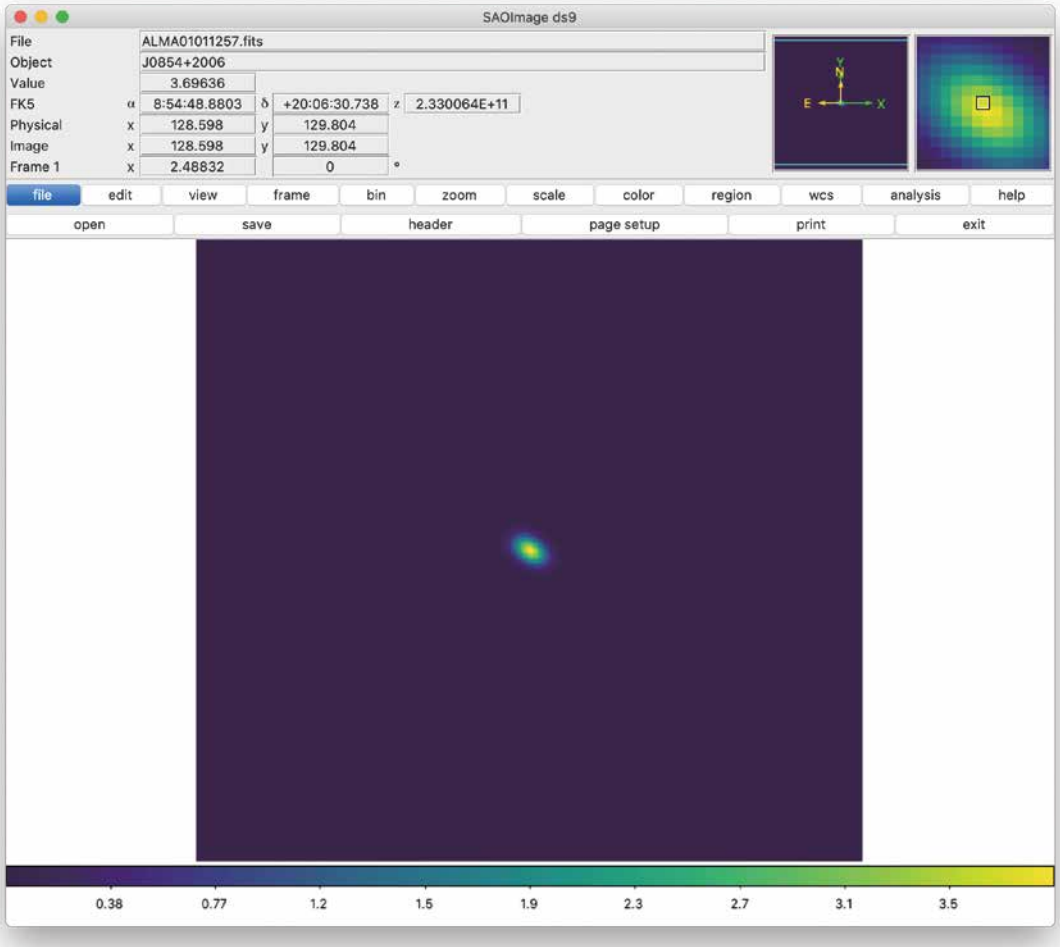

Figure 13: Visualisation of a sample ALMA dataset in SAOImage DS9 using a default linear colour scaling transformation. Object: J0854+2006

