Gaia Sky: Navigating the Gaia Catalog

Antoni Sagristà, Stefan Jordan, Thomas Müller, and Filip Sadlo

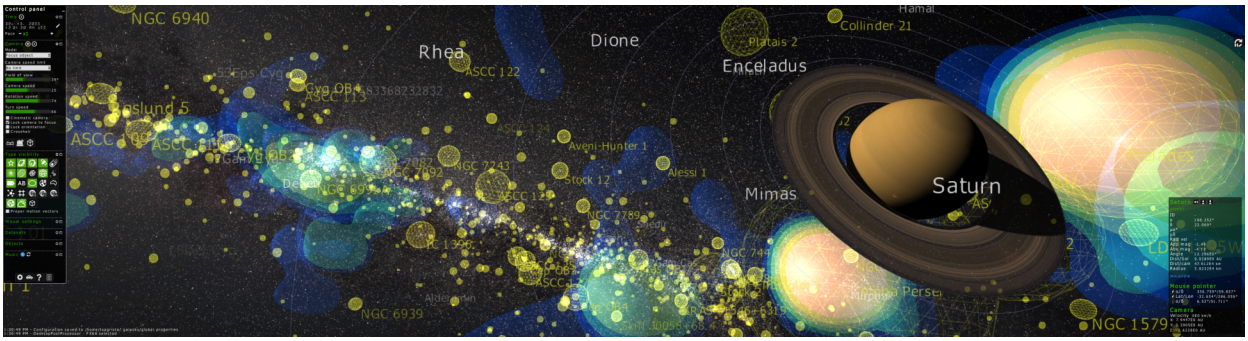


Fig. 1: Screenshot of Gaia Sky with Saturn and some of its moons, the Milky Way Star Clusters catalog indicated by yellowish spherical meshes, semi-transparent isosurfaces of OB star densities, and stars from Gaia showing parts of the Milky Way galaxy.

Abstract—In this paper, we present Gaia Sky, a free and open-source multiplatform 3D Universe system, developed since 2014 in the Data Processing and Analysis Consortium framework of ESA's Gaia mission. Gaia's data release 2 represents the largest catalog of the stars of our Galaxy, comprising 1.3 billion star positions, with parallaxes, proper motions, magnitudes, and colors. In this mission, Gaia Sky is the central tool for off-the-shelf visualization of these data, and for aiding production of outreach material. With its capabilities to effectively handle these data, to enable seamless navigation along the high dynamic range of distances, and at the same time to provide advanced visualization techniques including relativistic aberration and gravitational wave effects, currently no actively maintained cross-platform, modern, and open alternative exists.

Index Terms—Astronomy visualization, 3D Universe software, star catalog rendering, Gaia mission.

1 Introduction

These are exciting times for astronomy and astrometry research. The Gaia satellite, launched back in December 2013, has been measuring since July 2014 the star positions and proper motions of roughly one percent of the stellar content of our Milky Way, a spiral galaxy populated with 100–300 billion stars. It has two fields of view, separated by an angle of 106.5°, which are projected on the same focal plane by a complex system of mirrors. With 106 CCD sensors and about a billion pixels, it is the largest camera to ever be flown into space. DPAC, the Data Processing and Analysis Consortium, is the multinational endeavor with institutions from more than 20 countries responsible for data processing and construction of the final Gaia catalog. However, the release of this catalog is rolled out in an incremental fashion. The Gaia Data Release 1 (GDR1), published in September 2016, contains a catalog of over 1.1 billion 2D star positions. Additionally, more than two million parallaxes and proper motions (angular velocities of stars in the sky, as observed from Gaia) could be derived using cross-matching techniques with earlier catalogs, such as Tycho-2 [11] and Hipparcos [25], enabling this part of the star map to raise into the third dimension, as well as 250,000 radial velocities by cross-matching with RAVE [16] data. The second data release, GDR2, was released April 25, 2018. Based on more than 22 months of data collection, it represents the largest leap the Gaia catalog will ever make. It contains the positions, parallaxes, proper motions, magnitudes, and colors of more than 1.3 billion stars. Additionally, it offers more than 7 million radial velocities, half a million variable stars, and about 14 thousand asteroid orbits. It can be considered the first real Gaia dataset, and it is enabling astronomers to study the history, composition, and structure of our galaxy in great detail.

