Astronomy & Astrophysics manuscript no. paper
May 22, 2023

A GPU-accelerated viewer for HEALPix maps

https://www.sfu.ca/physics/cosmology/healpix/

Andrei V. Frolov[1,⋆]

Department of Physics, Simon Fraser University, 8888 University Drive, Burnaby BC V5A 1S6, Canada

Received 19 May 2023; accepted ???

ABSTRACT

Context. HEALPix by Górski et al. (2005) is de-facto standard for Cosmic Microwave Background (CMB) data storage and analysis, and is widely used in current and upcoming CMB experiments. Almost all the datasets in Legacy Archive for Microwave Background Data Analysis (LAMBDA) use HEALPix as a format of choice. Visualizing the data plays important role in research, and several toolsets were developed to do that with HEALPix maps, most notably original Fortran facilities and Python integration with `healpy`.
Aims. With the current state of GPU performance, it is now possible to visualize extremely large maps in real time on a laptop or a tablet. HEALPix Viewer described here is developed for macOS, and takes full advantage of GPU acceleration to handle extremely large datasets in real time. It compiles natively on Intel and Arm64 architectures, and uses Metal framework for high-performance GPU computations. The aim of this project is to reduce the effort required for interactive data exploration, as well as time overhead for producing publication-quality maps. Drag and drop integration with Keynote and Powerpoint makes creating presentations easy.
Methods. The main codebase is written in Swift, a modern and efficient compiled language, with high-performance computing parts delegated entirely to GPU, and a few inserts in C interfacing to `cfitsio` library for I/O. Graphical user interface is written in SwiftUI, a new declarative UI framework based on Swift. Most common spherical projections and colormaps are supported out of the box, and the available source code makes it easy to customize the application and to add new features if desired.
Results. On a M1 Max laptop, an `nside` = 8192 map is processed in real time, with geometry effects being rendered at 60fps in full resolution with no appreciable load to the machine. Main user-facing delays are limited to CPU-bound `cfitsio` load times, as well as sorting needed to construct Cumulative Distribution Function (CDF) estimators for statistical analysis (hidden in background queue).

Key words. Cosmology: cosmic background radiation – Methods: data analysis – Techniques: image processing

1. Introduction

Since its original discovery Dicke et al. (1965), cosmic microwave background (CMB) has been a veritable gold mine of information on cosmology in general and early universe physics in particular. After COBE measured CMB spectrum to be almost perfectly that of black body radiation Mather et al. (1990) and detected CMB temperature anisotropy for the first time Smoot et al. (1992), implications for cosmology were quickly explored, e.g. Efstathiou et al. (1992). Polarization of CMB was detected 10 years later Kovac et al. (2002), and the field was maturing quickly with a number of ground, balloon-borne, and satellite experiments, as well as work by theorists, to become truly a source for precision cosmology. Two of the satellite missions, NASA's WMAP Bennett et al. (2013) and ESA's Planck Planck Collaboration et al. (2020a) were particularly impactful. Current, future, and proposed CMB experiments will carry the torch, such as Simons Obervatory Ade et al. (2019), LiteBIRD Hazumi et al. (2012); Matsumura et al. (2014), CMB Stage 4 Abazajian et al. (2019), and PICO Hanany et al. (2019). Cosmic microwave background observations were used not only to measure cosmological parameters Hinshaw et al. (2013); Planck Collaboration et al. (2020b) and explore constraints on primordial fluctuations Planck Collaboration et al. (2020e,d,c), but they also provide information on astrophysical foregrounds and are becoming important for galactic science, e.g. Hensley et al. (2022).

De-facto standard for CMB data storage and analysis is HEALPix (Hierarchical Equal Area isoLatitude Pixelation) Gorski et al. (1999); Górski et al. (2005); Górski & Hivon (2011), used by both WMAP and Planck Hivon et al. (2015), as well as by most of the data published on LAMBDA (Legacy Archive for Microwave Background Data Analysis) Addison et al. (2019). Original HEALPix library and facilities were written in Fortran (with C bindings available), but it since has been ported or interfaced to a number of languages including R Fryer et al. (2019), Julia Tomasi & Li (2021), and most notably Python with `healpy` package Zonca et al. (2020).

Scientific data visualization plays an important role in research, and two of the most widely used tools for visualizing HEALPix maps are original `map2gif` facility of HEALPix, and `healpy` package. A number of interactive 3D visualization tools were developed, either written in Java Joliet et al. (2008); Fernique et al. (2010), cross-platform frameworks as LAMBDA's SkyViewer (uses Qt) and Univiewer Mingaliev & Stolyarov (2010) (uses wxWidgets), or targeting specific operating systems such as CMBview for macOS Portsmouth (2011). Some of these projects appear to be orphaned, and GPU technology has come a long way since then, both in hardware and software available. OpenCL superseded OpenGL for massively parallel heterogeneous computing, and in turn was followed by more optimized but vendor-specific GPU libraries such as Apple's Metal or AMD's Vulkan. It is now possible to process extremely large datasets on commodity hardware, and even on portable devices.

⋆ e-mail: `frolov@sfu.ca`

This paper describes HEALPix Viewer, a macOS application I wrote to visualize HEALPix data, which takes advantage of the latest technology (SwiftUI and Metal libraries) and provides a number of improvements both in convenience and performance over existing solutions. It implements a strict superset of `map2gif` features (including original color palettes and projections), is fully GPU accelerated, and is geared towards interactive data analysis and exploration. While perhaps not as flexible as what could be achieved with `healpy` given enough effort, it is much faster and more convenient. HEALPix Viewer is capable of handling extremely large maps (such as those that will be produced by Simons Observatory) without any shortcuts or approximations, and produces high-quality output for publication or direct inclusion into presentations via drag and drop. Both source code and binaries are available for developers and researchers.

The paper is organized as follows: Section 2 briefly describes HEALPix data format and pixelization of a sphere, Section 3 explains overall data processing pipeline structure, Section 4 focuses on data analysis tools provided, while Section 5 briefly goes over the user interface. Performance and limitations are discussed in Section 6, and conclusions are presented in Section 7. Appendices go over spherical projections implemented (A), Euler angles and generators of rotation (B), dynamics of a sphere used for animation of viewpoint transitions (C), and implementation of inverse error function used for map normalization (D).

2. HEALPix file format

HEALPix data is incapsulated in a FITS (Flexible Image Transport System) file format, with specific metadata tags identifying the file as HEALPix data. FITS format is widely used in astronomy, and has many available I/O libraries targeting different languages or programming environments. As in original HEALPix code, we chose `cfitsio` library, developed and maintained by NASA's HEASARC. It is efficient and highly portable, and compiles out of the box on Intel and Arm64 architectures, which covers all of the Apple hardware (including iPhone). Accessing C functions from Swift is easy once bridging header is generated (which can be done automatically).

HEALPix metadata parser was written from scratch, using Swift facilities not available in Fortran (such as complex enumeration types and dictionaries), which simplified execution logic to a large extent. Complete specification of HEALPix format is available here, but we had to loosen the strict requirements on some tags to read the data as produced and published by major experiments. Absolutely necessary are `PIXTYPE = "HEALPIX"` tag identifying file as containing HEALPix data, and `NSIDE` and `ORDERING` tags specifying map resolution and ordering scheme (can be either `RING` or `NESTED`) of the map. `POLAR` tag indicates polarization data is available (as Stokes Q and U parameters), and has two sign conventions supported (`IAU` and `COSMO`). Map data itself is stored in a binary table, chunked and potentially having different data formats for different columns. The data reading code is ported from original Fortran implementation in HEALPix, using `cfitsio` routines called directly from Swift.