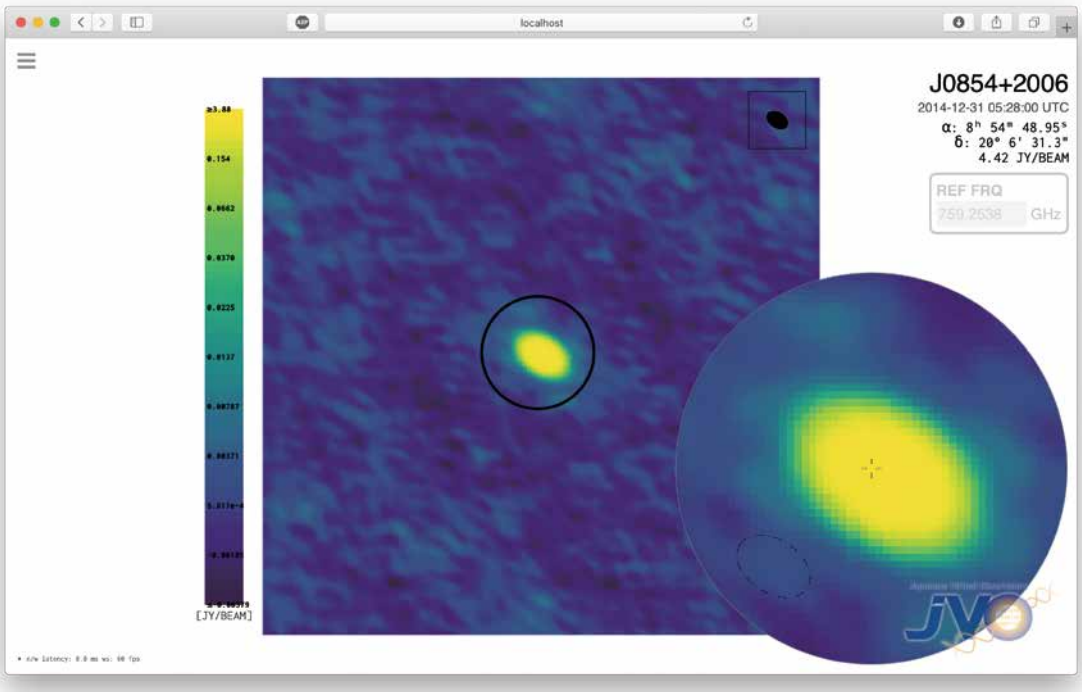

Figure 14: Visualisation of a sample ALMA dataset in FITSWebQL using a default ratio tone mapping function selected automatically by a rudimentary AI algorithm. Object: J0854+2006

Going one step further, if the FITSWebQL determines (based on the FITS header) that the underlying image comes from an optical telescope, and the selected tone mapping function (either predicted by the model or manually selected by a user) is set to *ratio*, a bi-section method is used to find automatically an optimum initial noise sensitivity setting (a slider in the Figure 3) such that the average image brightness on a scale between 0 and 1 is close to 0.1. This feature helps ensure that optical images containing bright stars "just look good". Again, the Figures 15 and 16 help underscore the effectiveness our software. A visual inspection of the Figure 15 — a default output from SAOImage DS9 — reveals a somewhat blurred or washed-out image of stars and galaxies. However, by automatically pre-selecting an appropriate tone mapping function and adapting automatically to the background noise, our FITSWebQL tool displays with a better clarity many more stars and galaxies (Figure 16).

3.3 Data compression

Introduced as one way[9] of dealing with ever growing FITS file sizes, the current version 4 of FITSWebQL uses a layered compression scheme. Ultimately the original float32 FITS data is held in RAM in a 16-bit half-float format, which means that displaying a 100 GB-large FITS file requires only 50GB of RAM. Half-float numbers can be converted to float32 very fast on modern CPUs. Moreover, some versions of AVX can perform in hardware arithmetic operations on a half-float data type. The next layer of compression is employed in the internal FITS file cache used by FITSWebQL. Here, 3D FITS data cubes are stored as ZFP fixed-rate compressed floating-point 2D arrays (Lindstrom, 2014): each 2D plane is compressed independently with a compression ratio of 4:1. Upon loading a FITS file into RAM the data is first ZFP-decompressed and converted into half-float. When a new FITS file is opened for the first time, it will be converted into half-float and then also compressed with ZFP for faster subsequent accesses directly from the FITSWebQL cache.

3.4 Distributed FITSWebQL

Compression alone — no matter how aggressive — probably cannot solve the problem of dealing with near-future Terabyte-class FITS files using present hardware. To this end we are experimenting with distributed computing, adding clustering capabilities to FITSWebQL so that many instances of it can run in parallel across a supercomputing cluster. Then each FITSWebQL instance would be responsible for handling only a small subset of a very large FITS data cube. After a careful consideration we have decided against using OpenMPI in favour of a custom solution involving a mixture of UDP (an automatic node discovery), HTTP and WebSockets[10]. The development work is already under way.

