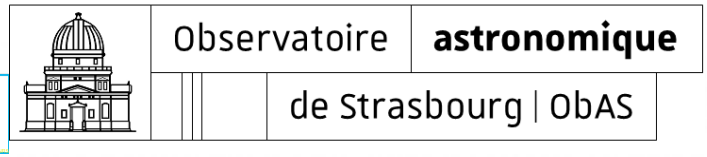
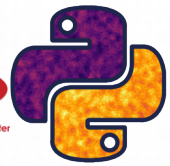
CDS HEALPix libraries

F.-X. Pineau[1]

[1] CDS, Observatoire Astronomique de Strasbourg

PyGamma19 workshop - Heidelberg, 18-22th March, 2019

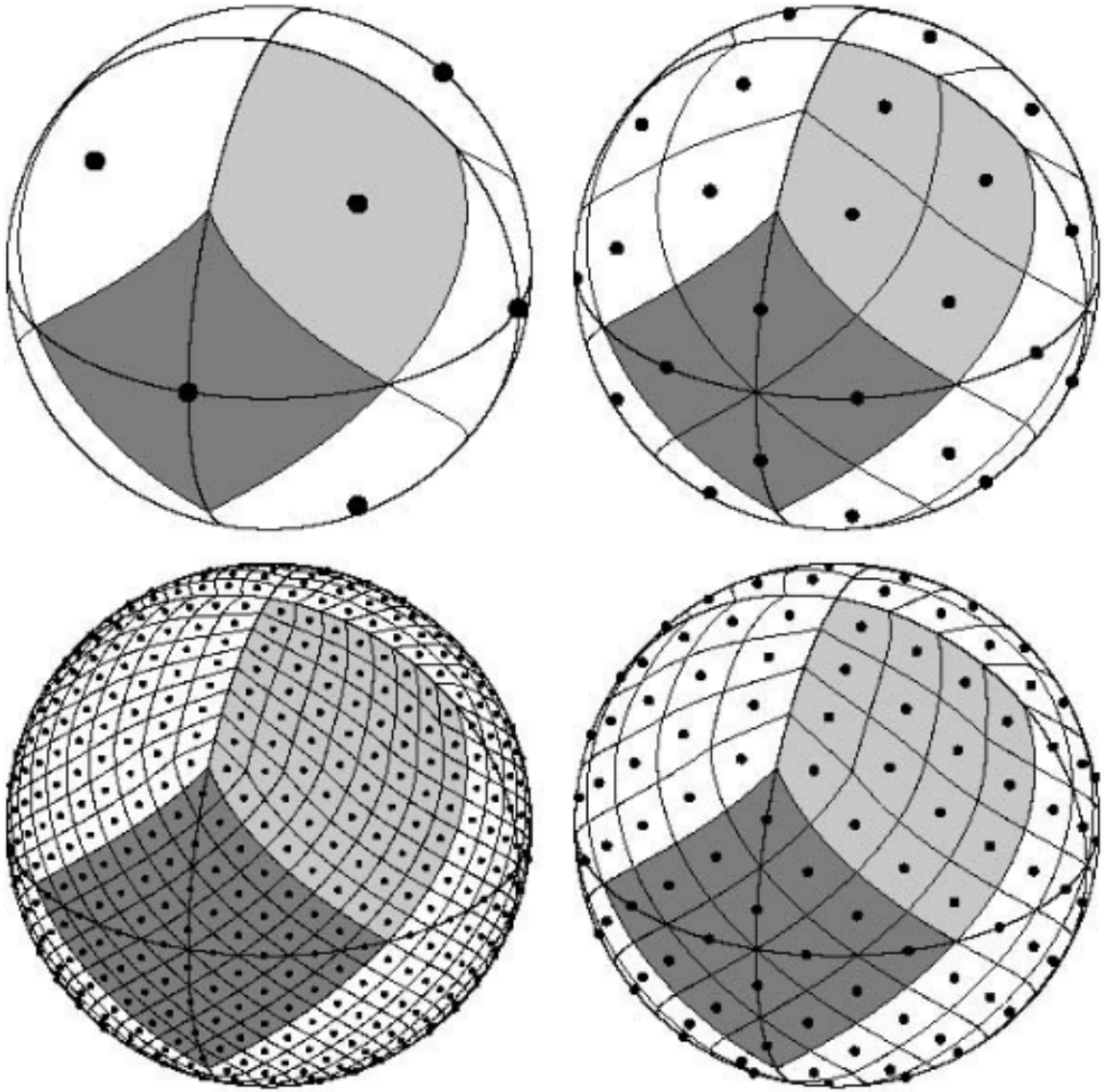




PyGamma19

Table of content

Introduction

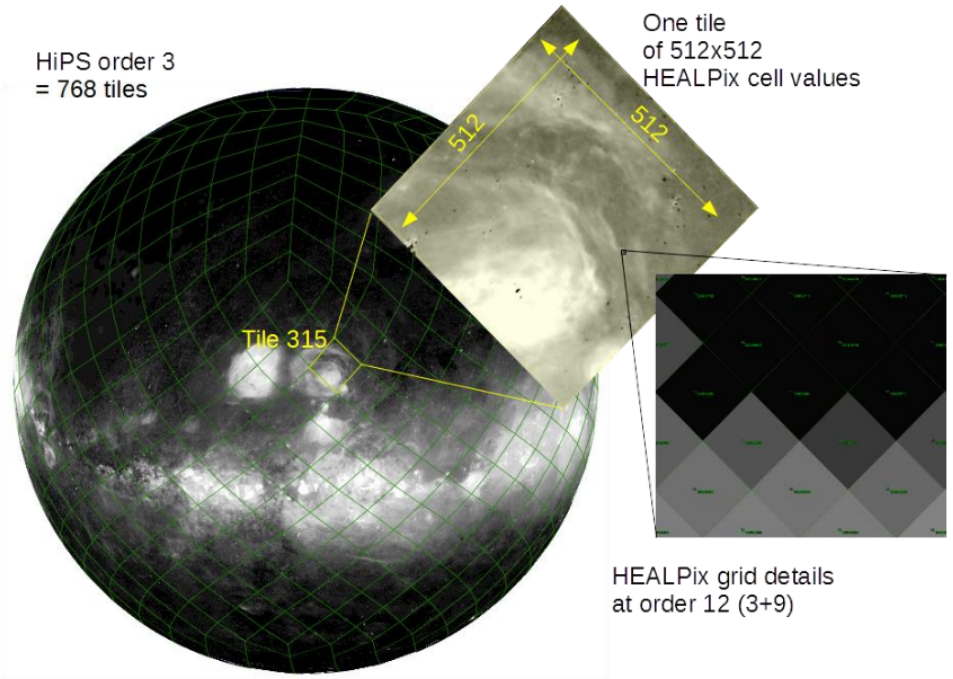


Credits: Górski et al. 2005 (2005ApJ...622..759G)

Introduction

Two IVOA standards: HiPS and MOC.

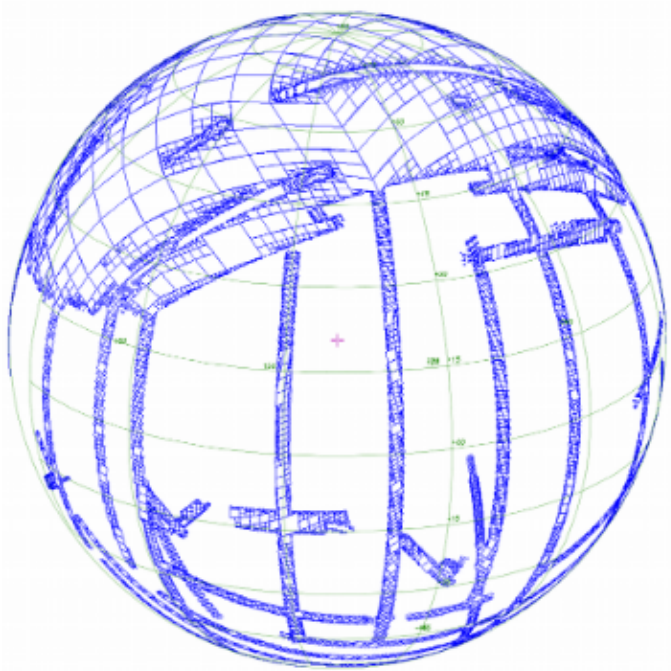
Hierarchical Progressive Survey



6 - *The HiPS Halpha Finkbeiner survey at HiPS order 3 (768 tiles) with HiPS tiles of 512x512 HEALPix cell values (51.53" resolution)*