HEALPix pixelization of a sphere provides equal area and iso-latitude pixels, facilitating efficient spherical transforms using fast Fourier transform for ϕ = const rings, while using Legendre transform in θ direction Górski et al. (2005). `RING` storage ordering enumerates pixels along iso-latitude rings and is best suited for spherical transform computation, while `NESTED` ordering enumerates pixels in a hierarchical fashion for 12 faces (using interleaved x and y bit pattern, also known as Morton ordering in image processing literature) and is best suited for localized memory access. To keep things simple, HEALPix Viewer converts `RING` data into `NESTED` ordering on input, and all further processing is done on `NESTED` maps. To illustrate the order in which `NESTED` pixels are enumerated, Figure 1 shows pixel index colored with Planck palette (deep blue to dark broun) in Mollweide projection, as seen from inside the sphere.

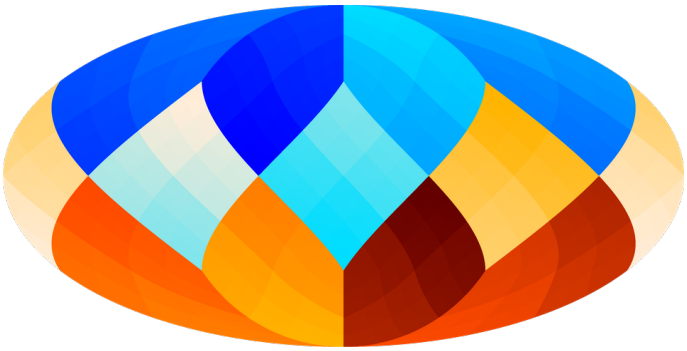

Fig. 1. HEALPix pixelization of a sphere colored according to `NESTED` pixel order (in Mollweide projection). 12 faces are clearly distinguishable, and within each, pixels are indexed in interleaved bit pattern (also known as Morton ordering).

3. Processing pipeline

Primary design goal of HEALPix Viewer is fast and easy data visualization, so implementation decisions were made in favour of performance versus ultimate accuracy. Processing pipeline is 32-bit floats (64-bit float maps are down-converted at load time), and hardware acceleration with graphical processing unit (GPU) is used as much as possible. Basic 1-point statistic tools are provided for convenience (PDF, CDF, various moments of the distribution), but neither multi-point statistics nor spherical transforms are implemented (the latter due to the fact that there is no GPU-accelerated spherical transform libraries available so far).

Overall structure of the data flow and user inputs is illustrated in Figure 2. HEALPix data gets loaded from disk into CPU memory, down-converted to 32-bit floats and rearranged into `NESTED` ordering if necessary, and then loaded to GPU buffer (on shared memory architecture the last step does not involve data transfer). A functional transform based on user input is optionally applied via GPU kernels, and transformed map stored into another GPU buffer of the same size as original map. Meanwhile, map data is dispatched for sorting and ranking in background for construction of CDF, ranked statistics, and equalized and normalized maps. As sorting algorithms are generally hard to parallelize efficiently, this step is carried out by CPU, albeit ranking of all of the loaded maps is dispatched concurrently to exploit CPU parallelism to the extent possible. Once constructed, ranked map is loaded into another GPU buffer.

Once the data is loaded into GPU memory, data values can be mapped into full color representation suitable for rendering. Color mapper uses internal HEALPix representation of a sphere as 12 faces covered by square bitmaps as output, producing a 2D texture array. Texture array (as opposed to array of textures) is an atomic data type supported on many GPU architectures, and could employ advanced memory management techniques such as hardware swizzling to improve performance, with calling function being isolated from particular details by samplers. Color mapping is also hardware accelerated, with user-specified

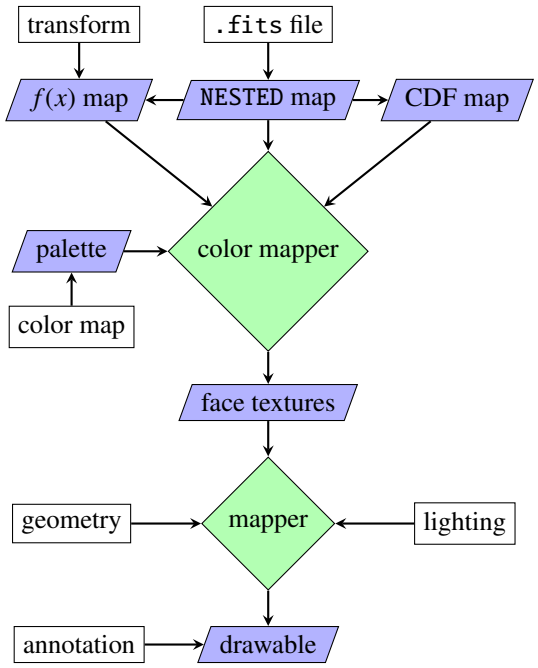

Fig. 2. Schematic of the data flow in HEALPix Viewer. Rectangles represent user input, blue-filled trapezoids represent GPU buffers and textures, green-filled diamonds represent GPU kernels. Lazy evaluation is used in color mapper, which is the main workload for high resolution maps. Geometry mapper runs at output resolution, and is cheap.

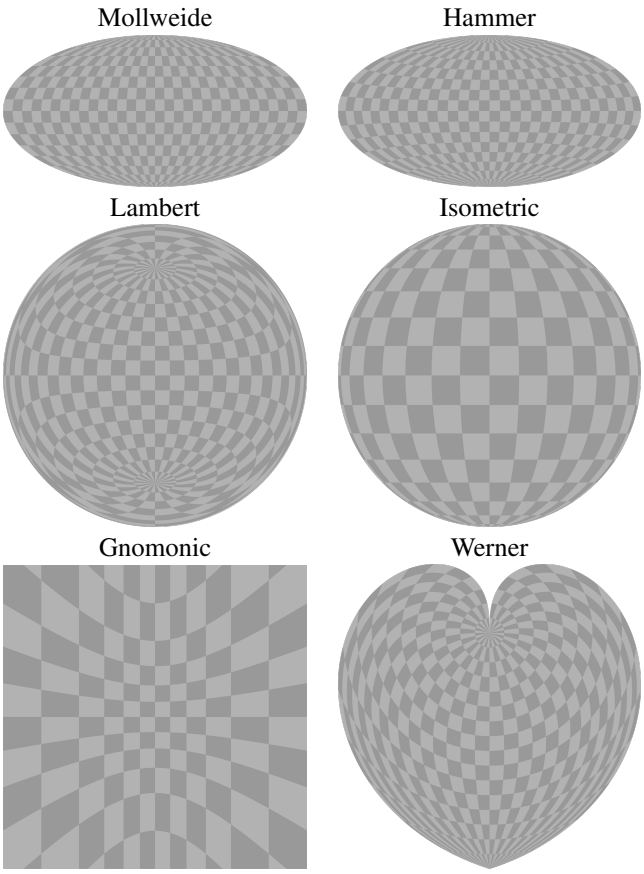

Fig. 3. Some of the projections available in HEALPix Viewer. Checkerboard pattern corresponds to the lines of constant latitude and longitude, as projected onto a two-dimensional screen plane from equator and zero longitude viewpoint.

color palette preloaded as 1D texture, and hardware linear interpolation sampler used to map data values into pixel color. Dispatch of the color mapper kernel is done as a 3D wavefront in x, y, and face index. This would usually result in a fairly localized data buffer access pattern, as in `NESTED` ordering pixels are stored in sequential face, bit-interleaved x, y order (also known as Morton ordering). No particular effort has been put into color management, with RGBA pixel values interpreted in device colorspace (usually sRGB), which is the convention established by `map2gif` in the original HEALPix code.

