Aladin Lite v3: Behind the Scenes of a Major Overhaul

Matthieu Baumann, Thomas Boch, François-Xavier Pineau, Pierre Fernique, Caroline Bot, Mark Allen

Aladin Lite v3: behind the scenes of a major overhaul

Matthieu Baumann[1] and Thomas Boch[1]

[1] *Université de Strasbourg, CNRS, Observatoire astronomique de Strasbourg, Strasbourg, France;* matthieu.baumann@astro.unistra.fr
thomas.boch@astro.unistra.fr

Abstract. Since its first version in 2013, Aladin Lite has gained significant traction and usage as an HiPS viewer running in the browser. Designed to be easy to embed, it is now used in more than fifty websites and portals in the professional astronomy community. Aladin Lite has been adopted as the sky visualisation component of popular applications: ESA Sky, ESO Science Archive or ALMA Science Archive.

We present a major overhaul of Aladin Lite taking advantage of the GPU with WebGL, and which responds to requests of users, developers and integrators in a context where browser-based applications and science analysis platforms are increasingly important.

While keeping the strengths of the original code, Aladin version 3 will introduce several new features: support of multiple projections (Aitoff, Mollweide, Orthographic, Mercator), support of FITS tiles, display of FITS images, heatmaps visualisation of catalogue data, improved rendering pipeline and coordinates grids. We will give an overview on the architecture used to develop these new functionalities, based on existing Rust code transpiled to WebAssembly, a portable high-performance low-level bytecode for the web supported in all modern browsers. We will also outline the technical challenges and limitations we encountered. Short video footage sequences will demonstrate the existing prototype throughout the presentation.

These improvements have been partially supported by the ESCAPE project and will also benefit to ipyaladin, the widget enabling the usage of Aladin Lite in Jupyter notebooks.

1. Features details

1.1. Multiple Screen Projections

The support of multiple screen projections, especially all sky ones have been implemented. This includes Hammer-Aitoff, Arc, Gnomonic (i.e. TAN), Mercator and Mollweide projections. These projections are not rendered tile by tile as in many sky/map viewers, instead it uses a ray-tracing-like approach. Each pixel colour is computed independently from its surrounding neighbours. This condition of independence makes the use of GPUs very adapted. For each pixel of the screen, the algorithm follows:

1. Unproject the **screen** pixel coordinates to get its **world** position with respect to the projection chosen. $\mathcal{P}^{-1}(s_x, s_y) = (w_x, w_y, w_z)^T$

2. Compute the HEALPix cell index h_i where the world position is located. δ_x δ_y offset positions within the cell are also computed.