Credits: `http://www.ivoa.net/documents/HiPS/`

Multi-Order Coverage map



Credits: `http://ivoa.net/documents/MOC/`

Introduction

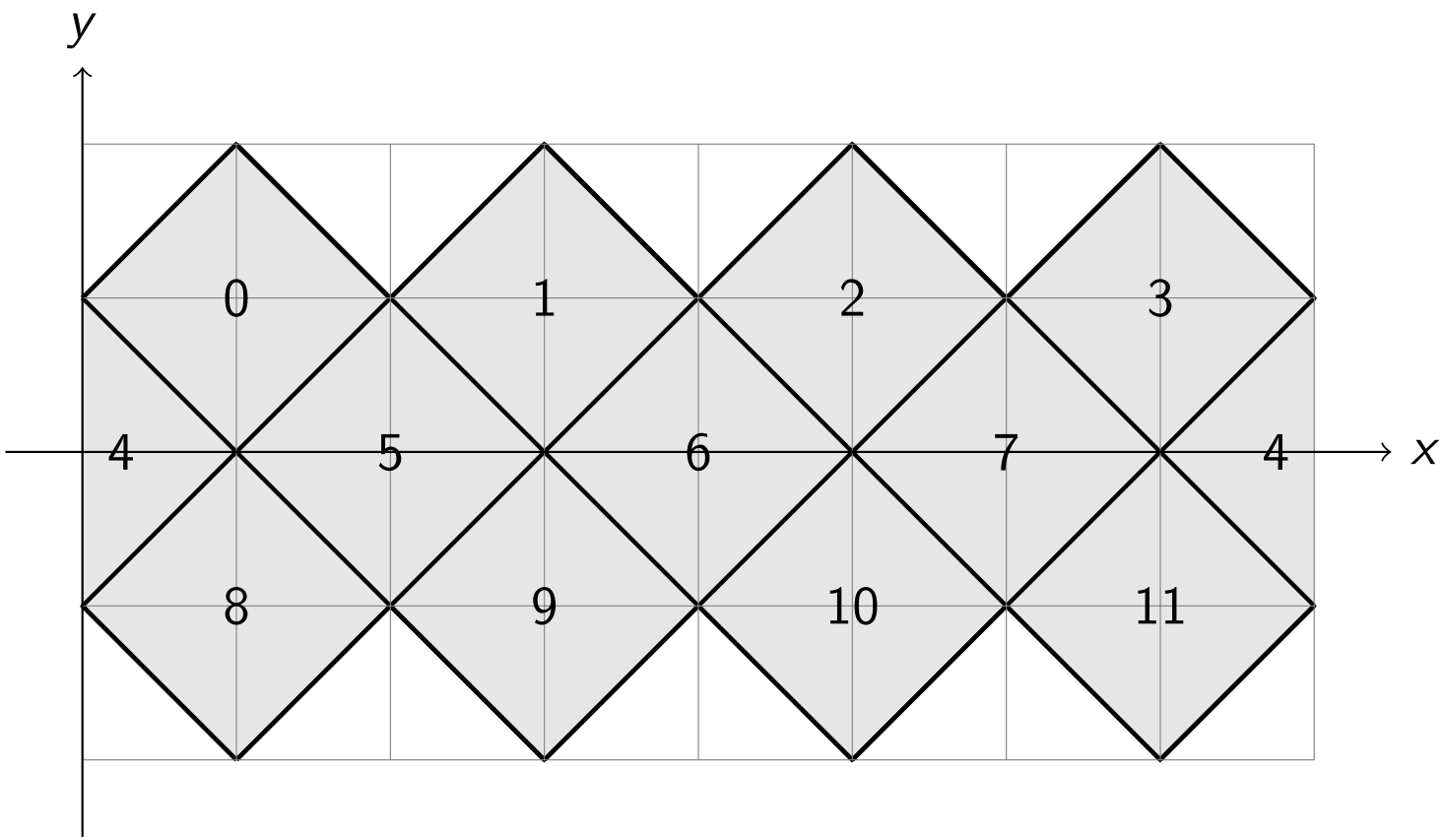
- Many libraries into the wild
  - "Official" libs (M. Reinecke for Java and C++), GPL licence
  - C/python lib in astropy-healpix (D. Lang/ T. Robitaille), BSD licence
  - Core functions in C (M. Reinecke), BSD licence
  - TypeScript (K. Michitaro), BSD licence
  - Julia (M. Tomasi), MIT licence
  - ...
- HEALPix widely used at CDS
  - at the heart of Aladin and AladinLite
  - used to index data in VizieR, SIMBAD, the cross-match service
- **Motivations** to develop a CDS library
  - Develop an internal expertise
  - Easier evolution for CDS's needs
  - Homogeneity: a same origin for various targets
    - ⋆ Java, Javascript, Python, PostgreSQL, ...
  - Control the licence ("official" librairies are in GPL)

Table of content

HEALPix: is a projection

HEALPix made for Cosmology (Górski et al. 2015).
It is first of all an:

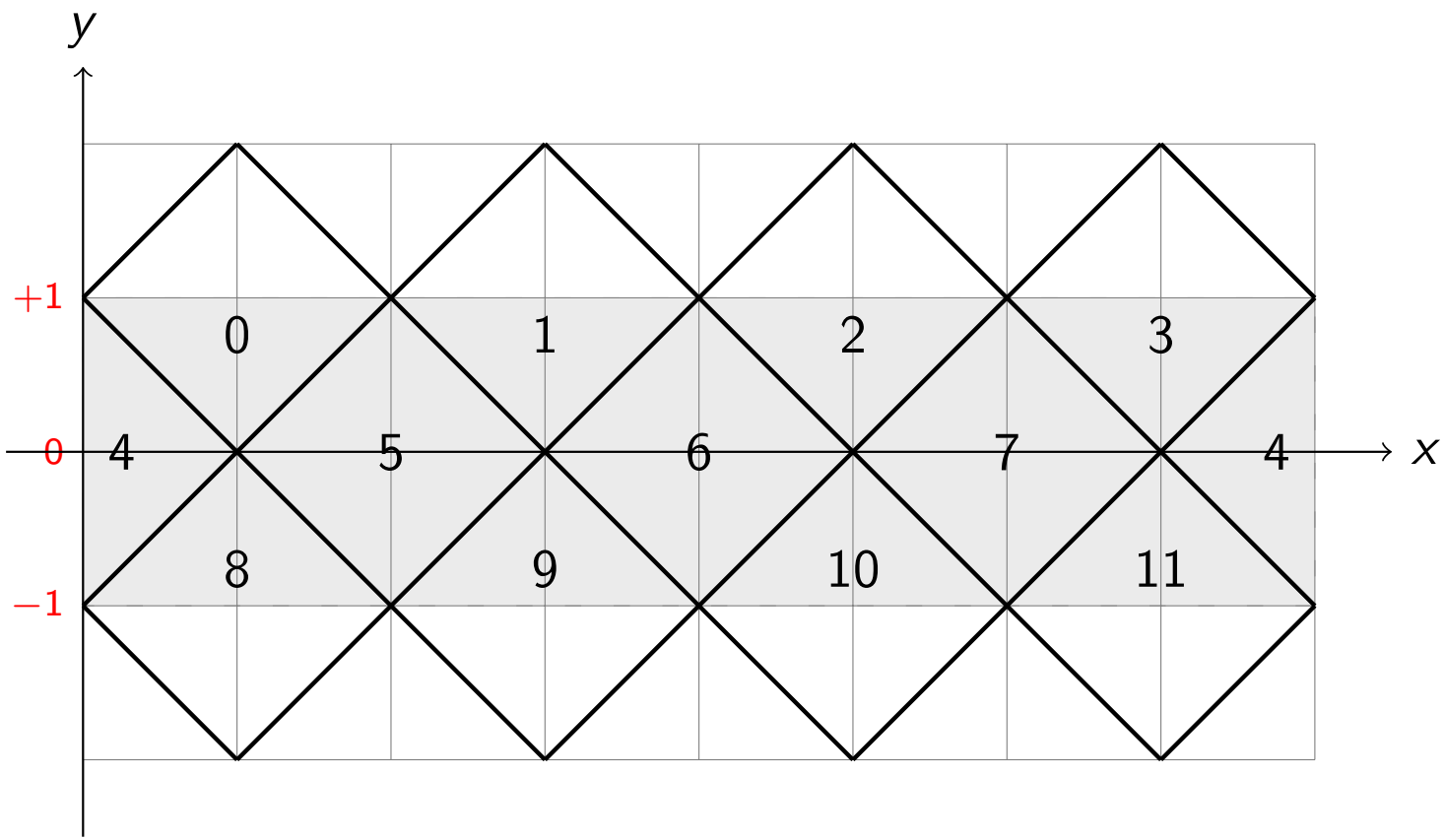
- iso-latitude;
- equal-area projection (i.e. $\mathrm{d}x\mathrm{d}y \propto \cos\delta \mathrm{d}\alpha\mathrm{d}\delta$);
- made of two other equal-area projections (Calabretta et al. 2007).



