Astronomical Data Analysis Software and Systems XXVIII
ASP Conference Series, Vol. 523
P.J. Teuben, M.W. Pound, B.A. Thomas, and E.M. Warner, eds.

The CDS HEALPix Library

Francois-Xavier Pineau and Pierre Fernique

Université de Strasbourg, CNRS, Observatoire astronomique de Strasbourg, UMR 7550, F-67000 Strasbourg, France;
francois-xavier.pineau@astro.unistra.fr

Abstract. The CDS is releasing a new HEALPix library implemented in Java, Rust and WebAssembly. The library focuses on the CDS needs, on performances and accuracy and is distributed under the 3-clause BSD license. Aladin desktop has already started to integrate the Java version of the library in its code. The current state of the library and its specific features are presented in this article.

1. Motivations

The motivations bringing the CDS to develop an HEALPix library from scratch are diverse. First of all, we wanted to develop an internal expertise in a key component of both CDS services and the HiPS IVOA standard. It will allow us to improve the HEALPix support in Aladin Lite by bringing the deepest addressable resolution from order 13 to order 24 and to add support for polygons. We also wanted to control the license to be able to switch from the GPL ("official library") to the 3-clause BSD license ("CDS library"). The 3-clause BSD will allow us to change the Aladin Lite license and to be compatible with, for example, the Astropy one. Finally, by writing our own library, making changes fitting with the CDS needs (mainly in Aladin, Aladin Lite and the cross-match service) will be easier.

2. Languages

We considered different programming languages for this library: Java, which is widely used at CDS; Rust, which we wanted to test and has a good support for WebAssembly; and C, which is often supported by large projects for the development of external modules.

2.1. Java

Since it is a popular language widely used at CDS – Aladin, SIMBAD and the cross-match service are fully written in Java – the HEALPix library has been developed first in Java. All features mentioned in this document were made available in the Java version of the library.

2.2. Rust and WebAssembly

Rust is a recent and promising open-source language sponsored by Mozilla which *pursues the trifecta: safety, concurrency, and speed* (Rust weekly newsletter). It has the additional aim to offer *high-level ergonomics and low-level control* (online Rust book). Since it is a compiled language, we use Rust to generate both WebAssembly files and static or dynamic libraries that can be called from Python or PostgreSQL. So far, mainly basic HEALPix features meeting with the Aladin Lite needs have been implemented in Rust (cell number from coordinates, cell center, cell vertices, cell neighbors, approximate cells-in-cone, projection/de-projection).

WebAssembly is a bytecode standardized by the W3C and compatible with all recent Web browsers. It aims to complement Javascript by providing better performance, and can be generated from compiled languages like C, C++ or Rust. We mainly target WebAssembly for Aladin Lite, but the library may also meet with the needs of other web applications using HEALPix.

2.3. The C programming language?

Rust pre-compiled binaries are similar to C and could be distributed in software like Astropy. However, installing Rust tools is necessary if a user wants to manually compile a module from the source code. Integrating Rust code into large projects like Astropy or PostgreSQL modules is thus not straightforward. This issue may bring us to develop a C version of the library.

3. Features

The features implemented in the HEALPix library meet the specific needs of CDS tools. So far, the code does not support the RING scheme – but it offers functions converting a NESTED cell number into a RING cell number and vice-versa – and Fast Fourier Transforms which are extensively used in the cosmology community. It does support a number of interesting features beyond a basic set. We mention here but a few.

3.1. BMOC for cone and polygon queries

A BMOC is an extension of a MOC storing for each cell an additional status flag telling if the cell is either partially or fully covered by the area the MOC represents. This binary coverage information allows avoiding useless time-consuming distance computations when performing cone-search queries on a table: the sources inside a cell "fully" covered by the cone does not have to be tested.

The library offers an approximated and an exact "cells-overlapped-by-cone" solution. In both cases the result is a BMOC. The exact solution does not contain false positive cells and it thus avoids possibly useless distance computations and disk accesses.
In addition, the library offers a very-fast – but with large approximations – cells-overlapped-by-cone function dedicated to map/reduce based cross-matches.

The library also supports self intersecting polygons of any size providing a BMOC as a result. The algorithm so far resorts to an approximation: it considers a cell border between two vertices as if it was a great-circle arc. Although an exact solution is possible it would be computationally less efficient.

3.2. Other features

The library supports an internal projection/de-projection in addition to a version compatible with the WCS HPX projection. We recall that a projection/de-projection consists in computing Euclidean from spherical coordinates and vice-versa.

Possibly useful for cross-match applications the library also provides: a function computing an upper limit on the largest center-to-vertex distance depending both on the order and on the position of the cell on the sky, and an ordered list of small cells surrounding a larger cell to take into account border effects while ensuring the sequential access to data stored on spinning disks, etc.

4. Technical details

We provide in this section a few key mathematical elements of the library internals.