All of the above is done via lazy evaluation, i.e. data is processed only when parameters change, and operates at full map resolution. In contrast, user-facing view is updated at hardware monitor refresh rate (usually 60fps on most modern Macs) to present smooth animations, with rendering done directly into the drawable presented to the user. This is carried out at the screen resolution, and would effectively be down-sampling of the data for Planck (nside = 2048) and especially Simons Observatory (nside = 8192) maps. A down-sampling strategy always involves a compromise between what is kept versus what is filtered out. In HEALPix viewer, nearest neighbour sampling was chosen for the reason that it preserves 1-point PDF of the high-resolution data. Oversampling of the output for export is optionally available (which produces visually more pleasing maps), and is implemented via Lanczos interpolation, once again in hardware.

As all the user-facing images have planar geometry, projection of a sphere must be done, and it is carried out by a geometry mapper. HEALPix Viewer supports many projections out of the box, some of which are illustrated in Figure 3. For complete list and projection formulae, refer to Appendix A. Implementing a new one is a matter of a few lines of code, due to modular design. The screen coordinates (X, Y) get converted to cartesian plane ones (x, y) via affine transformation

$$\begin{bmatrix} x \\ y \end{bmatrix} = \mathbb{M} \cdot \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}, \tag{1}$$

where transform $\mathbb{M}$ is a 2x3 matrix precomputed on CPU side and passed as a constant parameter to geometry mapper kernels. It depends on screen resolution, viewport size, zoom level, and desired output alignment. It can also incorporate coordinate flips (screen Y direction is usually upward, while bitmap Y direction is downward), viewpoint inversion (looking at the sphere from inside versus outside effectively flips x direction), and overall rotation of the output image (currently not used). The plane coordinates are then mapped in the geometry mapper kernel to a unit 3D vector $\boldsymbol{u}$ via inverse spherical projection $\boldsymbol{u} = p(x, y)$, and if the supplied values are outside of bounds of projected sphere, background color is used for output.

As one would often like to rotate the sphere to view a particular location, a rotation matrix $\mathbb{R}$ is applied to vector $\boldsymbol{u}$, with

$$\boldsymbol{v} = \mathbb{R} \cdot \boldsymbol{u} \tag{2}$$

finally used to lookup the pixel color in 2D texture array produced by color mapper. Optionally, lighting effects could be applied, with overall pixel brightness depending on light direction $\boldsymbol{l}$ via scalar product $\boldsymbol{l} \cdot \boldsymbol{u}$ (light location is fixed in the viewer space and is not rotated with the map).

Rotation matrix $\mathbb{M}$ is usually parametrized by Euler angles, which are latitude, longitude, and azimuth of the viewpoint (latitude and longitude can be specified in `map2gif`, but not azimuth). Details of this rotation matrix parametrization, along with the way to extract Euler angles from rotation matrix are given in Appendix B.1. One of the features I wanted to implement is a smooth animation between two different orientations when changing the view. This is accomplished via evolution of generator of rotation as a damped simple harmonic oscillator. Rotation matrix in three dimensions can be represented as

$$\mathbb{R} = \exp[\mathbb{W}], \tag{3}$$

with antisymmetric 3x3 matrix $\mathbb{W}$ corresponding to a dual 3-vector $\boldsymbol{w} = *\mathbb{W}$, known as generator of rotation. This representation is quite familiar, as it appears in rotation of a solid body as angular velocity $\boldsymbol{\omega}$. Details of this representation are given in Appendix B.2, with formulae for forward and inverse transformations. When user specifies a new target orientation $\boldsymbol{t}$, the current viewpoint generator $\boldsymbol{w}$ is evolved as

$$\ddot{\boldsymbol{w}} + \gamma\dot{\boldsymbol{w}} + \kappa(\boldsymbol{w} - \boldsymbol{t}) = 0, \tag{4}$$

with rotation matrix $\mathbb{R}$ updated correspondingly for each frame, using real system time in case frame cadence is not uniform. Parameters γ and κ set timescales of damping and oscillation correspondingly, with the values chosen so that evolution is slightly overdamped with timescale of a few seconds. Evolution itself is computed with 6-th order symplectic integration scheme Yoshida (1990); Candy & Rozmus (1991), vectorized on CPU side, which is not strictly necessary but is easy to implement and fast. Details of this are presented in Appendix C.

Finally, the rendered image might be saved into a PNG file for use with other software or as paper figures. PNG format is chosen because it supports full 24-bit color and transparency, in which it is superior to GIF format used by `map2gif`. Formats based on lossy Fourier compression such as JPEG are not suitable for high-quality output due to compression artifacts. Rendering for export is done by the same GPU code as for the screen, except affine transformation parameters are adjusted accordingly, and rendering is done into 32-bit integer RGBA context. Colorbar render can be added to the figure if desired, using the same strategy. One might also wish to have a text annotation, which is rendered using Core Text framework, and can use any fonts available on the system (San Francisco Compact is used by default). Resulting texture is copied into CPU bitmap, which is output as an image file using Core Image routines.

4. Data analysis

While sophisticated data analysis is not the primary goal for HEALPix Viewer and is best carried out by dedicated code, some data analysis facilities are provided for convenience and improved visualization. They are currently restricted to 1-point transformations, where a pixel value x gets mapped into a value of a function $f(x)$. The list of implemented transformations is as follows:

- $f(x) = \ln(x - \mu)$
- $f(x) = \mathrm{asinh}[(x - \mu)/\sigma]$
- $f(x) = \mathrm{atan}[(x - \mu)/\sigma]$
- $f(x) = \tanh[(x - \mu)/\sigma]$

- PDF equalization
- PDF normalization

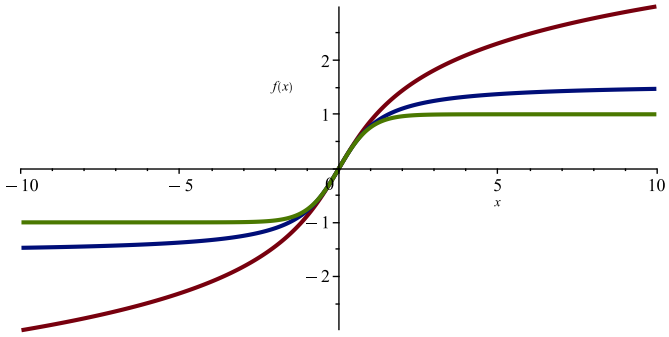

Fig. 4. Three range-limiting transforms used in HEALPix Viewer. Dark brown line is asinh transform, dark blue is atan transfrom, and dark green is tanh transfrom. All three have the same derivative near the origin, but differ in asymtotics for large argument values (logarithmic, rational, and exponential fall-off correspondingly).