Equatorial region ($|\sin\delta| \leq 2/3$)

- Cylindrical Equal Area projection (CEA):

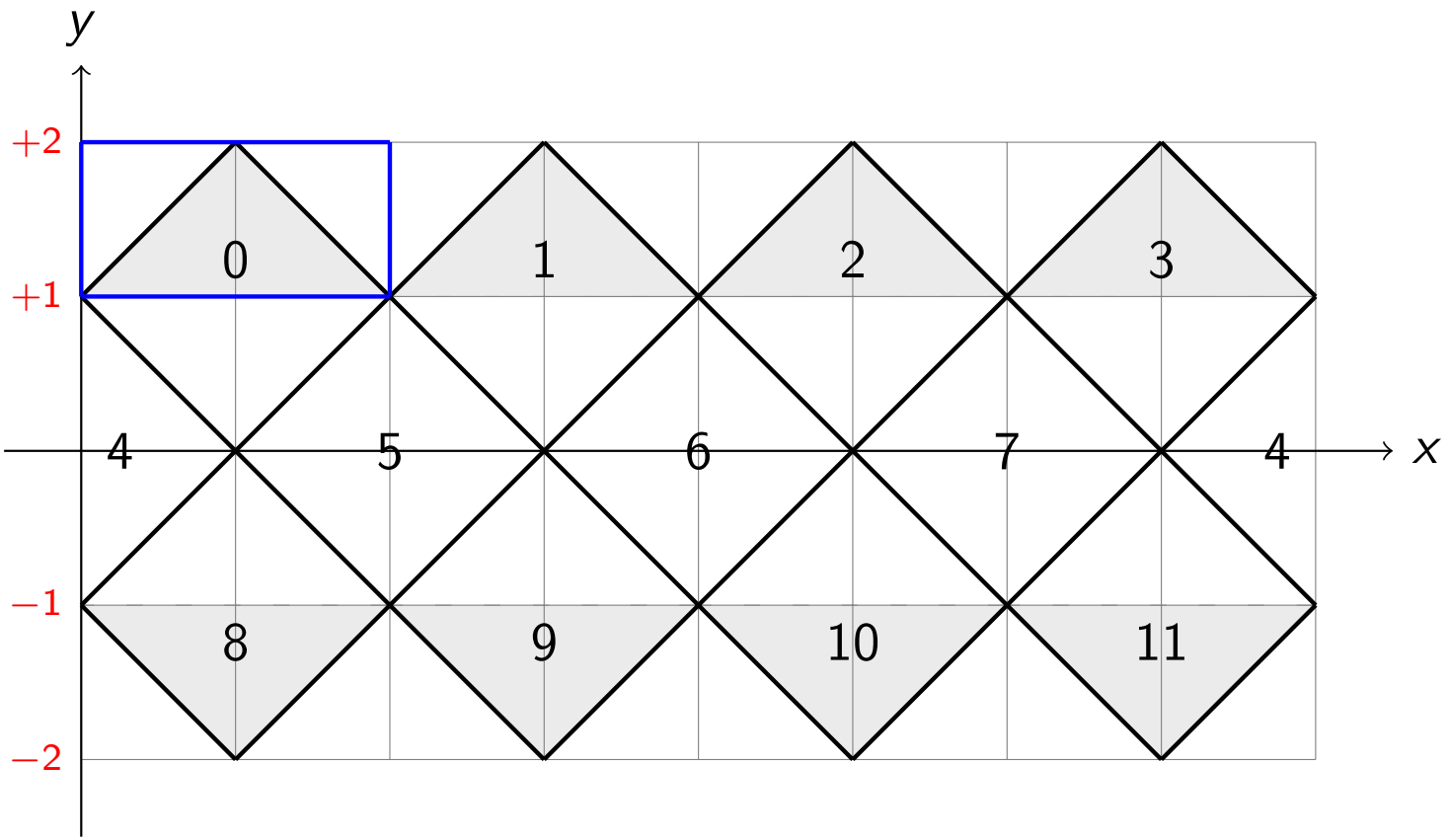
$$\left\{\begin{array}{|rcl|}\hline x & = & \frac{4}{\pi}\alpha \\ y & = & \frac{3}{2}\sin(\delta) \\ \hline\end{array}\right. \Rightarrow \left\{\begin{array}{lcl} \alpha \in [0, 2\pi] & \rightsquigarrow & x \in [0, 8] \\ \sin\delta \in [-\frac{2}{3}, \frac{2}{3}] & \rightsquigarrow & y \in [-1, 1] \end{array}\right.$$



Polar caps ($|\sin\delta| > 2/3$)

- Collignon projection for $\alpha \in [0, \pi/2]$ and $\sin\delta > 3/2$:

$$\left\{ \begin{array}{rcl} t & = & \sqrt{3(1-\sin\delta)} \\ x & = & (\alpha\frac{4}{\pi} - 1)t + 1 \\ y & = & 2 - t \end{array} \right. \Rightarrow \left\{ \begin{array}{l} \alpha \in [0, \frac{\pi}{2}] \\ \sin\delta \in]\frac{2}{3}, 1] \end{array} \right. \rightsquigarrow \begin{array}{l} t \in [0, 1[\\ x \in]0, 2[\\ y \in]1, 2] \end{array}$$



Polar caps ($|\sin\delta| > 2/3$)

Problem

- Numerical issue with $t = \sqrt{3(1-\sin\delta)}$ near the poles
  - $\arcsin(0.9999999999999999) \approx 89.99999919$ deg
  - $\frac{\pi}{2} - \arcsin(0.9999999999999999) \approx 2.917$ mas

Our solution

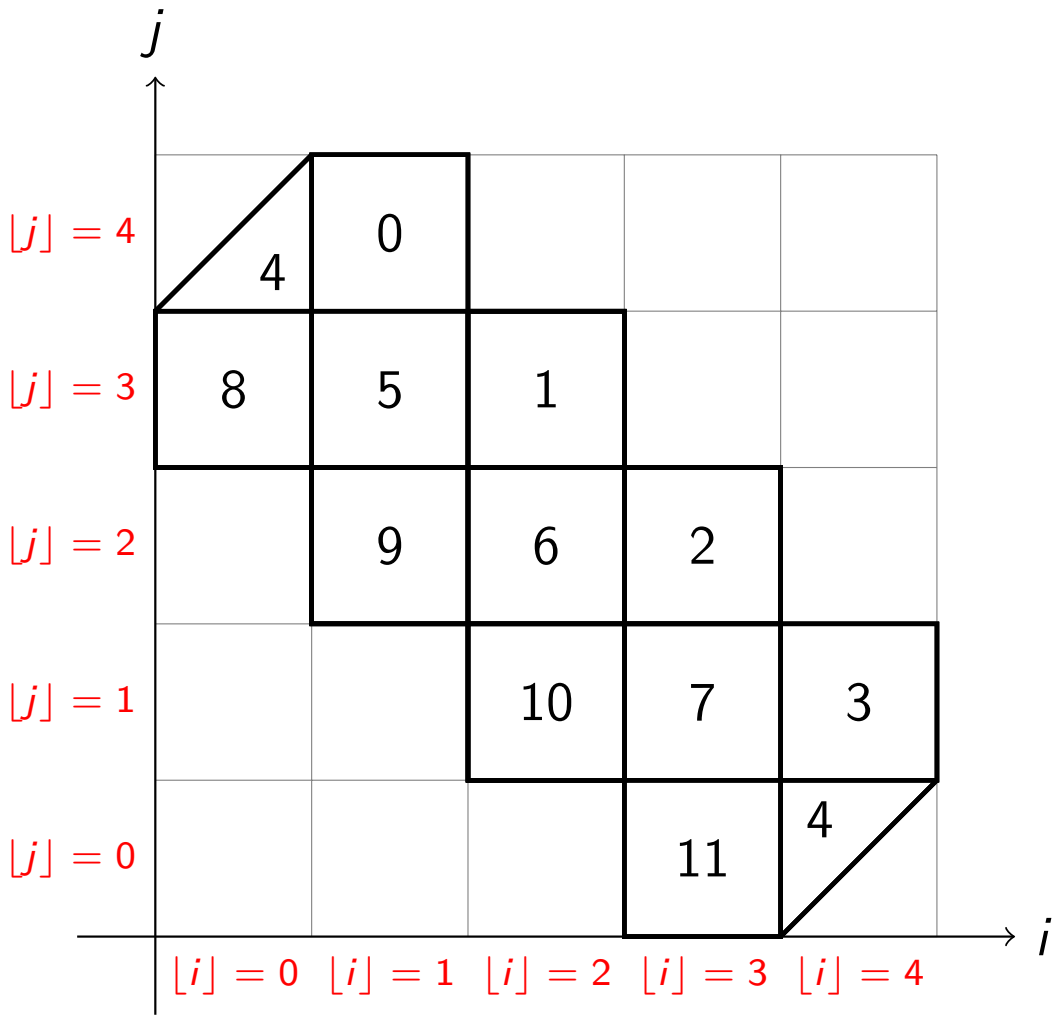
- Use instead $\boxed{t = \sqrt{6}\cos(\frac{\delta}{2} + \frac{\pi}{4})}$