The appeal of such a mission goes well beyond astronomy, and in order to spark an interest in the laymen, clear and simple messages and catching media material are usually required. In this paper, we present a free and open-source software package created with the main objective of delivering the Gaia catalog to related research areas and the general public, and enabling anyone to gain insight into not only the mission and its catalog, but also supporting astrometric, astronomic, and cosmological research.

The contributions of this paper include:

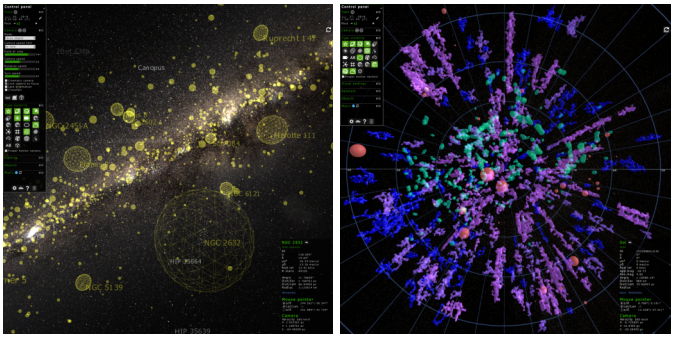
- A new magnitude–space level-of-detail octree structure,
- a floating camera approach to address single-precision floating-point number accuracy issues,
- integrated visualization of relativistic effects, and
- the representation of proper motions of stars.

2 Related Work

The works that are most closely related to Gaia Sky are comprised by open-source, freely available 3D universe software packages that run on laptops and desktop computers. In this category, we find Open Space (Section 2.1), Celestia (Section 2.2), and Space Engine (Section 2.3). Section 2.4 covers less closely related software. The compared systems do not have a strong academic record to compare to, so in all cases, our comparisons were conducted from a user's point of view. A detailed performance evaluation of these systems with respect to Gaia Sky is provided in Section 5.

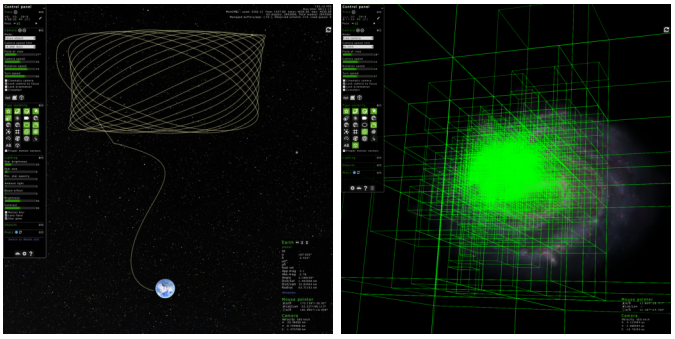
- *Antoni Sagristà is with Heidelberg University. E-mail: toni.sagrista@iwr.uni-heidelberg.de.*
- *Stefan Jordan is with Heidelberg University. E-mail: jordan@ari.uni-heidelberg.de.*
- *Thomas Müller is with Max Planck Institute for Astronomy. E-mail: tmueller@mpia.de.*
- *Filip Sadlo is with with Heidelberg University. E-mail: sadlo@uni-heidelberg.de.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx



(a) (b)

Fig. 2: (a) Objects from the MWSC (Milky Way Star Clusters) catalog indicated by spherical meshes. (b) Semi-transparent isosurfaces of dust (green), HII regions (red), and hot OB stars (blue-purple) [12].



(a) (b)

Fig. 3: (a) Gaia trajectory from Earth to its site of operation, the Lagrangian point L_2, and its Lissajous orbit. The scales are adapted such that both parts can be shown together. (b) Octree structure of Gaia Sky's 90% parallax relative error catalog.

2.1 Open Space

Open Space [4] is an open-source interactive data visualization framework designed to visualize the entire Universe. We have evaluated its latest available public version, 0.11.1, which is a beta release. The package is only available for Windows in executable form, even though it can be built for macOS and Linux. On the other hand, Gaia Sky is out of beta since version 1.0.0 and provides off-the-shelf builds for Linux (rpm, deb, aur, sh), macOS (dmg), and Windows (32 and 64-bit exe installers).