4. FITSWebQL Personal Edition

Due to a popular demand since version 3 we have been offering a standalone desktop edition of FITSWebQL. After the installation (a rather time consuming process involving compiling a lot of software) users can access their own FITS files from a local web page — shown in the Figure 17 — and view them through an exactly the same user interface as FITSWebQL from the JVO Portal. Both the server and local editions share the underlying code base; both clients run in a web browser. The Personal Edition runs across three major platforms: Linux, macOS and Windows 10 WSL. Readers can access the source code on GitHub without any restrictions at https://github.com/jvo203/fits_web_ql.

5. Final Remarks

The paper gives an overview of the two most recent major FITSWebQL releases: version 3 coded in C/C++ and the Rust version 4. C/C++ offers the best overall performance but it does so at the price of potential memory bugs and data races that can be difficult and/or time-consuming to debug, especially in a concurrent multi-threaded environment. A C/C++ programmer always needs to maintain a state of high concentration during work so as to prevent introducing bugs. On the other hand, Rust offers stability and performance with fewer bugs to start with. Learning Rust can often teach a programmer to write better, safer C/C++ and acquire better programming habits. Overall the JVO experience with Rust has been positive although the Rust compiler can be painful to work with initially. Especially when learning Rust it is easy to "hit a wall" and get stuck, keep searching for a suitable solution for several days. On a positive note, Rust offers excellent

[9] Another way is to use distributed computing and/or a clustered file system.
[10] The final solution may change.