Demonstration

$$1 - \sin\delta = 1 + \cos(\delta + \frac{\pi}{2}) = 2\frac{1 + \cos\left(2(\frac{\delta}{2} + \frac{\pi}{4})\right)}{2} = 2\cos^2(\frac{\delta}{2} + \frac{\pi}{4})$$

Base cells numbering

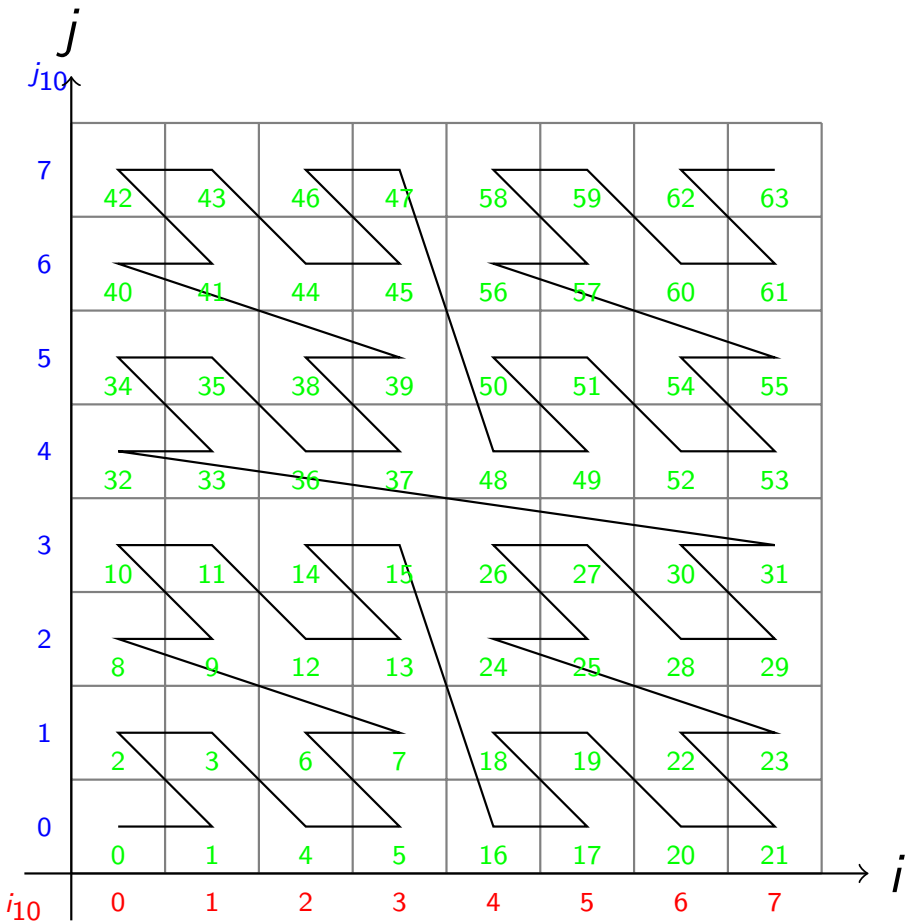
- Base cell $\in [0, 12[\Rightarrow$ coded on 4 bits $(2^3 + 2^2 + 2^1 + 2^0 = 15)$
- Requires a function $f : (\lfloor i \rfloor, \lfloor j \rfloor) \rightsquigarrow b_4 b_3 b_2 b_1$

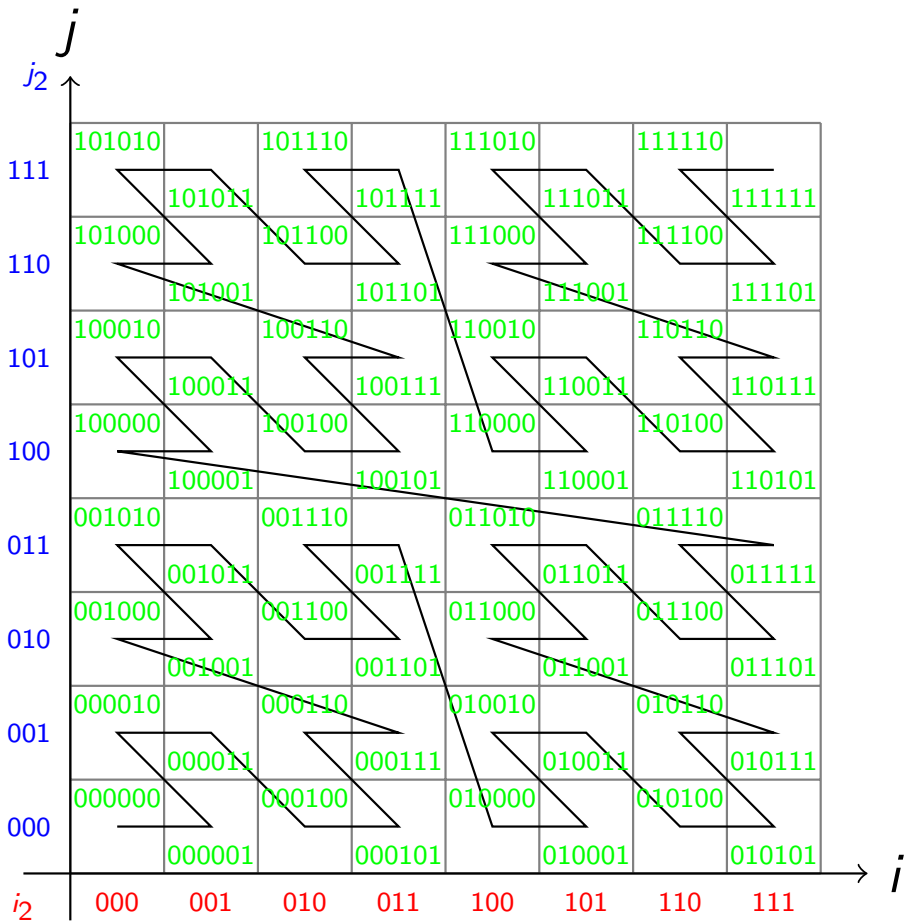


Z-order curve ($nside = 8$)

$$\begin{cases} i & \leftarrow & \lfloor (i \times nside) \mod nside \rfloor \\ j & \leftarrow & \lfloor (j \times nside) \mod nside \rfloor \end{cases} \Rightarrow (i,j) \in [0, nside-1]^2$$

Bit interleaving: $i = i_1 i_2 i_3 i_4$, $j = j_1 j_2 j_3 j_4 \rightsquigarrow z = j_1 i_1 j_2 i_2 j_3 i_3 j_4 i_4$ (Morton code)