Parameter μ corresponds to the nominal center of the distribution, while parameter σ corresponds to overall distribution width, and is set on logarithmic scale in the user interface. Logarithmic scaling (first transform on the list) is widely used to compress dynamical range of data for visualization, and is in particular useful for representing intensity of the higher frequency maps in CMB experiments (such as 353GHz, 545GHz, and 857GHz data in Planck) as foregrounds in galactic plane are vastly brighter than high-latitude diffuse emission. Logarithmic scaling has hard lower cut-off, with data $x \le \mu$ resulting in non-normalizable output (colored as `NaN` color in HEALPix Viewer).

The next three transforms correspond to symmetric and increasingly harder clamping of input data, as illustrated in Figure 4. The last two transforms deserve special mention, as they are rarely if ever seen in CMB literature, but are common in image processing and very good for data visualization. PDF equalization applies a function which produces output with uniform 1-point PDF, while normalization extends on that by producing Gaussian 1-point PDF. By itself, it's a useful preprocessing step to apply to data for multi-variate and multi-point analysis (e.g. a notion of copula in statistics), but it also serves for producing almost optimal color gradation in color-mapped image, providing a one-button "make it pretty" option.

CDF estimators and ranked maps are produced by indexing the map data by its value, which is done using quadsort algorithm, slightly modified to be thread-safe. While not as widely known as quicksort, quadsort is stable (preserves ordering of equal values), on average a bit faster, and scales vastly better ($O(n \log n)$ versus $O(n^2)$) for the worst case (ordered data, which appears all the time in masked maps). CDF estimator is then easily computed as

$$F(x) = \{\#\text{ of pixels with value} < x\}/n \tag{5}$$

while output map with uniform PDF is achieved by transforming the ranked map pixel p into

$$f(p) = \{\text{rank of pixel } p\}/n. \tag{6}$$

Current implementation has a side effect of having uniform value areas ranked in pixel order (since quadsort is stable), but fixing that is more trouble than it's worth (for CMB data it would only affect masked areas). Gaussian PDF can be produced by applying inverse Gaussian CDF transformation to uniform PDF map

$$f(x) = \sqrt{2}\,\mathrm{erf}^{-1}(2x - 1), \tag{7}$$

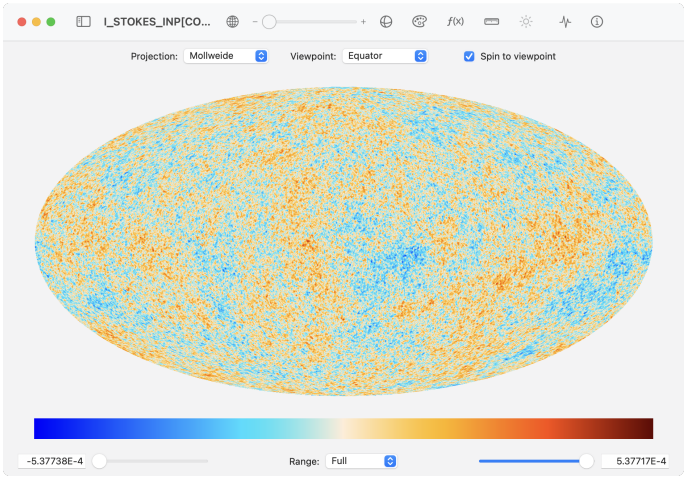

Fig. 5. HEALPix Viewer user interface displaying inpainted Planck 2018 SMICA CMB map, with top and bottom toolbars exposed.

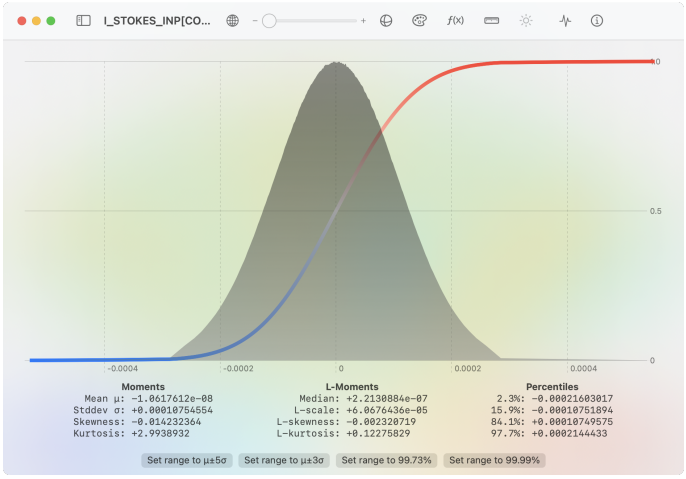

Fig. 6. HEALPix Viewer statistics overlay displaying statistics summary of inpainted Planck 2018 SMICA CMB map.

which requires a bit of extra work to implement as erf and its inverse are not available in either GPU nor CPU libraries Giles (2012). Details of that are explained in Appendix D.

5. User interface

User interface paradigm is based around top toolbar which allows user to adjust render parameters, and bottom toolbar which displays the colorbar and adjusts color mapping parameters. Top toolbar is hierarchical, and has several parameter groups which can be selected by buttons in the main window title bar. The top toolbar parameter hierarchy is as follows:

- map projection:
  - projection choice
  - viewpoint presets
  - spin animations
- view orientation:
  - viewpoint latitude
  - viewpoint longitude
  - viewpoint azimuth
- color scheme:
  - color palette
  - below min color
  - above max color
  - `NaN` color
  - background color
- data transform:
  - function $f(x)$
  - parameter μ
  - parameter $\ln\sigma$
- lighting effects:
  - light latitude
  - light longitude
  - effect amount

Screenshot of the user interface with both top and bottom toolbars exposed is shown in Figure 5. Both toolbars can be hidden if screen space is at a premium. Lightning effects have to be enabled in View menu if desired. Also in View menu are options to control application appearance (light or dark mode), and to enable cursor readout. Bottom toolbar shows the colorbar and the controls for its bounds. Colorbar range can be restricted to symmetric, positive, or negative if desired. Left toolbar (which also can be hidden, and is hidden by default) shows the list of loaded map data, and allows you to chose which one to display. More than one window can be opened simultaneously, with independent settings defaulting to the ones in Data menu. If Data menu selection changes, it will be applied to active window only.

In addition to toolbars, there are two information overlays available - data statistics and FITS header. They can be accessed by buttons in the title bar, and will temporary cover map view with a semi-transparent background. Data statistics overlay displays PDF and CDF (discontinuities of which are represented as a bar graph), as well as moments of the distribution (mean, standard deviation, skewness and kurtosis), L-moments Hosking (1990) (median, L-scale, L-skewness, and L-kurtosis), and percentile brackets of the distribution corresponding to 1σ and 2σ of Gaussian one, as shown in Figure 6. Buttons to quickly set colorbar range to preset values are added for convenience. FITS header overlay displays complete header of the FITS file, with all the cards and comments unaltered and unparsed. This is useful for debugging, as well as if one wants to check metadata not supported by HEALPix Viewer natively.