The default dataset is downloaded automatically at startup of Open Space and contains a lot of data of the Solar System and its space missions. It implements virtual texturing to provide high resolution views of planetary surfaces, and it also makes use of height maps. Even though Gaia Sky can accommodate this kind of data, by default only the planets, a selection of moons, some minor planets, 14,000 asteroids, and the Oort cloud are provided. Also, Gaia Sky does not implement terrain level-of-detail or use height maps, as its focus is on the exploration of the Gaia star catalog.

Open Space uses a scene graph for its internal model, like Gaia Sky. Open Space, however, makes use of the more complex Dynamic Scene Graph [2] approach to be able to represent a high dynamic range of distances. Gaia Sky, on the other hand, uses the floating camera approach (Section 3.2) to avoid floating-point precision issues.

2.2 Celestia

Celestia has been around for many years, even though its development was stopped from 2011 to 2017. The current build is version 1.6.1 on Windows and macOS, and 1.5.1 on Linux. It still uses Hipparcos as the main catalog (120 K stars), and it is mainly focused on the Solar System, including virtual texturing for handling on-demand high-resolution textures of planetary surfaces and SPICE kernels for Solar System objects. Gaia Sky, in contrast, can handle much larger catalogs, but does not implement virtual texturing. Celestia has built over the years a vibrant and active community, producing different data packs.

2.3 Space Engine

Space Engine is proprietary closed-source software and Windows-only. It borrows heavily from the gaming industry, making use of impressive graphics, particle effects, and procedurally generated environments and worlds. As a star catalog, it uses Hipparcos. It contains only 130 K real objects, counting stars and other object types. Gaia Sky's focus, on the other hand, is not so much on graphics but on the efficient representation of the star catalog.

2.4 Other Related Work

Other related work falls mainly in two categories. The first comprises proprietary 3D Universe software packages, which often run on planetaria, while the second category addresses Gaia-related scientific visualization packages and frameworks.

In the realm of 3D Universe software, Uniview [14], Digistar [7], and Mitaka [1] are the main players in planetaria software worldwide, and offer advanced, visually appealing simulations. StarStrider [8] offers a 3D star chart and includes relativistic effects.

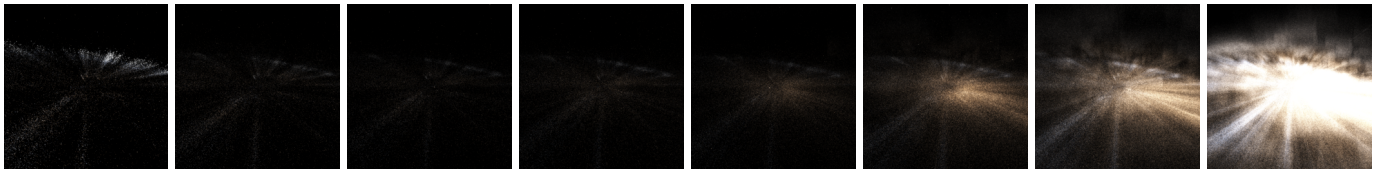
In the Gaia visualization domain, the Gaia Archive Visualization Service [21] offers a web-based interactive visual exploration tool, that features direct access to the Gaia catalog in a visual collaborative environment based on 2D and 3D linked views. Topcat [31] is a desktop tool for operations on astronomical catalogs and tables. It offers 2D and 3D plotting and is focused on astronomy use cases. Since version 4.2-1, it supports GBIN files, the native Gaia data format. Vaex [5], although not Gaia-specific, was also developed within DPAC, and is a visualization and exploration tool for big tabular data with its own volume rendering package. Gaia Sky focuses on a direct spatial representation of the catalog, but all these tools can be used in conjunction with Gaia Sky via SAMP [30], a messaging protocol that enables astronomy software tools to interoperate and communicate, to provide additional functionality such as filtering and clustering.