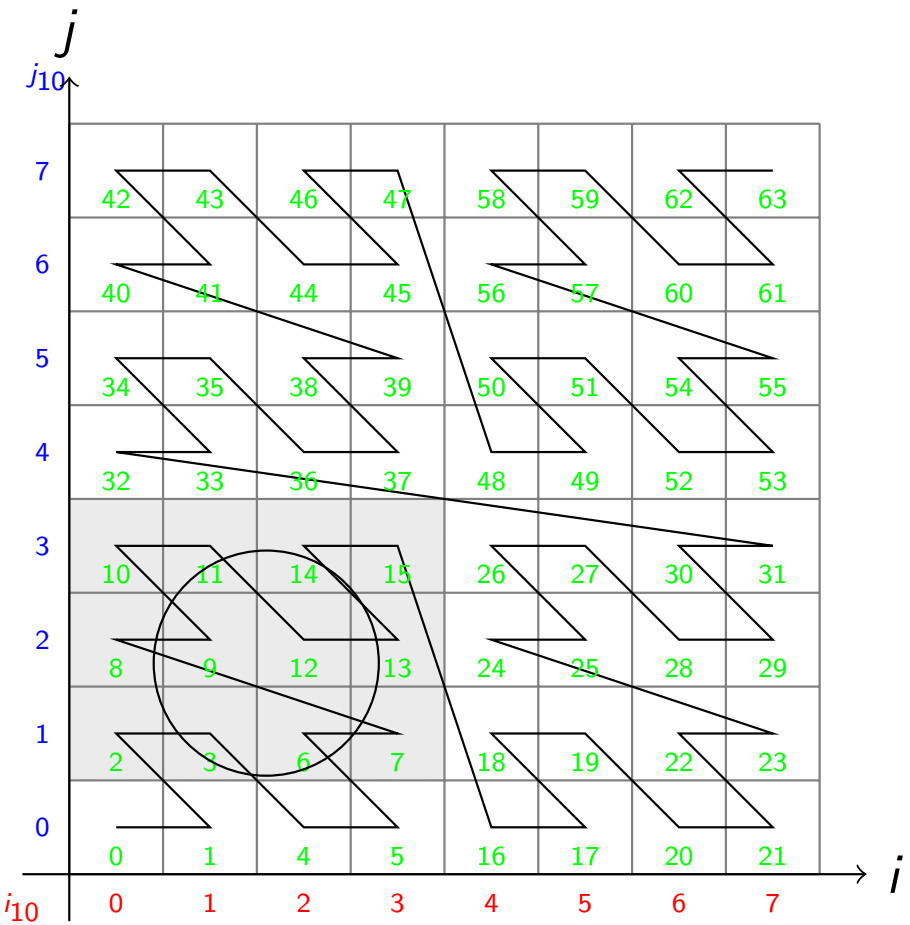


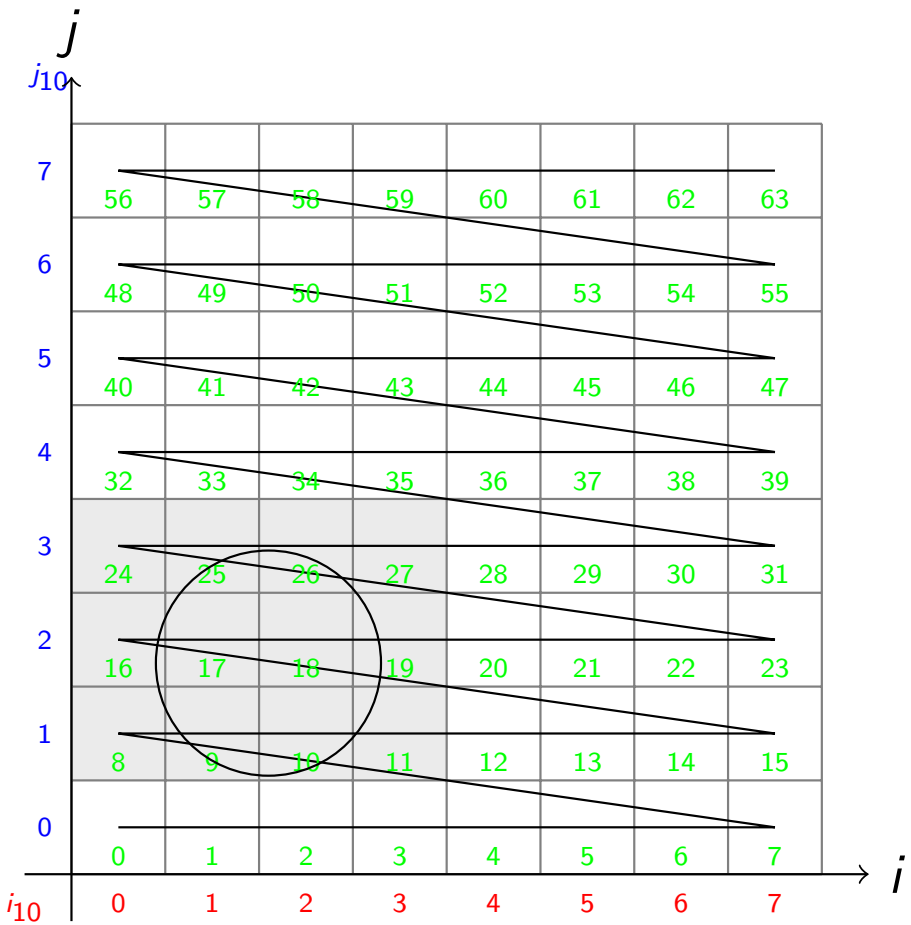


Z-order curve

Advantage over row by row scheme: less seeks, smaller seeks (in mean)

- Better performances when retrieving data from HDD (Peano-Hilbert even better)





Bit interleaving

Several methods possible for 32-bit integers

- `https://graphics.stanford.edu/~seander/bithacks.html#InterleaveTableObvious`
  - Naive for loop
  - Table lookup (the fastest)
  - 64-bit multiply (no tested yet)
- "Hacker's Delight" by Henry S. Warren, Jr. §Shuffling Bits:

```
abcd efgh ijkl mnop ABCD EFGH IJKL MNOP
abcd efgh ABCD EFGH ijkl mnop IJKL MNOP
abcd ABCD efgh EFGH ijkl IJKL mnop MNOP
abAB cdCD efEF ghGH ijIJ klKL mnMN opOP
aAbB cCdD eEfF gGhH iIjJ kKlL mMnN oOpP
```

```
x = (x & 0x0000FF00) <<  8 | (x >> 8) & 0x0000FF00 | x & 0xFF0000FF;
x = (x & 0x00F000F0) <<  4 | (x >> 4) & 0x00F000F0 | x & 0xF00FF00F;
x = (x & 0x0C0C0C0C) <<  2 | (x >> 2) & 0x0C0C0C0C | x & 0xC3C3C3C3;
x = (x & 0x22222222) <<  1 | (x >> 1) & 0x22222222 | x & 0x99999999;
```

Bit interleaving

CDS adaptation to account for 64-bit integers

```
h = (0x00000000FFFF0000L & h) <<  16
  | (0x0000FFFF00000000L & h) >> 16
  | (0xFFFF00000000FFFFL & h);
h = (0x0000FF000000FF00L & h) <<  8
  | (0x00FF000000FF0000L & h) >> 8
  | (0xFF0000FFFF0000FFL & h);
h = (0x00F000F000F000F0L & h) <<  4
  | (0x0F000F000F000F00L & h) >> 4
  | (0xF00FF00FF00FF00FL & h);
h = (0x0C0C0C0C0C0C0C0CL & h) <<  2
  | (0x3030303030303030L & h) >> 2
  | (0xC3C3C3C3C3C3C3C3L & h);
h = (0x2222222222222222L & h) <<  1
  | (0x4444444444444444L & h) >> 1
  | (0x9999999999999999L & h);
```

Bit interleaving

- "Hacker's Delight" by Henry S. Warren, Jr. §Shuffling Bits, XOR based, adapted to account for 64-bit integers

```
t = (h ^ (h >> 16)) & 0x00000000FFFF0000L; h = h ^ t ^ (t << 16);
t = (h ^ (h >>  8)) & 0x0000FF000000FF00L; h = h ^ t ^ (t <<  8);
t = (h ^ (h >>  4)) & 0x00F000F000F000F0L; h = h ^ t ^ (t <<  4);
t = (h ^ (h >>  2)) & 0x0C0C0C0C0C0C0C0CL; h = h ^ t ^ (t <<  2);
t = (h ^ (h >>  1)) & 0x2222222222222222L; h = h ^ t ^ (t <<  1);
```

- BMI2 instructions (not available in Java)

```
t = pdep_u64(i, 0x5555555555555555) | pdep_u64(j, 0xAAAAAAAAAAAAAAAA)
```

- Best performances (in my tests): BMI2.0 (if available), then Lookup table.
  - but lookup tables use more cache...