Main map view responds to common gestures - two finger pinch to zoom, two finger rotation to change azimuth, right click to center on a clicked location (rotating along a geodesic), and option-right click (rotating while keeping azimuth fixed). If animations are enabled, two-finger scroll gesture will kick the sphere in the direction of the scroll, while keeping target location which it will eventually return to. If cursor readout is enabled in the View menu, the coordinates and map value under the cursor will be displayed at the top of the map view.

Drag and drop is supported to enable simple integration with other applications, such as Powerpoint or Keynote for presentations. Drop HEALPix file into main window to load it, and drag the map and colorbar (individually) to the application which accepts image files as a drop. Dragged map will be exported at current screen size. If you want custom resolution output or oversampling, export the image via File menu.

6. Performance and limitations

From inception, the idea was to make an *interactive* data visualization and exploration tool, which means data processing has to be fast enough to keep up with user inputs. That design

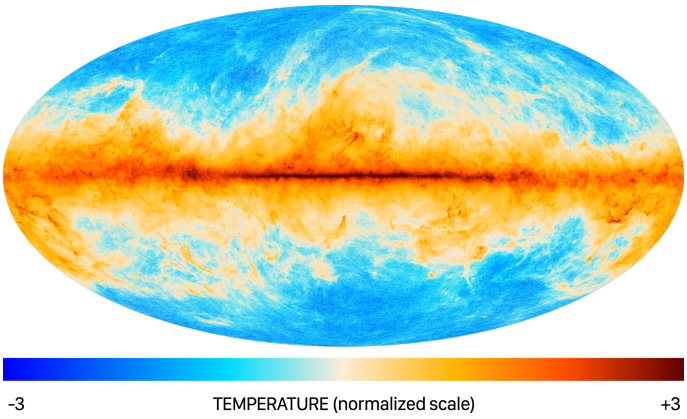

Fig. 7. Planck 353GHz dust map (353GHz data with CMB and point source contributions subtracted) as rendered by HEALPix Viewer for export with 4x oversampling. Colorbar corresponds to normalized map, and is clamped at $\pm 3\sigma$.

goal was achieved in HEALPix Viewer. Its processing pipeline is vastly more efficient than tweaking `map2gif` parameters and re-rendering (which involves map load from disk every time `map2gif` is run), and outperforms CPU-bound solutions (such as `healpy` ran in an interactive envirnonment) by a large margin. In fact, all of the data processing except initial map load and ranking appears near-instantaneous to user, with map view responding to parameter changes in real time (except for initial texture allocation in *really* large maps). Geometry changes involve only rendering at screen resolution (which is not that high compared to allowed map size), and run at full 60fps at any map size tested. In fact, geometry rendering is so efficient it can be ran continuously without substantial power draw by the GPU.

Despite being fast, the rendering pipeline is accurate to single precision (limited by 32-bit floats used in processing). HEALPix pixel boundaries and sphere geometry projections are accurate even at the highest zoom level (a factor of 1024), unlike some implementation which use OpenGL vertex shaders (which usually involves approximating sphere geometry by polygons). Data statistics and moments are derived from down-sampled CDF (currently set to 4096 samples, but easily changed), so are less precise than what could be achieved in a more sophisticated analysis. Exported image is in full 32-bit color supporting transparency, and oversampling up to a factor of 4 is available to improve rendering quality of text fonts and noisy maps if desired. Typical output is shown in Figure 7.

The main user-facing delay is initial load of data from the HEALPix file, which is blocking and comes with indefinite progress indicator in the user interface. This is mostly limited by `cfitsio` implementation and disk access performance, and can range around $350 - 775$ms per `NESTED` single precision map at nside $= 2048$ resolution (read from solid state disk on M1 Max laptop). Down-converting, reordering, and sign flips of polarization data are done in a single pass by a number of hard-coded C kernels for performance (Swift array implementation is fairly fast, but enforces bounds check on every access, which is quite noticeable overhead at data sizes involved). Initial load time is affected by data configuration, in particular `RING` to `NESTED` reordering can add up to 600ms overhead per map. Further optimization is not feasible, as it would require rewriting large parts of `cfitsio`, which is a fairly monumental task. Sorting and ranking of loaded maps is also expensive, but is done concurrently in the background, which hides the processing delay from the user to some extent (features which depend on ranked data get released as soon as data is available).

HEALPix Viewer is capable of dealing with extremely large maps without complexity associated with hierarchical data representation. Main limitation on map size comes not from rendering pipeline performance, but physical size of VRAM and hard-coded limits on texture sizes in GPU libraries. As all the data gets loaded into GPU memory, it could easily exceed VRAM amount of consumer video cards. A single HEALPix map has $n_{\rm px} = 12\,\mathrm{nside}^2$ pixels, at 4 bytes per pixel in a single precision buffer (of which 3 are used). A color-mapped texture has $4n_{\rm px}$ floating point RGBA values, at 4 bytes per value for single precision. Total maximal amount of VRAM needed per map is then $336\,\mathrm{nside}^2$ bytes, or 1.3Gb per nside $= 2048$ map, plus whatever memory needed for UI and output rendering (normally much less). This is not a problem on shared memory architectures such as M1 and M2 (given enough system memory is available), but could be a serious limit on older hardware with separate GPU video cards (VRAM amounts of 4Gb are common). Memory footprint can be significantly reduced by using 8-bit integer textures (at expense of slightly worse color gradation), which may be added as an option in the future.

Second limit comes from hard-coded maximal texture size for Metal libraries, which is 16384 pixels, and limits nside of the map to 16384 and output bitmap resolution to $16384/n_{\rm os}$. This means that Simons Observatory maps are supported at full resolution (nside $= 8192$), and maximal printed output size would be 2.8×1.4m at 150dpi resolution. nside $= 8192$ maps were tested on commodity laptop used for development (M1 Max with 64Gb system memory), and geometry rendering still runs at full 60fps. Data transforms have noticeable lag, but are still quite responsive except for a blocking delay on initial function transform buffer allocation, which takes a couple of seconds. Perceived performance could be improved by operating on downsampled proxy map during active user input phase (e.g. slider adjustment), with full resolution processed as soon as parameter values stop changing. The code was tested on both higher- and lower-end hardware as well, but permutations tested are limited to my hardware collection. Community feedback would be very welcome on this.

Multiple GPUs are currently not supported. While the data processing pipeline is trivially parallelizable, bookkeeping and memory management details required would have to be hand-managed and would slow down the development. This is not a problem on current SoC hardware, but would be an issue on older machines with discrete or external GPUs. The first GPU on the system list is used for rendering, which might not be optimal, and the strategy could be revised in the future versions.

HEALPix viewer binary is built against macOS 12 libraries, with exception of data statistics overlay which requires plotting libraries which only became available in macOS 13 (and is disabled on earlier versions). While most of the GPU kernels can be easily ported to OpenCL to be ran on other operating systems, the rest of the source code is tightly coupled to SwiftUI libraries, and porting user interface code would involve complete re-write. For this reason, support for Windows and Linux is not planned.