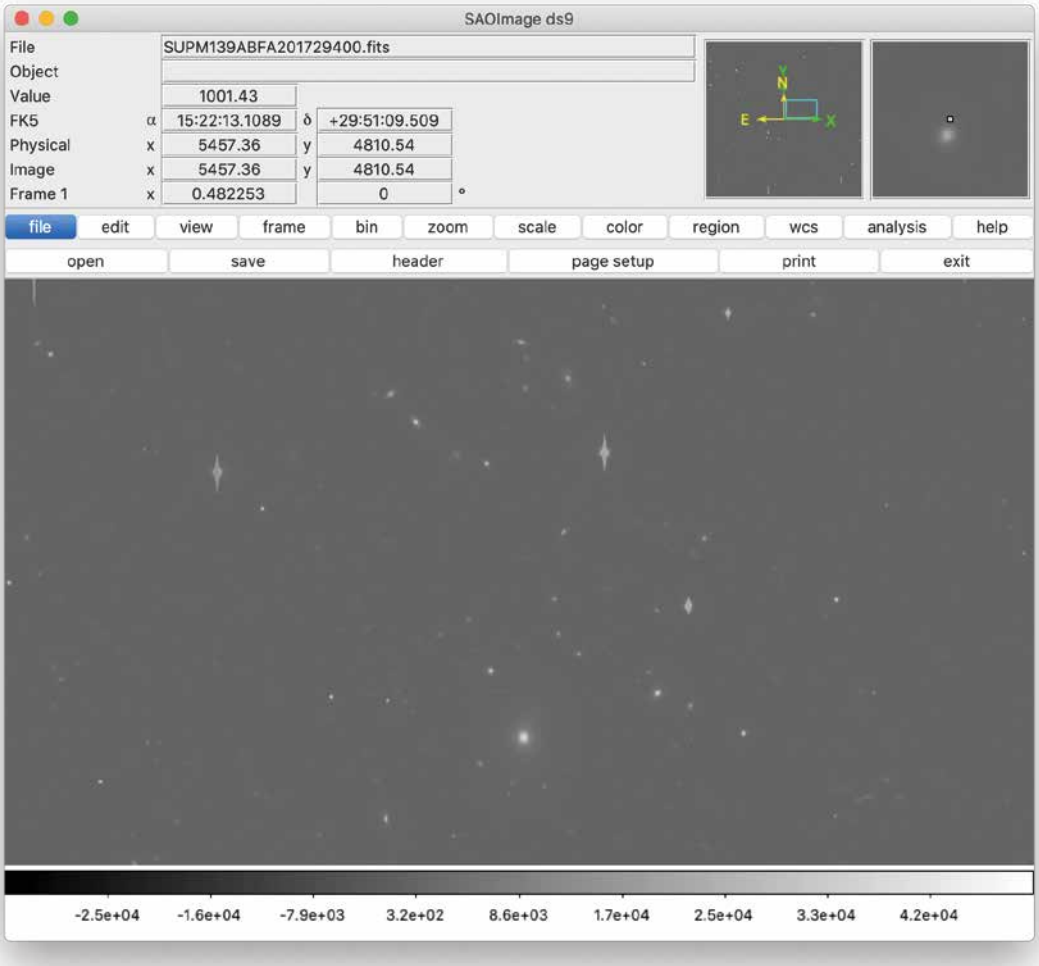

Figure 15: A representative optical dataset with stars and galaxies as viewed in SAOImage DS9 using its default linear colour scaling function. Telescope: Subaru Prime Focus Camera

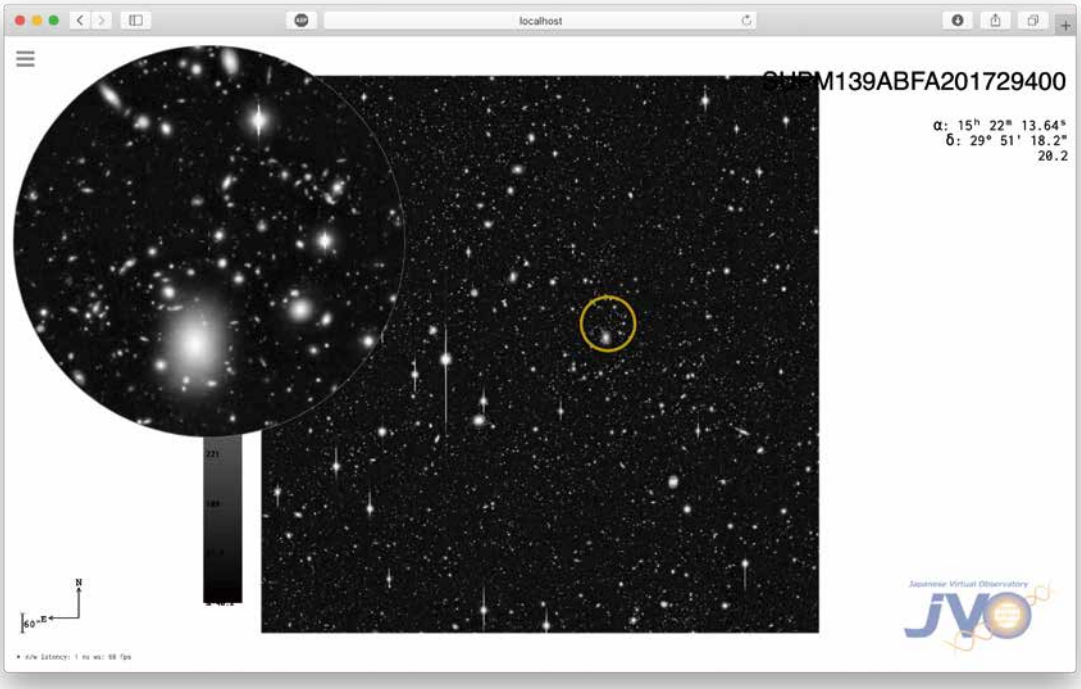

Figure 16: A representative optical dataset with stars and galaxies displayed using FITSWebQL. The tool automatically selected a ratio tone mapping function and adjusted the noise sensitivity too. Telescope: Subaru Prime Focus Camera