HEALPix cell number

- Project (α, δ) into the Euclidean plane
- Shift, rotate and scale so that each base cell as a size *nside* $\times$ *nside*
- Compute the base cell number
- Compute the Morton number from the coordinates inside the base cell
- Final 64-bit cell number

$$\begin{aligned} ipix &= \ldots 00 b_3 b_2 b_1 b_0 j_1 i_1 j_2 i_2 j_3 i_3 j_4 i_4 \ldots \\ order + ipix &= \ldots 0 s b_3 b_2 b_1 b_0 j_1 i_1 j_2 i_2 j_3 i_3 j_4 i_4 \ldots \end{aligned}$$

  - base cell: 4 bits
  - 2 bits by level
  - $\Rightarrow$ 29 levels ($4 + 29 \times 2 = 62$ bits)
  - 63 bits with the sentinel bit coding the order
- Remark: $(ipix_2 = \ldots 0 b_3 b_2 b_1 b_0 j_1 i_1 j_2 i_2) >> 2 = (ipix_1 = \ldots 0 b_3 b_2 b_1 b_0 j_1 i_1)$

Table of content

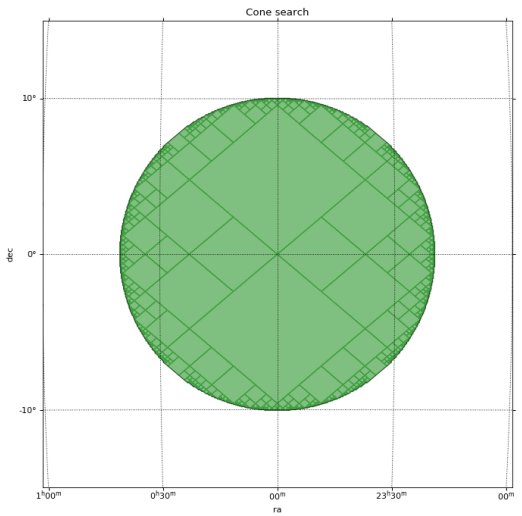
CDS libraries

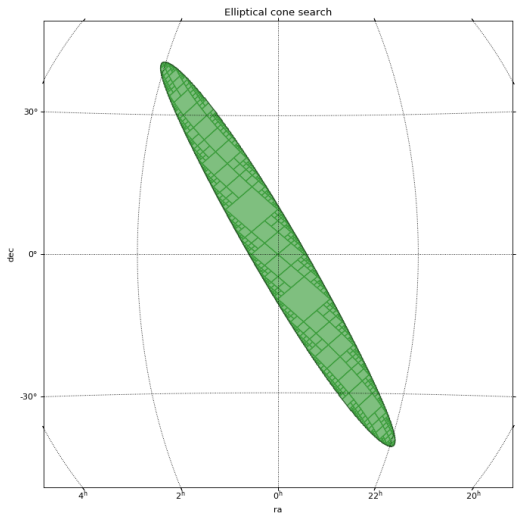
- Two libraries developped at CDS
  - Java: `https://github.com/cds-astro/cds-healpix-java`
    - ★ 3-clause BSD licence
    - ★ Clean code + code to be cleaned + exploratory code
    - ★ Now replaces the "official" lib in Aladin
  - Rust: `https://github.com/cds-astro/cds-healpix-rust/`
    - ★ Apache or MIT licence (at your convenience)
    - ★ Clean code only
    - ★ See M. Baumann's presentation on Rust!
- The Rust lib is available
  - in WebAssembly for Web developpment
    - ★ Integration in Aladin Lite currently being tested
  - in Python (Rust binaries are like C ones)
    - ★ See M. Baumann's presentation on the Python wrapper!

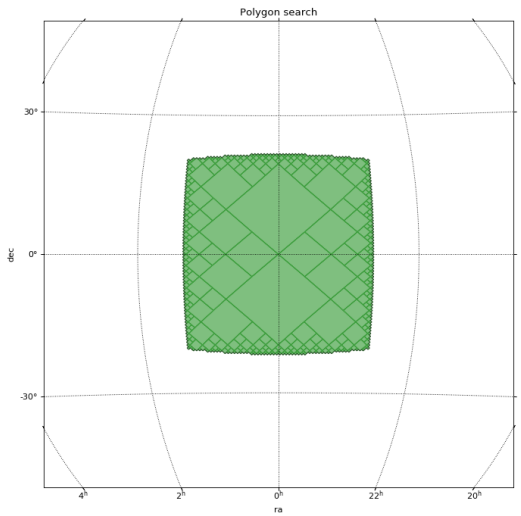
```
pip install cdshealpix
```

Functionalities

- projection, deprojection: $(\alpha, \delta) \leftrightarrow (x, y)$
- *ipix* from (α, δ)
- cell center, vertices, neighbours, path along cell edge, ...
- cone/ellipse/polygone coverage (BMOC)
- logical operations (NOT/AND/OR/XOR) on BMOC
- ...