7. Conclusions

Cosmic microwave background is one of the pillars of precision cosmology, and a number of new high-resolution surveys are being built or planned. Multi-band temperature and polarization maps are needed to remove astrophysical backgrounds and reconstruct primordial CMB fluctuations. CMB experiments not only constrain a handful of cosmological parameters, but serve

as a window to early universe physics (in particular, the search for primordial gravitational waves is currently under way with attempts to detect B-mode polarization) and provide astrophysical insights with reconstructed foreground maps (for example, information on magnetic field of our own galaxy). Full or partial sky maps are produced from time-ordered data by sophisticated mapping algorithms, and main data analysis is carried out on this reduced data set. Storing or processing it requires a way to pixelize the sphere, and HEALPix (Hierarchical Equal Area isoLatitude Pixelation) is widely used for that purpose.

This paper presents a new interactive visualization application developed for macOS using latest software and technology, which I called HEALPix Viewer for the lack of a better name. It is fully GPU accelerated, and can handle extremely large maps (which is a common trend in future experiments, for example Simons Observatory would produce almost giga-pixel maps). Software design was done from scratch and uses modern language features of Swift to full advantage, making the code modular and easy to maintain. Data pipeline was designed with performance in mind, and is fast enough to process and render nside = 8192 maps in real time on a laptop, given enough memory. User interface makes many common tasks easy and convenient, for example rendered maps can be dropped directly into a Keynote or Powerpoint presentation. HEALPix Viewer is distributed as an universal application that can natively run on Intel and Apple Silicon hardware for easy installation if you just want to use it, and has complete source code available if you want to add new features or use the code in other projects.

Development is ongoing, and further improvements could be made in the future. In particular, visualization of polarization and vector fields via line integral convolution Cabral & Leedom (1993) is planned. Feedback and feature requests are welcome, and should be directed to the author.

Acknowledgements. This work was supported in part by NSERC Discovery Grant "Testing fundamental physics with B-modes of Cosmic Microwave Background anisotropy". The author thanks Sigurd K. Næss for stimulating discussions, and everyone who participated in beta testing.

References

Abazajian, K., Addison, G., Adshead, P., et al. 2019, arXiv e-prints, arXiv:1907.04473

Addison, G. E., Switzer, E. R., Greason, M. R., et al. 2019, arXiv e-prints, arXiv:1905.08667

Ade, P., Aguirre, J., Ahmed, Z., et al. 2019, J. Cosmology Astropart. Phys., 2019, 056

Bennett, C. L., Larson, D., Weiland, J. L., et al. 2013, ApJS, 208, 20

Cabral, B. & Leedom, L. C. 1993, Imaging Vector Fields Using Line Integral Convolution

Candy, J. & Rozmus, W. 1991, Journal of Computational Physics, 92, 230

Dicke, R. H., Peebles, P. J. E., Roll, P. G., & Wilkinson, D. T. 1965, ApJ, 142, 414

Efstathiou, G., Bond, J. R., & White, S. D. M. 1992, MNRAS, 258, 1P

Fernique, P., Oberto, A., Boch, T., & Bonnarel, F. 2010, in Astronomical Society of the Pacific Conference Series, Vol. 434, Astronomical Data Analysis Software and Systems XIX, ed. Y. Mizumoto, K. I. Morita, & M. Ohishi, 163

Fryer, D., Li, M., & Olenko, A. 2019, arXiv e-prints, arXiv:1907.05648

Giles, M. 2012, in GPU Computing Gems Jade Edition, ed. W.-M. W. Hwu (Morgan Kaufmann Publishers)

Górski, K. M. & Hivon, E. 2011, HEALPix: Hierarchical Equal Area isoLatitude Pixelization of a sphere, Astrophysics Source Code Library, record ascl:1107.018

Górski, K. M., Hivon, E., Banday, A. J., et al. 2005, ApJ, 622, 759

Gorski, K. M., Wandelt, B. D., Hansen, F. K., Hivon, E., & Banday, A. J. 1999, arXiv e-prints, astro

Hanany, S., Alvarez, M., Artis, E., et al. 2019, arXiv e-prints, arXiv:1902.10541

Hazumi, M., Borrill, J., Chinone, Y., et al. 2012, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 8442, Space Telescopes and Instrumentation 2012: Optical, Infrared, and Millimeter Wave, ed. M. C. Clampin, G. G. Fazio, H. A. MacEwen, & J. Oschmann, Jacobus M., 844219

Hensley, B. S., Clark, S. E., Fanfani, V., et al. 2022, ApJ, 929, 166

Hinshaw, G., Larson, D., Komatsu, E., et al. 2013, ApJS, 208, 19

Hivon, E., Reinecke, M., & Gorski, K. M. 2015, in IAU General Assembly, Vol. 29, 2247779

Hosking, J. R. M. 1990, Journal of the Royal Statistical Society. Series B (Methodological), 52, 105

Joliet, E., O'Mullane, W., Górski, K. M., et al. 2008, in Astronomical Society of the Pacific Conference Series, Vol. 394, Astronomical Data Analysis Software and Systems XVII, ed. R. W. Argyle, P. S. Bunclark, & J. R. Lewis, 319

Kovac, J. M., Leitch, E. M., Pryke, C., et al. 2002, Nature, 420, 772

Mather, J. C., Cheng, E. S., Eplee, R. E., J., et al. 1990, ApJ, 354, L37

Matsumura, T., Akiba, Y., Borrill, J., et al. 2014, Journal of Low Temperature Physics, 176, 733

Mingaliev, S. M. & Stolyarov, V. A. 2010, Astrophysical Bulletin, 65, 296

Planck Collaboration, Aghanim, N., Akrami, Y., et al. 2020a, A&A, 641, A1

Planck Collaboration, Aghanim, N., Akrami, Y., et al. 2020b, A&A, 641, A6

Planck Collaboration, Akrami, Y., Arroja, F., et al. 2020c, A&A, 641, A10

Planck Collaboration, Akrami, Y., Arroja, F., et al. 2020d, A&A, 641, A9

Planck Collaboration, Akrami, Y., Ashdown, M., et al. 2020e, A&A, 641, A7

Portsmouth, J. 2011, CMBview: A Mac OS X program for viewing HEALPix-format sky map data on a sphere, Astrophysics Source Code Library, record ascl:1112.011

Smoot, G. F., Bennett, C. L., Kogut, A., et al. 1992, ApJ, 396, L1

Tomasi, M. & Li, Z. 2021, Healpix.jl: Julia-only port of the HEALPix library, Astrophysics Source Code Library, record ascl:2109.028

Yoshida, H. 1990, Physics Letters A, 150, 262

Zonca, A., Singer, L., Lenz, D., et al. 2020, healpy: Python wrapper for HEALPix, Astrophysics Source Code Library, record ascl:2008.022

Appendix A: Projections of a sphere

A.1. Mollweide projection

Mollweide projection (commonly used in CMB literature) maps an entire sphere onto an ellipse of 2:1 aspect ratio. It is an equal-area pseudocylindrical projection which keeps iso-latitude lines straight and horizontal, at expense of shape distortion near the boundaries. Inverse transformation from Cartesian plane (x, y) to spherical coordinates (θ, ϕ) is

$$\xi = \arcsin y, \quad \theta = \arccos \frac{2\xi + \sin(2\xi)}{\pi}, \quad \phi = \frac{\pi x}{2\cos\xi}. \tag{A.1}$$

Projection bounds are set by $|y| \le 1$ and $|\phi| \le \pi$, with projection extent $|x| \le 2$ and $|y| \le 1$.