4.1. Projection: simplified equations

HEALPix is quite extensively described in Calabretta (2004), Górski et al. (2005), Calabretta & Roukema (2007) and Reinecke & Hivon (2015). It is first of all an equal-area projection composed from two other projections. Internally we have chosen a projection scale such that all coordinates in the projection plane are $\in [0, 8[$ on the X-axis and $\in [-2, 2]$ on the Y-axis. The internal simplified equations are:

Cylindrical equal-area projection in the equatorial region:

$$\boxed{\begin{cases} X &= \alpha \times \frac{4}{\pi} \\ Y &= \sin(\delta) \times \frac{3}{2} \end{cases}} \Rightarrow \begin{cases} \alpha \in [0, 2\pi] & \rightsquigarrow & X \in [0, 8] \\ \sin\delta \in [-\frac{2}{3}, \frac{2}{3}] & \rightsquigarrow & Y \in [-1, 1] \end{cases}$$

Collignon (pseudo-cylindrical equal-area) projection in the polar caps, for $\alpha \in [0, \pi/2]$ and $\sin\delta > 3/2$:

$$\boxed{\begin{cases} t &= \sqrt{3(1-\sin\delta)} \\ X &= (\alpha\frac{4}{\pi} - 1)t + 1 \\ Y &= 2 - t \end{cases}} \Rightarrow \begin{cases} \alpha \in [0, \frac{\pi}{2}] & & t \in [0, 1[\\ \sin\delta \in]\frac{2}{3}, 1] & \rightsquigarrow & X \in]0, 2[\\ & & Y \in]1, 2] \end{cases}$$

4.2. Precision at poles

The formula $t = \sqrt{3(1-\sin\delta)}$ causes non-negligible numerical inaccuracies near the poles due to the $1 - \sin\delta$ expression it contains: in facts, $\arcsin(1 - 1.0 \times 10^{-15}) \approx 89.99999919$ deg, and $\frac{\pi}{2} - \arcsin(1 - 1.0 \times 10^{-15}) \approx 2.917$ mas.

We thus replaced the previous equation by the equivalent but numerically stable form: $t = \sqrt{6}\cos(\frac{\delta}{2} + \frac{\pi}{4})$. This form is also computationally less expensive since we spare a time-consuming square-root operation (the square-root applying here on a constant instead of a variable).

4.3. The exact cells-in-cone solution

Basically, if the four vertices of a cell are inside a cone then the cell is fully overlapped by the cone. If at least one vertex is inside the cone and one vertex is outside then

the cell is partially overlapped by the cone, but the cone also contains 4 special points such that the slope of the tangent line to the projected cone on that point equals plus or minus one. If a cell contains such a point then it is also partially overlapped by the cone. Finally, for large cells containing no vertices and no special points, we have to test if the center of the cone is inside the cell.

To compute the coordinates of the four "special" points, we first use the Haversine formula to get an accurate cone expression at small radii:

$$\Delta\alpha = 2\arcsin\left(\sqrt{\frac{\sin^2\frac{\theta}{2} - \sin^2\frac{\delta-\delta_0}{2}}{\cos\delta_0\cos\delta}}\right).$$

In the equatorial region, the equation of tangent lines is:

$$\frac{\mathrm{d}\Delta X}{\mathrm{d}Y} = \frac{\mathrm{d}\Delta X}{\mathrm{d}\Delta\alpha}\frac{\mathrm{d}\Delta\alpha}{\mathrm{d}\delta}\frac{\mathrm{d}\delta}{\mathrm{d}z}\frac{\mathrm{d}z}{\mathrm{d}Y} = \pm 1.$$

The projection formulae $z = \sin\delta$, $X = 4/\pi\alpha$, $Y = 3/2z$ lead to $\frac{\mathrm{d}\Delta X}{\mathrm{d}\Delta\alpha} = 4/\pi$, $\frac{\mathrm{d}\delta}{\mathrm{d}z} = \frac{1}{\cos\delta}$, $\frac{\mathrm{d}z}{\mathrm{d}Y} = 2/3$ and, finally, we find the special points latitudes by solving numerically:

$$\boxed{\frac{1}{\cos\delta}\frac{\mathrm{d}\Delta\alpha(\delta)}{\mathrm{d}\delta} \mp \frac{3\pi}{8} = 0}.$$

In the polar caps, the equation of tangent lines is:

$$\frac{\mathrm{d}\Delta X}{\mathrm{d}Y} = \frac{\mathrm{d}\Delta X}{\mathrm{d}\delta}\frac{\mathrm{d}\delta}{\mathrm{d}z}\frac{\mathrm{d}z}{\mathrm{d}Y} = \pm 1$$

with $z = \sin\delta$, $t = \sqrt{3(1-z)} = \sqrt{6}\cos(\frac{\delta}{2} + \frac{\pi}{4})$, $X = (\frac{4}{\pi}\alpha - 1)t + 1$, $Y = 2 - t$, leading to $\frac{\mathrm{d}\delta}{\mathrm{d}z} = \frac{1}{\cos\delta}$, $\frac{\mathrm{d}z}{\mathrm{d}Y} = \frac{2}{3}t$ and, finally, we find the special points latitudes by solving numerically:

$$\boxed{\frac{t(\delta)}{\cos\delta}\frac{\mathrm{d}}{\mathrm{d}\delta}\left[(\frac{4}{\pi}\alpha(\delta) - 1)t(\delta)\right] \mp \frac{3}{2} = 0}.$$

5. Conclusion

The CDS has been developing a new HEALPix library which has a permissive license and contains innovative features like the BMOC. Most of the features have been tested and the new Java library is been replacing the "standard" one in Aladin. The library is publicly available on GitHub.

References

Calabretta, M. R. 2004, ArXiv Astrophysics e-prints. astro-ph/0412607
Calabretta, M. R., & Roukema, B. F. 2007, MNRAS, 381, 865
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
Reinecke, M., & Hivon, E. 2015, A&A, 580, A132. 1505.04632