3 Gaia Sky

Our framework, named Gaia Sky [27] (Figure 1), is an open-source endeavor developed since 2014 in the framework of the Data Processing and Analysis Consortium of ESA's Gaia cornerstone astronomy mission. Gaia aims at creating a six-dimensional map of more than one billion stars with positions and velocities. In this context, the main mission of Gaia Sky is to deliver an off-the-shelf visualization of the Gaia catalog, and to aid in the production of outreach material. However, it has a wide range of other applications, from scientific to purely recreational. For instance, it was used within DPAC to help visualize and understand the stray light path, an optical issue with the spacecraft construction arisen in its first months of operation.

Gaia Sky contains a full simulation of our Solar System, with all its planets, dwarf planets, the main moons, and thousands of asteroids and asteroid orbits. In its current version, it allows for the exploration of Gaia DR2. We offer different datasets based on DR2 data, selected according to different parallax relative error criteria. They contain from 5 million stars to more than 600 million. Other included data are star clusters, extragalactic sources, and derived structures, like iso-density surfaces for hot star types, dust, and HII regions (see Figure 2). Beyond that, Gaia Sky also enables the visualization of Gaia's trajectory from Earth to its site of operation, $1.5 \cdot 10^6$ km behind the Earth away from the Sun, where it follows a Lissajous orbit around the Lagrangian point L_2 (see Figure 3a), and the octree structure, the Gaia data is stored in (Figure 3b).

The main strength of Gaia Sky, however, is that it can handle catalogs in the hundreds of millions of stars. The main challenges, that led to novel design solutions in Gaia Sky, include the efficient handling of these data to enable interactive exploration, and managing the very large range of scales with sufficient numerical precision. Gaia



(a) 0 / 100 K (b) 1 / 166 K (c) 2 / 167 K (d) 3 / 287 K (e) 4 / 521 K (f) 5 / 1.54 M (g) 6 / 4.48 M (h) 7 / 14.4 M

Fig. 4: View of all the stars in the first eight levels of the octree of the catalog with a parallax relative error up to 90% (601 million stars). Captions provide octree depth / number of stars in respective level.

Sky also features, besides a large number of exploration tools, a scripting interface, which can be accessed by directly running the scripts, or via a REST API, a variety of stereoscopic modes, a panorama (360°) mode, a planetarium output mode, and a VR spinoff. Additionally, a camera recorder utility enables the recording and later playback of the full camera state at a user-defined frame rate.

Before we provide a detailed description of the system (Section 4), we describe our data representation technique (Section 3.1), our approach to ensure sufficient numerical precision (Section 3.2), and, our most recent addition, relativistic effects (Section 3.3). Some performance hints are given in Section 5.

3.1 Data Representation and Access

One of the central challenges in the conception and development of Gaia Sky was the effective handling of the large star catalog. Compared to other star catalogs, GDR2 is large in several regards. First, it is large in the number of stars. GDR2 contains 1.33 billion stars with, however, varying quality of the line-of-sight distance (denoted parallax relative error). Here, we use a subset of 601 million stars for which the parallax relative error is smaller than 90.0%. This is orders of magnitude larger than previous 3D star catalogs. We offer a selection of DR2-based star catalogs on our website. The largest of them, which contains 1.33 billion stars and is still in beta, uses the geometrical Bayesian distance determination method by Bailer-Jones et al. [3]. Second, GDR2 is large in terms of spatial scales. Our Milky Way galaxy spans about 10^{21} m in its diameter, but on the other hand, the dimensions of the Gaia spacecraft are only a few meters. Third, it is large in the range of star brightnesses. In contrast to most previous catalogs, GDR2 includes astrometric data, spanning roughly 18 orders of magnitude in brightness [9]. These different high dynamic ranges are intertwined, and an effective navigation needs to take this information into account and exploit it. A tailored level-of-detail (LOD) structure with streaming capabilities is therefore comprising a central part of Gaia Sky's core system.

In many applications, LOD is accomplished by an octree, whose higher levels contain coarser approximations of the scene components, usually called impostors, while lower levels contain more detailed versions. During navigation, these octrees are traversed, and distance-based culling is employed to decide on the visibility of each of the octants of the octree. The idea underlying these approaches is to omit detail that the observer cannot perceive, i.e., to omit detail that would appear too small, because it is too far away compared to its size.