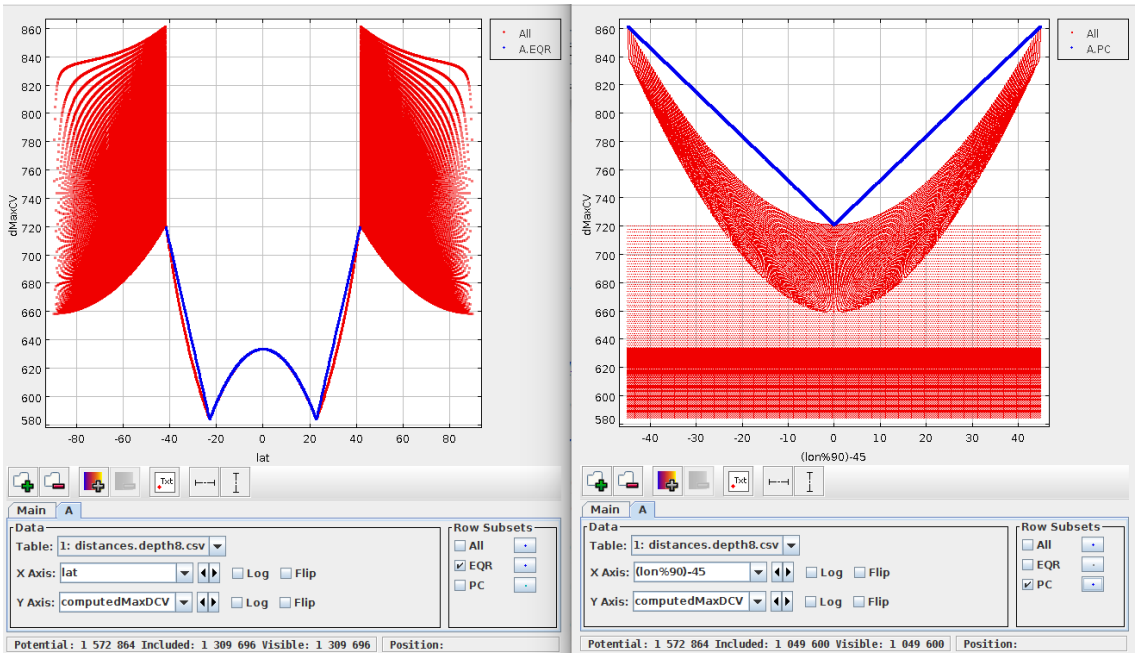




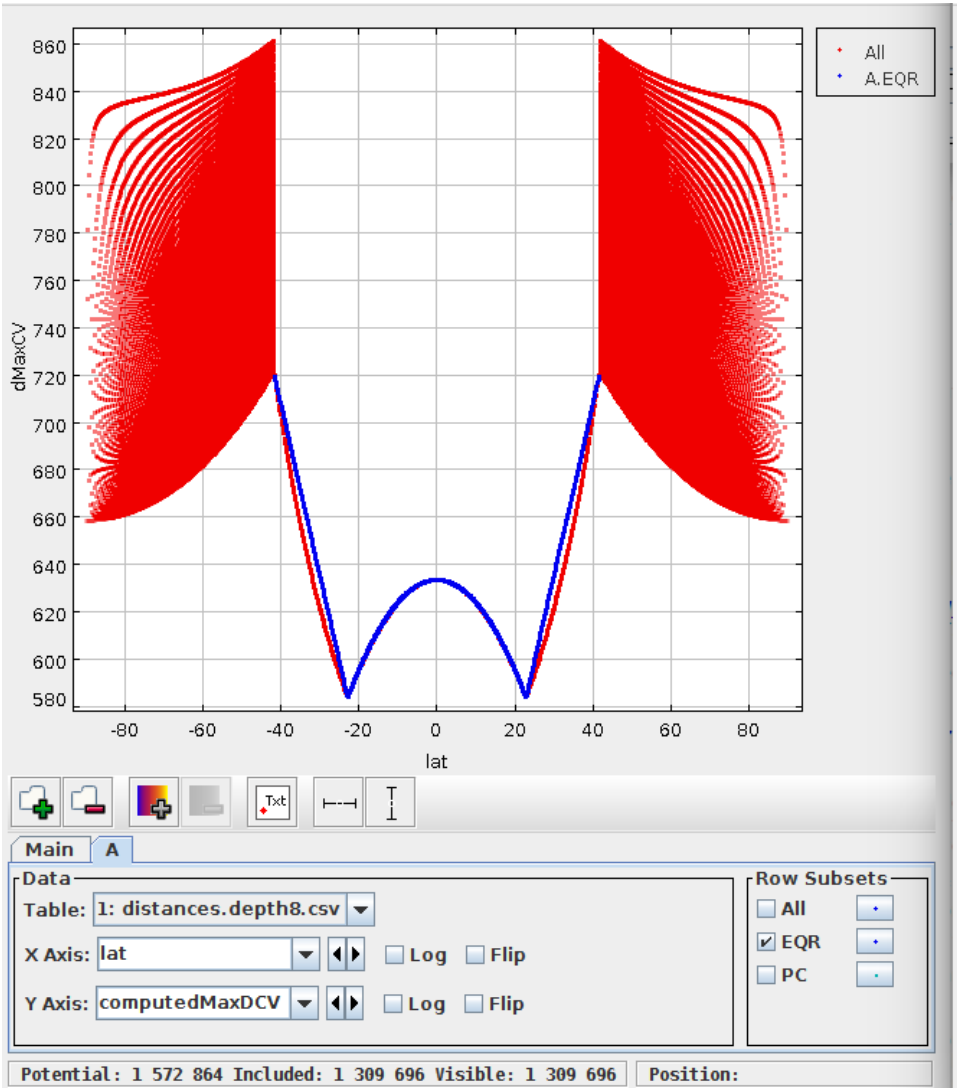
Credit: cdshealpix python documentation

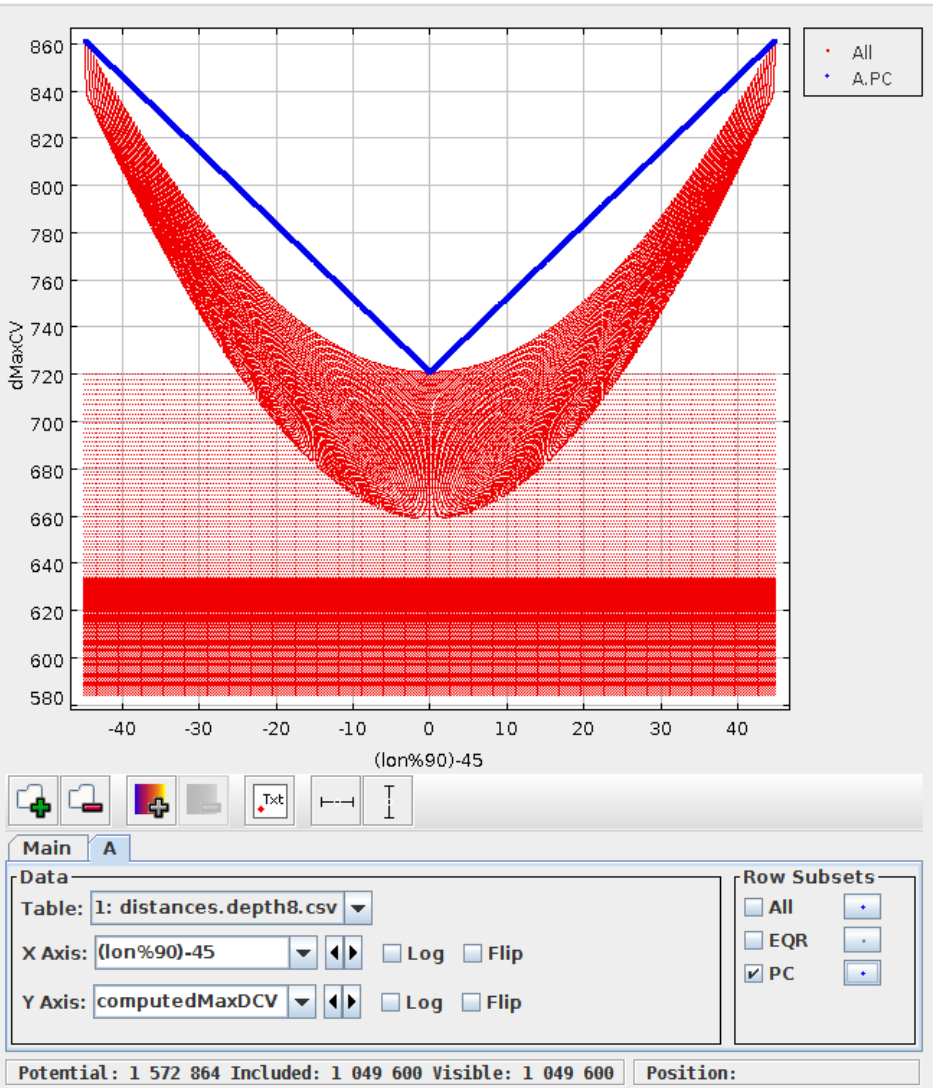
Characteristic quantities

- Smallest possible distance between two opposite borders of a cell
  - Order at which a cone intersect at most a pixel and its 8 neighbours
  - Used to the 9 starting cells in cone/ellipse/polygon queries
- Largest cell center-vertex distance as a function of (α, δ)
  - Approx cone/ellipse searches
  - Fast cell rejection test when performing a cone search



Center-vertex distance





BMOC

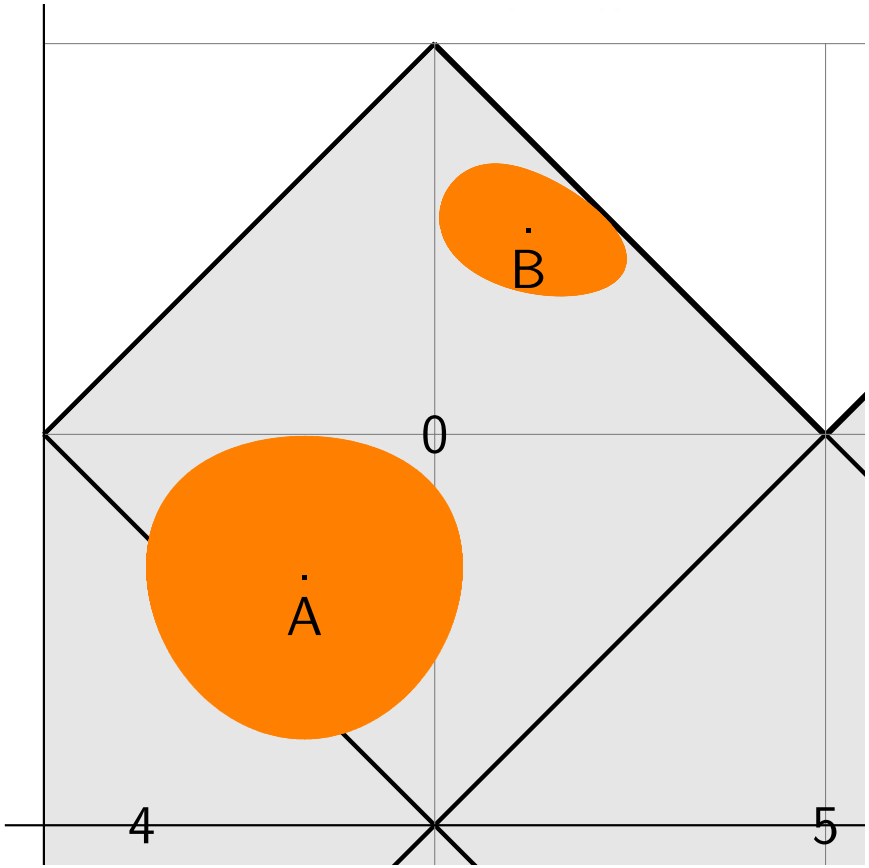
- Query cone/ellipse/polygone returns a BMOC
  - BMOC = MOC contaning for each cell a binary flag

    false the cell area is not fully covered by the cone
    true the cell area is fully covered by the cone
  - specific internal bit scheme: $...0 b_3 b_2 b_1 b_0 j_1 i_1 j_2 i_2 s 0...0 f$
    - ⋆ $\Rightarrow$ natural ordering follow Z-order curve ordering

    s sentinel bit coding the order
    f flag bit
- Advantages e.g. retrieving sources in a catalogue file
  - Retrieve sequentially (if data ordered by HEALPix index)
  - No need for the expansive distance test for sources in the cells fully covered by the cone
- Logical operations NOT/AND/OR/XOR implemented