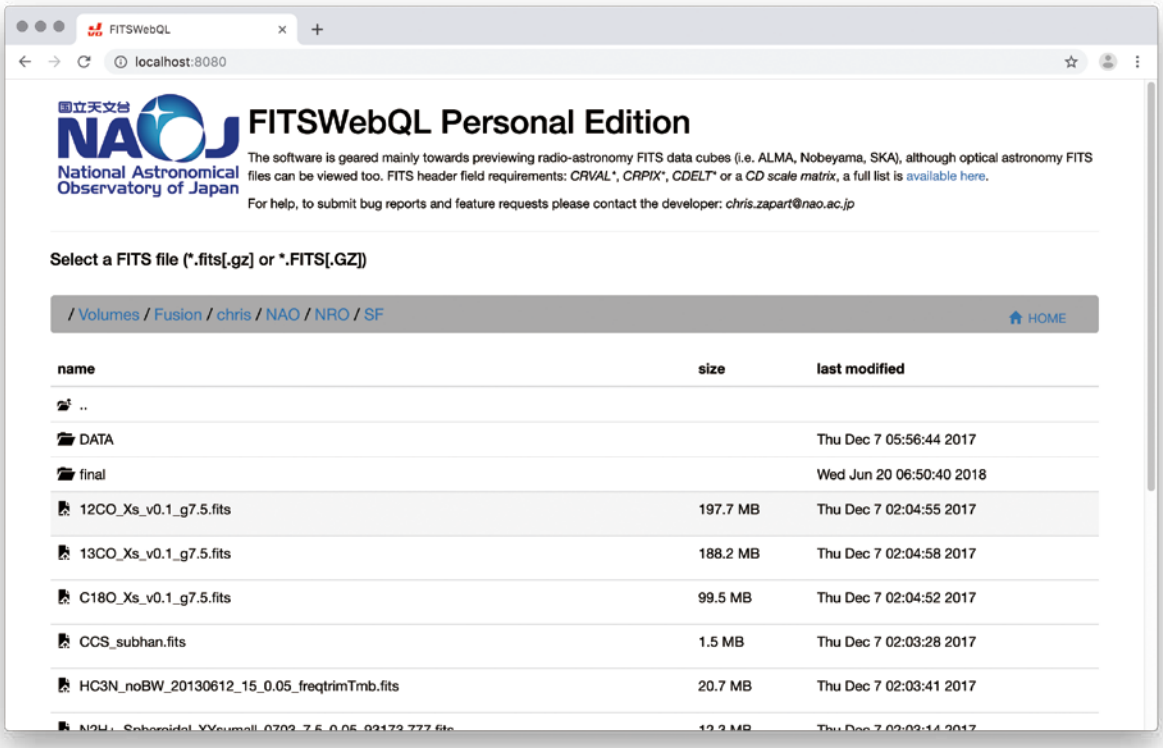

Figure 17: FITSWebQL Personal Edition.

documentation and tutorials. We encourage interested readers to consult the "Rust Book" available at

https://doc.rust-lang.org/stable/book/.

Looking forward to the next-generation FITSWebQL capable of handling Terabyte-class FITS files, the author has been split between using Rust on one hand and returning to C/C++ on the other. On one hand the current Rust FITSWebQL v4 continues to be updated with new features (for example a recent integration of the Intel Integrated Performance Primitives library, FPZIP Lindstrom and Isenburg (2006) spectrum compression, spectrum downsampling with Largest-Triangle-Three-Buckets (LTTB) Steinarsson (2013), or clustering capabilities under development). On the other hand for some time the author has been developing an experimental high-performance C/C++ FITSWebQL SE[11] in order to test various compression ideas. The readers can access its incomplete source code at

https://github.com/jvo203/FITSWebQL.

The decision whether to stick with Rust or return to C/C++ is not an easy one. From the developer's point of view the recently standardised 2017 C++ and its 2020 successor offer attractive alternatives to Rust. Whilst reference counted C++ smart pointers for automatic memory tracking/release (*std::shared_ptr*) have been available since C++11, the more recent C++ standards also introduce Parallel STL containers as well as coroutines (lightweight threads). Especially the use of lambda functions passed to the destructor argument of the *std::shared_ptr* allows a programmer to use flexible data structures at runtime, in which the underlying pointer can either point to a RAM region or to an *mmapped* disk file. C/C++ also benefits from thread-safe lock-less data structures available in the Boost library. A skillful and **very careful** use of a modern C++ can result in memory-safe programs nearly on par with Rust, all whilst taking advantage of the mature C/C++ ecosystem. Indeed a major problem — a showstopper for us — encountered whilst adding clustering capabilities to the Rust version 4 of FITSWebQL has been incomplete and/or unstable Rust bindings to message passing libraries like ZeroMQ/czmq or nanomsg. Another issue is a relative immaturity of the Rust ecosystem (this will likely improve with time). A look at the Figure 10 reveals that the application uses Rust mainly to handle the networking part and for general memory housekeeping. All the vital computationally intensive parts are in fact mainly handled by external C/C++ libraries called from within Rust. At the time of writing there were no pure Rust equivalents to these