In Gaia Sky, however, we follow a different approach, because there are substantial differences between typical scenes and GDR2. First, stars are, by nature, discrete. That is, although they are huge objects, they are clearly confined and, given the incredible distances between them, they have to be considered points on galactic scales. Therefore, using smoothed and coarsened representations as impostors for sets of stars would not be an optimal choice, as the impostors would need to remain consistent from all viewing directions. Additionally, stars in Gaia Sky can move during the simulation due to their proper motions, and their shading parameters can be manipulated in real-time, making it even harder to use impostors. At the same time, the visibility of stars is not determined by their distance alone—it is the combination of their absolute brightness (which is defined for a fixed distance of 10 parsec) together with their distance to the observer (and possibly extinction) that is responsible if a star can be perceived or not. These considerations motivate our main contribution regarding star catalog representation in Gaia Sky: magnitude–space level-of-detail (MS-LOD).

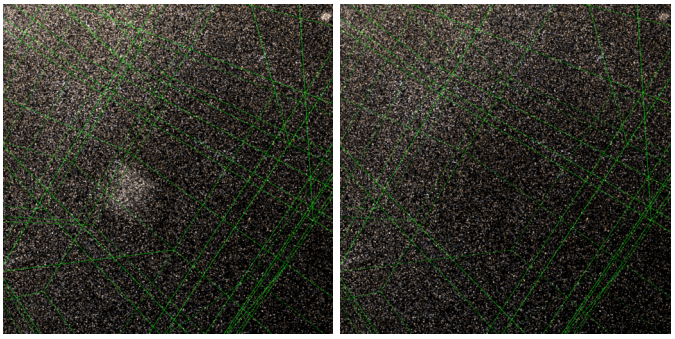
We exploit the correlation between distance, absolute star brightness, and apparent star brightness by constructing an octree that contains the stars sorted by absolute magnitude in a descending manner. That is, the root level contains the brightest stars, and lower levels progressively contain fainter and fainter stars. Additionally, the mapping of magnitudes to octree levels is bijective, so that each level contains a well-defined range of absolute magnitudes, and a given absolute magnitude can only be assigned to one level. Figure 4 shows an example for the stars in the first eight levels of the octree, from depth zero, the root (a), to level seven (h). Due to the non-uniform distribution of stars and the limited number of stars in each octant, some irregularities can be seen because the stars of a level are inspected in an isolated manner here. From (a) to (d), the number of stars in each level is nearly the same, but the brightness of the stars decreases, as does the overall brightness of the resulting image. From (f) to (h), the brightness of the stars still decreases, but the number of octants as well as the number of stars increases. That is in particular the case because there are many more faint stars in the Universe than bright ones. Hence, because of the additive blending, the overall image brightness strongly increases from (a) to (h).

Octree construction. For the construction of the MS-LOD octree structure, the whole catalog needs to be loaded into memory, where the stars are sorted by absolute magnitude from brightest to faintest. The starting point is the Cartesian axis-aligned bounding box that contains all the stars, and that serves as the root of the octree. The recursive subdivision of the octree is controlled by the parameter N_σ which is the maximum number of stars in an octant.

In practice, we start with level $n = 0$ and fill the root node with the N_σ brightest stars. If there are more stars than N_σ, we subdivide the root into its 8 octants. Then, in the next level, we take stars from the top of the (brightest) list and assign them to the corresponding level octant, determined by the position of the star, until an octant reaches N_σ stars. Next, we proceed one level deeper and start over. This procedure is carried out recursively until no star is left. Thus, a small N_σ leads to larger octrees with more octants, whereas a larger value leads to smaller, more compact octrees which are faster to process but have more stars in each node. Nevertheless, this has only an effect on performance and not on visual appearance. Note that this octree construction method provides a mapping between absolute magnitude and depth level, which adapts automatically to the absolute magnitude distribution of the input catalog. This partitioning in magnitude space between depths guarantees that brighter stars will always be visible from farther away than fainter stars.

Octree traversal. For rendering, the octree is traversed and the visibility of each octant is determined using a user-defined view angle Θ, which is compared to the solid angle θ_i of each octant i with respect to the position of the camera. Hence, Θ represents a draw distance measure by defining a threshold below which the octants are not visible, and above which they become visible. Octant i is visible if $\theta_i \geq \Theta$ and it intersects the camera's view frustum.