HEALPix Query cone

- Exact list of cells intersecting a cone
  - 4 vertices in the cone
    $\Rightarrow$ cone contains the cell
  - 1-3 vertices in the cone
    $\Rightarrow$ cone overlaps the cell
  - cell contains a special point
    $\Rightarrow$ cone overlaps the cell
    - ★ 4 special points: slope of tangent line (in the Cartesian projection) $= \pm 1$
  - case of large cells: check if center of the cone is in the cell



Cone A ($\alpha = 30, \delta = 25, \theta = 16.5°$)
Cone B ($\alpha = 67.5, \delta = 67.5, \theta = 8°$)

HEALPix Query cone

Expression of a cone (from Haversine):

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

Then equation of tangent lines:

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

In the equatorial zone

$$\begin{aligned} z &= \sin\delta \\ X &= 4/\pi\alpha \\ Y &= 3/2z \end{aligned}$$

Thus

$$\begin{aligned} \frac{\mathrm{d}\Delta X}{\mathrm{d}\Delta\alpha} &= 4/\pi \\ \frac{\mathrm{d}\delta}{\mathrm{d}z} &= \frac{1}{\sqrt{1-z^2}} = \frac{1}{\cos\delta} \\ \frac{\mathrm{d}z}{\mathrm{d}Y} &= 2/3 \end{aligned}$$

and, finally

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

HEALPix Query cone

Expression of a cone (from Haversine):

$$\alpha(\delta) = \alpha_0 \pm 2 \arcsin\left(\sqrt{\frac{\sin^2\frac{\theta}{2} - \sin^2\frac{\delta-\delta_0}{2}}{\cos\delta_0 \cos\delta}}\right)$$

In the polar caps

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

with

$$\begin{aligned} z &= \sin\delta \\ t &= \sqrt{3(1-z)} = \sqrt{6}\cos(\frac{\delta}{2} + \frac{\pi}{4}) \\ X &= (\frac{4}{\pi}\alpha - 1)t + 1 \\ Y &= 2 - t \end{aligned}$$

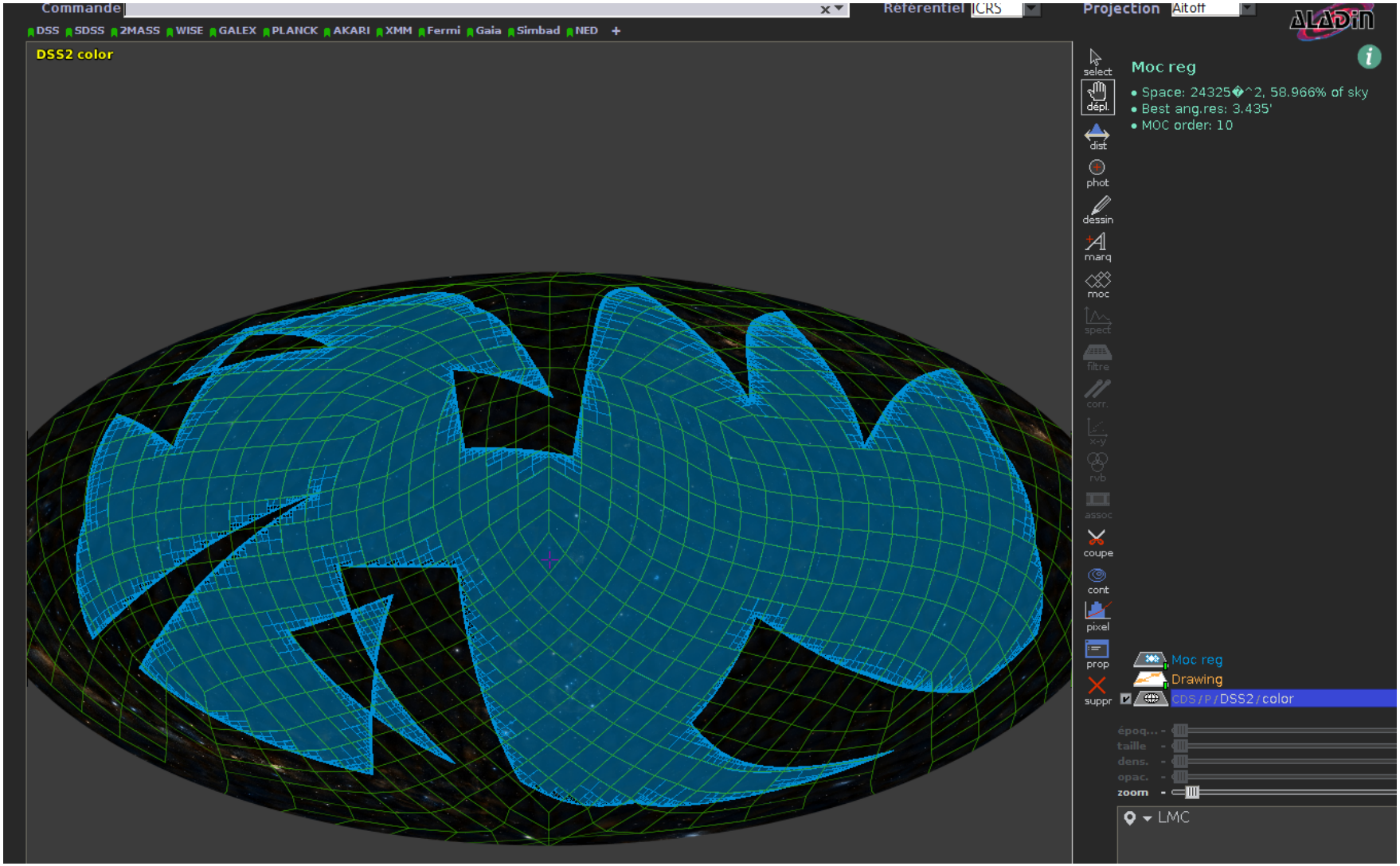
thus

$$\begin{aligned} \frac{\mathrm{d}\delta}{\mathrm{d}z} &= \frac{1}{\cos\delta} \\ \frac{\mathrm{d}z}{\mathrm{d}Y} &= \frac{2}{3}t \end{aligned}$$

and

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

HEALPix Query polygon

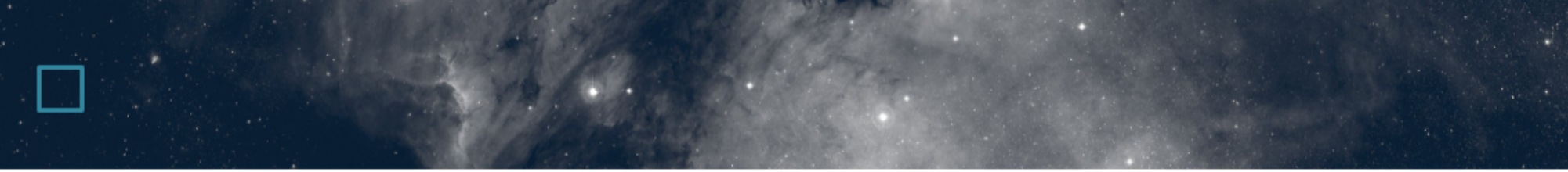


Exact solution looking for a special point for each polygone edge:
great circle = cone with $R = \pi/2$

Current status

- Clean code (Rust / Python), open source, documented, with regression tests
  - e.g. 73 tests (41 tests + 32 doctests) and 6 benchmarks in Rust
- Young library which will continue being improved and enriched
- In production in Aladin (Java); Used internally (Rust); Aladin Lite (WASM)?

Feature \Language	Java	Rust	Python
Cell number from coo	yes	yes	yes
center, vertices	yes	yes	yes
path along celle edge	yes	could	could
neihbour, neighbours	yes	yes	yes
ordered deeper neighbours	yes	could	could
ring2nested, nested2ring	yes	could	could
Approx cone coverage	yes	yes	yes
Exact cone coverage	yes		
Approx ellipse coverage	almost	yes	yes
Exact polygone coverage	yes	yes	yes
BMOC logical operations	almost	yes	almost



To be continued...