[11] "SE" stands for **S**upercomputer **E**dition.

libraries. An inquisitive reader could therefore question the point of using Rust: would it not be less troublesome to call native C/C++ libraries from within C++ for an end-to-end integrated native solution? Given the aforementioned shortcomings of Rust in the area of message passing between supercomputing cluster nodes, with a bit of a "heavy heart" we have taken the decision to return to C/C++ for the FITSWebQL Supercomputer Edition.

Acknowledgement

The authors would like to thank Vale T. Kobayashi for his in-numerous suggestions and an eye for detail in finding bugs, especially during the development phase of the prior ALMA WebQL v2.

References

Allen, M. G., Baines, D., Bunn, S. E., Cui, C., Taylor, M., Zolotukhin, I.: 2013, ALMA VO Service. IVOA Newsletter .

Eguchi, S., Kawasaki, W., Shirasaki, Y., Komiya, Y., Kosugi, G., Ohishi, M., Mizumoto, Y.: 2013, Prototype Implementation of Web and Desktop Applications for ALMA Science Verification Data and the Lessons Learned, in: Friedel, D.N. (Ed.), Astronomical Data Analysis Software and Systems XXII, p. 255. 1211.3790.

Kalman, R. E.: 1960, A New Approach to Linear Filtering and Prediction Problems, *Journal of Fluids Engineering*, **82**, 35–45.

Lindstrom, P.: 2014, Fixed-rate compressed floating-point arrays, *IEEE Transactions on Visualization and Computer Graphics*, **20**.

Lindstrom, P., Isenburg, M.: 2006, Fast and efficient compression of floating point data. *IEEE transactions on visualization and computer graphics*, **12**, 1245–1250.

Lovas, F. J.: 2009, Revision. NIST recommended rest frequencies for observed interstellar molecular microwave transitions. https://dx.doi.org/10.18434/T4JP4Q

Maple, C.: 2003, Geometric design and space planning using the marching squares and marching cube algorithms, Proceedings of 2003 International Conference on Geometric Modeling and Graphics, 90–95.

Markwick-Kemper, A. J., Remijan, A. J., Fomalont, E.: 2006, The Splatalogue (Spectral Line Catalogue) and Calibase (Calibration Source Database), in: American Astronomical Society Meeting Abstracts #208, p. 130.

Murphy, K. P.: 2013. Machine learning: a probabilistic perspective. MIT Press, Cambridge, Mass. [u.a.].

Steinarsson, S.: 2013, Downsampling time series for visual representation. https://github.com/sveinn-steinarsson/flot-downsample

Zapart, C., Shirasaki, Y., Ohishi, M., Mizumoto, Y., Kawasaki, W., Kobayashi, T., Kosugi, G., Eguchi, S.: 2019a, ALMAWebQL v2: a Modern Interactive Client-server Architecture for Fast Previewing of Large ALMA Datasets, in: Molinaro, M., Shortridge, K., Pasian, F. (Eds.), Astronomical Data Analysis Software and Systems XXVI, p. 753.

Zapart, C., Shirasaki, Y., Ohishi, M., Mizumoto, Y., Kawasaki, W., Kobayashi, T., Kosugi, G., Morita, E., Yoshino, A., Eguchi, S.: 2019b, An Introduction to FITSWebQL, in: Teuben, P. J., Pound, M. W., Thomas, B. A., Warner, E. M. (Eds.), Astronomical Data Analysis Software and Systems XXVIII, p. 13. 1812.05787.