A.2. Hammer projection

Visually similar to Mollweide projection, Hammer projection trades reduced shape distortion for iso-latitude lines being curved. It is also equal area and maps an entire sphere onto an ellipse of 2:1 aspect ratio. Inverse transformation from Cartesian plane (x, y) to spherical coordinates (θ, ϕ) is

$$q = 1 - \tfrac{x^2}{16} - \tfrac{y^2}{4}, \quad \theta = \arccos\left(q^{\frac{1}{2}} y\right), \quad \phi = 2\arctan \tfrac{q^{\frac{1}{2}} x}{2(2q-1)}. \tag{A.2}$$

Projection bounds are set by $q \ge 1/2$, with projection extent $|x| \le \sqrt{8}$ and $|y| \le \sqrt{2}$.

A.3. Lambert projection

Also known as Lambert azimuthal projection, it maps an entire sphere onto a disk. It is also equal-area projection, with inverse transformation from Cartesian plane (x, y) to unit vector $\boldsymbol{n}$

$$q = 1 - (x^2 + y^2)/4, \quad \hat{\boldsymbol{n}} = \left[2q - 1, q^{\frac{1}{2}} x, q^{\frac{1}{2}} y\right]. \tag{A.3}$$

Projection bounds are set by $q \ge 0$, with projection extent $|x| \le 2$ and $|y| \le 2$.

A.4. Isometric projection

Isometric, or perhaps more appropriately orthographic, projection is the view of a sphere by an observer at infinity. It maps half of a sphere onto a disk. This is the most familiar projection, as it corresponds to looking at a physical sphere. Inverse transformation from Cartesian plane (x, y) to unit vector $\boldsymbol{n}$ is

$$q = 1 - x^2 - y^2, \quad \hat{\boldsymbol{n}} = \left[q^{\frac{1}{2}}, x, y\right]. \tag{A.4}$$

Projection bounds are set by $q \ge 0$, with projection extent $|x| \le 1$ and $|y| \le 1$.

A.5. Gnomonic projection

Gnomonic projection is defined by mapping the sphere to a tangent plane in the direction of radial vector. It maps great circles to straight lines (meaning projected straight lines are geodesic), and maps half of a sphere onto an infinite plane. Inverse transformation from Cartesian plane (x, y) to unit vector $\boldsymbol{n}$ is

$$\hat{\boldsymbol{n}} = [1, x, y] \,/\, \sqrt{1 + x^2 + y^2}. \tag{A.5}$$

A.6. Mercator projection

Mercator projection is a cylindrical projection quite often used in cartography and navigation. It is conformal and maps the entire sphere onto an infinite strip. Inverse transformation from Cartesian plane (x, y) to spherical coordinates (θ, ϕ) is

$$\phi = x, \quad \theta = \frac{\pi}{2} - \arctan(\sinh y). \tag{A.6}$$

Projection bounds are set by $|\phi| \le \pi$, with infinite extent in y direction.

A.7. Cylindrical projection

Cylindrical, or plate carrée, projection is a trivial mapping of spherical coordinates onto a plane. It is neither equal-area nor conformal, but it maps parallels and meridians into straight lines. It maps an entire sphere onto a rectangle of 2:1 aspect ratio. Distortion near poles is extreme. Inverse transformation from Cartesian plane (x, y) to spherical coordinates (θ, ϕ) is

$$\phi = x, \quad \theta = \frac{\pi}{2} - y. \tag{A.7}$$

Projection bounds are set by $|\phi| \le \pi$, $0 \le \theta \le \pi$, with corresponding extent $|x| \le \pi$ and $|y| \le \pi/2$.

A.8. Werner projection

Werner projection is an equal-area pseudoconic projection which maps the entire sphere into a heart shape. Least commonly used of the projections listed here, it nevertheless was known from circa 16th century. Inverse transformation from Cartesian plane (x, y) to spherical coordinates (θ, ϕ) is

$$\rho = \sqrt{x^2 + y^2}, \quad \phi = \frac{\rho}{\sin\rho} \operatorname{atan2}(x, -y), \quad \theta = \rho. \tag{A.8}$$

Projection bounds are set by $|\phi| \le \pi$, $0 \le \theta \le \pi$, with projection extent being asymmetrical in y direction and transcendental. After shifting y coordinate by 1.111983413 to center the image, the extent is almost exactly at 1:1 aspect ratio, with $|x| \le 2.021610497$ and $|y| \le 2.029609241$. It is implemented because it looks cool.

Appendix B: Representations of rotation group

B.1. Euler angles

Rotation of a sphere which brings a viewpoint with a given latitude ϑ, longitude ϕ, and azimuth ψ to the direct line of sight can be parametrized as a composition of rotations in xy, xz, and yz planes. These are usually known as Euler angles, with

$$\mathbb{R}_{xz} = \begin{bmatrix} \cos\vartheta & 0 & -\sin\vartheta \\ 0 & 1 & 0 \\ \sin\vartheta & 0 & \cos\vartheta \end{bmatrix}, \tag{B.1}$$

$$\mathbb{R}_{xy} = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{B.2}$$

$$\mathbb{R}_{yz} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix}. \tag{B.3}$$

Note that some of the signs are different from the usual notation, because we are interested in rotation which brings a given location into a fixed position, not the other way around. Order of

rotations matters, as they do not commute. The composite rotation matrix is

$$\mathbb{R} = \mathbb{R}_{xy}\,\mathbb{R}_{xz}\,\mathbb{R}_{yz} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}, \tag{B.4}$$

with matrix elements being

$$R_{11} = \cos\vartheta\cos\phi, \tag{B.5}$$
$$R_{12} = -\sin\vartheta\cos\phi\sin\psi - \sin\phi\cos\psi, \tag{B.6}$$
$$R_{13} = -\sin\vartheta\cos\phi\cos\psi + \sin\phi\sin\psi, \tag{B.7}$$
$$R_{21} = \cos\vartheta\sin\phi, \tag{B.8}$$
$$R_{22} = -\sin\vartheta\sin\phi\sin\psi + \cos\phi\cos\psi, \tag{B.9}$$
$$R_{23} = -\sin\vartheta\sin\phi\cos\psi - \cos\phi\sin\psi, \tag{B.10}$$
$$R_{31} = \sin\vartheta, \tag{B.11}$$
$$R_{32} = \cos\vartheta\sin\psi, \tag{B.12}$$
$$R_{33} = \cos\vartheta\cos\psi. \tag{B.13}$$

It should be remembered that in Swift, vectorized matrix types like `float3x3` are stored column-wise, with indexing starting from zero, so matrix element R_{31} would actually be referred to as `R[0,2]` in the code, and so on. Actual rotation matrix is evaluated as a product B.4 in the code, not long-form expressions of (B.5–B.13), which is vectorized and more efficient.