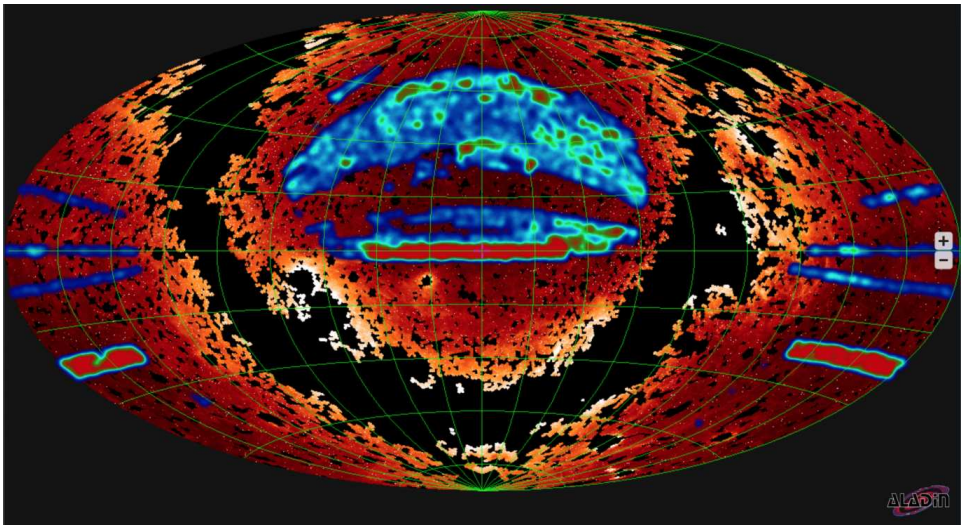

Figure 1. Aladin Lite with all its new coming features

3. Retrieve the good tile texture based on the HEALPix cell index h_i computed during the previous step.

4. Within the tile texture, get the pixel colour located at δ_x δ_y inside the texture.

1.2. Blending of Multiple Image Surveys

Aladin Lite, like Aladin desktop, renders image surveys formatted by the HiPS (Hierarchical image Progressive Survey) standard (Fernique et al. 2017). The coming overhaul of Aladin Lite enables the rendering of FITS tile HiPSes thanks to a basic FITS parser implemented in Rust using the **nom** crate. The support of FITS tiles is a great improvement, allowing the user to see much more dynamic in those images than given in any compressed ones (JPG, PNG, ...).

On top of that, thanks to WebGL and as the tile textures are available from the GPU, it is possible to change the screen pixels in real-time. Common operations are in order:

1. cutout parameters tuning (fig 2), allowing to see more background, change the contrast/luminosity.

2. transfer function correction of the raw values giving grayscale pixel values (fig 3).

3. color mapping of the grayscale pixel values (fig 4).

Finally, as image surveys are often given per frequency bands, one is now able to blend multiple surveys on the client and in real time!

1.3. Up to 1 Million Rows Catalogue Rendering

GPUs are very efficient for doing repetitive rendering tasks. Common APIs such as OpenGL, WebGL and more recently WebGPU feature "instancing" techniques boosting

Figure 2. Effect of cutout parameters. *Left:* Too much background *Center:* Good balance *Right:* Higher details at the center of galaxy but less brightness

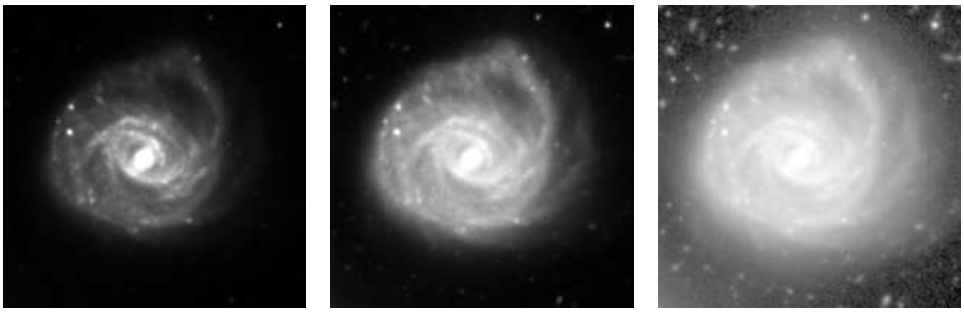

Figure 3. Transfer function effect on the raw values. *Left:* $\mathcal{H} : x \rightarrow x$ *Center:* $\mathcal{H} : x \rightarrow \frac{sinh^{-1}(10 \times x)}{3}$ *Right:* $\mathcal{H} : x \rightarrow \frac{\ln(1000 \times x+1)}{\ln(1000)}$

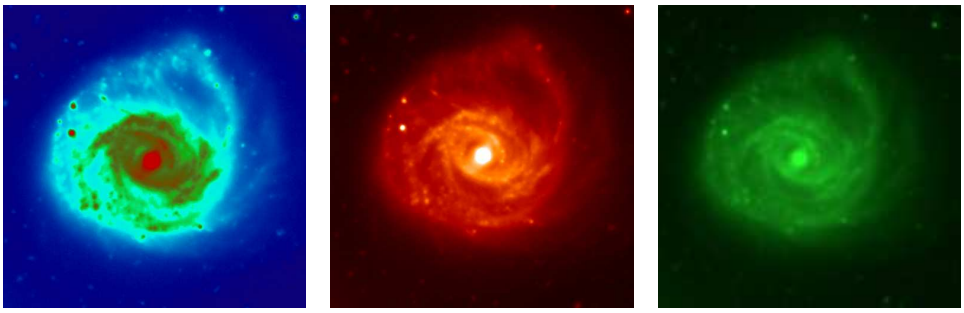

Figure 4. Grayscale values mapped to a colormap. *Left:* Rainbow *Center:* Red Temperature *Right:* Constant green

the performance of rendering a specific mesh at different locations. This is very adapted for rendering simple primitives a thousand times such as simple catalogue sources. The catalogue rendering implements the following features:

- Catalogue sources downloaded through an async JavaScript TAP query.
- HEALPix index array built from the position array, allowing fast and cache-resident retrievals of the sources lying in the field of view (Górski et al. 2005).
- Heat-map visualisation of the sources making visible high density spots.

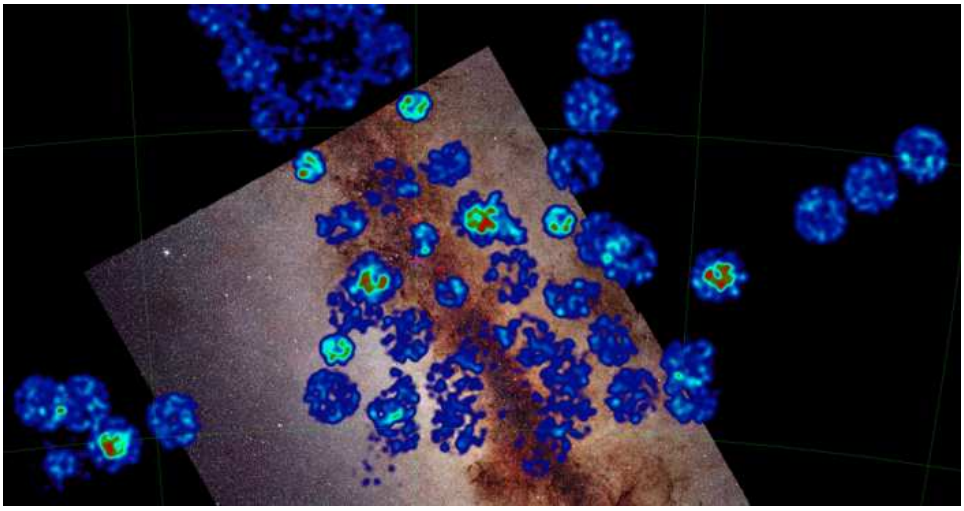

Figure 5. APOGEE catalogue sources heat-map, 183232 sources

2. Rust, WebAssembly and WebGL

The Rust system programming language has been chosen for this project. Rust is a compiled, garbage collector free language. In that sense, it is similar to C++ but relies heavily on the RAII principle which is part of the Rust compiler. Ownership and borrowing rules have been added to the Rust compiler, making Rust its very unique signature. The compilation phase will hence prevent you from e.g. returning a reference to a resource owned by the current function, ...

On top of that, Rust can be compiled to Web-Assembly, a portable bytecode designed for high performance applications. Web-Assembly standard is part of the W3C making it reliable for developing applications for the long term support.

The **wasm_bindgen** crate makes the interfacing with the JavaScript sandbox possible. From Rust, one can export functions/classes to JavaScript, manipulate the DOM and access the WebGL API.

The WebGL API makes available your GPU from the browser. It offers an API similar to OpenGL and is supported by 98-99% of user's browsers.

Acknowledgments. This work has been partly supported by the ESCAPE project (the European Science Cluster of Astronomy & Particle Physics ESFRI Research Infrastructures) that has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement n. 824064.

References

Fernique, P., Allen, M., Boch, T., Donaldson, T., Durand, D., Ebisawa, K., Michel, L., Salgado, J., & Stoehr, F. 2017, HiPS - Hierarchical Progressive Survey Version 1.0, IVOA Recommendation 19 May 2017. 1708.09704

Górski, K. M., Hivon, E., Banday, A. J., Wandelt, B. D., Hansen, F. K., Reinecke, M., & Bartelmann, M. 2005, ApJ, 622, 759. astro-ph/0409513