Euler angles can be extracted from a given rotation matrix $\mathbb{R}$

$$\vartheta = \arcsin R_{31}, \tag{B.14}$$
$$\phi = \mathrm{atan2}(R_{21}, R_{11}), \tag{B.15}$$
$$\psi = \mathrm{atan2}(R_{32}, R_{33}), \tag{B.16}$$

although one has to be mindful of the branches. The arcsin expression (B.14) is meant to have $|\vartheta| < \pi/2$, corresponding to spherical coordinate $\theta = \pi/2 - \vartheta$ range of $0 < \theta < \pi$. The last two expressions go degenerate when $|\vartheta| = \pm\pi/2$ (i.e. at the poles), for which case the rotation matrix reads

$$\mathbb{R} = \begin{bmatrix} 0 & -\sin(\phi \pm \psi) & \mp\cos(\phi \pm \psi) \\ 0 & \cos(\phi \pm \psi) & \mp\sin(\phi \pm \psi) \\ \pm 1 & 0 & 0 \end{bmatrix}.$$

Longitude and azimuth cannot be distinguished at the poles, and only their combination

$$\phi \pm \psi = \mathrm{atan2}\,(-R_{12}, R_{22}) \tag{B.17}$$

is determined. For definiteness, one can choose $\psi = 0$ there.

B.2. Generators of rotation

Special orthogonal transformations form a Lie group, and as such are representable by corresponding Lie algebra generators. A rotation matrix $\mathbb{R}$ is thus representable by exponent of an antisymmetric matrix $\mathbb{W}$

$$\mathbb{R} = \exp[\mathbb{W}] \equiv \sum_{n=0}^{\infty} \frac{\mathbb{W}^n}{n!}. \tag{B.18}$$

In three dimensions, 3x3 antisymmetric matrix $\mathbb{W}$ has only 3 independent components, and can be written as a dual of three-vector $\boldsymbol{w}$

$$\mathbb{W} \equiv *\boldsymbol{w} = \begin{bmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{bmatrix}. \tag{B.19}$$

A specific property of 3x3 antisymmetric matrices which simplifies things a lot is that Krylov space has low dimension, since

$$\mathbb{W}^3 = -\|\mathbb{W}\|^2\,\mathbb{W}, \tag{B.20}$$

where matrix norm is defined as

$$\|\mathbb{W}\|^2 \equiv -\frac{1}{2}\,\mathrm{Tr}\,\mathbb{W}^2 = |\boldsymbol{w}|^2. \tag{B.21}$$

Resumming power series in (B.18), one obtains

$$\mathbb{R} = \exp[\mathbb{W}] = \mathbb{I} + \frac{\sin w}{w}\,\mathbb{W} + 2\,\frac{\sin^2\frac{w}{2}}{w^2}\,\mathbb{W}^2, \tag{B.22}$$

where $\mathbb{I}$ is identity matrix and $w = |\boldsymbol{w}|$. Conversely, one can read off generator vector from a given rotation matrix $\mathbb{R}$ by

$$\boldsymbol{w} = \mathrm{atan2}\left(s, \tfrac{1}{2}(t-1)\right)\hat{\boldsymbol{s}}, \tag{B.23}$$

where

$$\boldsymbol{s} = *\mathbb{R} \equiv \tfrac{1}{2}\left[R_{32} - R_{23}, R_{13} - R_{31}, R_{21} - R_{12}\right], \tag{B.24}$$
$$t = \mathrm{Tr}\,\mathbb{R} = R_{11} + R_{22} + R_{33}. \tag{B.25}$$

Generator of rotation $\boldsymbol{w}$ represents rotation by an angle w around the axis $\hat{\boldsymbol{w}}$, and is defined up to an integer multiple of 2π in length. When setting the target rotation generator, one should pick the value closest to the current one among possible equivalent choices to ensure the shortest rotation between the two.

Appendix C: Sphere dynamics

Transition between different viewpoints specified by user is animated by solving a damped simple harmonic oscillator equation for rotation generator $\boldsymbol{w}$, asymptoting to target location $\boldsymbol{t}$

$$\ddot{\boldsymbol{w}} + \gamma\dot{\boldsymbol{w}} + \kappa(\boldsymbol{w} - \boldsymbol{t}) = 0. \tag{C.1}$$

This can be readily evolved by second-order accurate operator splitting scheme

$$\boldsymbol{w} \mapsto \boldsymbol{w} + \dot{\boldsymbol{w}}\,\tfrac{dt}{2} \tag{C.2}$$
$$\dot{\boldsymbol{w}} \mapsto \dot{\boldsymbol{w}} - \left[\gamma\dot{\boldsymbol{w}} + \kappa(\boldsymbol{w} - \boldsymbol{t})\right]dt \tag{C.3}$$
$$\boldsymbol{w} \mapsto \boldsymbol{w} + \dot{\boldsymbol{w}}\,\tfrac{dt}{2} \tag{C.4}$$

commonly used and known by many names (e.g. leapfrog and kick-drift). It happens to be symplectic for Hamiltonian equations of motion (i.e. a canonical transformation, albeit for approximate Hamiltonian), and the order can be increased by forming operator sandwiches with particularly selected time steps, so higher order commutators would vanish. Among those, 6-th order symplectic scheme first derived by Yoshida (1990) is still cheap and very convenient to implement

$$H_6(\Delta t) = \prod_{-3\le i\le 3} H_2\left(\alpha_{|i|}\Delta t\right), \tag{C.5}$$

where H_2 is the second order evolution step (C.2–C.4), and coefficients $\alpha_{|i|}$ are specifically chosen to be (in quad precision)

$$\alpha_0 = 1.3151863206839112188824972823886251, \tag{C.6}$$
$$\alpha_1 = -1.1776799841788710069641568096431573, \tag{C.7}$$
$$\alpha_2 = 0.2355732133593581336847931829785346 02, \tag{C.8}$$
$$\alpha_3 = 0.7845136104775572638194976338663 49876. \tag{C.9}$$

Appendix D: Inverse error function

To transform uniform distribution into a Gaussian one, one needs to compute an inverse error function. Since this is not a part of standard GPU libraries, a workable implementation is needed. An approximation derived in Giles (2012) is as follows

$$\mathrm{erf}^{-1}(x) = \begin{cases} xp_1(w), & w < 5 \\ xp_2(w), & w \geq 5 \end{cases}, \tag{D.1}$$

where p_1 and p_2 are polynomials of order 8, and

$$w = -\ln(1 - x^2). \tag{D.2}$$

Polynomial p_1 (in Horner form) is evaluated as

```
w = w - 2.5;
p =  2.81022636e-08;
p =  3.43273939e-07 + p*w;
p = -3.5233877e-06  + p*w;
p = -4.39150654e-06 + p*w;
p =  0.00021858087  + p*w;
p = -0.00125372503  + p*w;
p = -0.00417768164  + p*w;
p =  0.246640727    + p*w;
p =  1.50140941     + p*w;
```

while polynomial p_2 (actually of argument $\sqrt{w}$) is

```
w = sqrt(w) - 3.0;
p = -0.000200214257;
p =  0.000100950558 + p*w;
p =  0.00134934322  + p*w;
p = -0.00367342844  + p*w;
p =  0.00573950773  + p*w;
p = -0.0076224613   + p*w;
p =  0.00943887047  + p*w;
p =  1.00167406     + p*w;
p =  2.83297682     + p*w;
```

The approximation is good down to last digit of single precision floats for entire range of argument values, produces the right asymptotics, and is quite cheap to evaluate (involving only 8 multiply-accumulate operators).