When an octant is visible, its stars are sent to the rendering system, and its children octants, if any, are processed the same way. If an octant is not visible, its stars are not rendered and its children octants are not processed. Without any further modifications, however, this would



(a) (b)

Fig. 5: (a) Direct octree traversal based on the view angle Θ alone yields pop-ins of octants. (b) LOD smoothing to fade-in the octants.

yield pop-ins of octants, as shown in Figure 5 and in the accompanying video. Therefore, we employ a LOD smoothing mechanism to fade-in octants (and its stars) into view. This is accomplished by a linear mapping of the solid angle θ_i to $[0,1]$, with $\theta_i = \Theta$ mapping to zero, and $\theta_i = \Theta + r$ mapping to one, where r is an offset to be used as a masking value for the star shader. Usually, the star shading (Sect. 4.4.4) automatically renders distant, faint stars invisible, but this entirely depends on Θ. Thus, large values for Θ make pop-ins more probable and the smoothing more useful.

3.2 Floating Camera

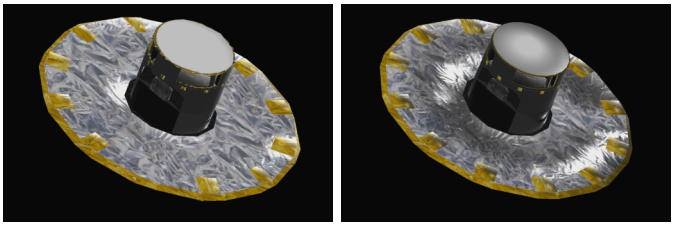
Gaia Sky contains a scene which represents vast distance ranges, from centimeters to gigaparsecs, far beyond what can be represented using 32-bit floating-point arithmetic. To avoid such issues, we follow a new approach which is somewhat complementary to previous ones like, e.g., the dynamic scene graph proposed by Axelsson et al. [2] or the logarithmic landmarks introduced by Li et al. [17].

The idea is to keep the camera at the origin of the global coordinate system at all times, and offsetting the whole scene graph so that the current camera is always located at position $(0\ 0\ 0)$. In practice, we add a new node at the top of the scene graph (Figure 10) with a translation by the inverse of the camera position vector, which is updated every frame. In detail, we split the standard vertex transformation pipeline from object space to the camera's clip space into two parts:

$$\mathbf{p}_{\text{clip}} = \mathbf{M}_{\text{proj}} \cdot \mathbf{M}_{\text{view}} \cdot \mathbf{M}_{\text{model}} \cdot \mathbf{p}_{\text{obj}} = \mathbf{M}_{\text{proj}} \cdot \mathbf{M}_{\text{mv}} \cdot \mathbf{p}_{\text{obj}}. \quad (1)$$

The crucial step is the transformation defined by the model-view matrix $\mathbf{M}_{\text{mv}}$ between object coordinates, $\mathbf{p}_{\text{obj}}$, and camera coordinates. For that, we use the arbitrary-precision floating-point operations from the `java.math` package on the CPU to handle every transformation within the scene graph. The resulting model-view matrix is then uploaded to the GPU, where we can still use single float precision shaders to convert from local camera coordinates to clip coordinates $\mathbf{p}_{\text{clip}}$, and finally determine screen coordinates.

Figure 6 shows the Gaia telescope model which is located in the Lagrangian point L_2, about 1.5 million kilometers behind the Earth away from the Sun, while the camera is only 275.04 m away from the telescope. Hence, the model matrix $\mathbf{M}_{\text{model}}$ consists of a translation vector for Gaia with a length of $1.5 \cdot 10^{11}$ m (distance Sun–Earth) plus $1.5 \cdot 10^{9}$ m, while the view matrix $\mathbf{M}_{\text{view}}$ consists of a translation vector of a similar length. Using global coordinates and standard 32-bit floating-point shaders, this yields jittering of the vertices due to the lack of floating point precision, and finally leads to incorrect per-pixel lighting (Figure 6a). Note that in Figure 6a, we even had to artificially reduce the distance to Gaia down to 1200 km because rendering broke down completely for larger distances. With our method (Figure 6b) based on the arbitrary-precision model-view matrix calculation on the CPU, there is no jittering problem because distances of a few centimeters compared to the relative distance between camera and the telescope can be handled by single float precision.



(a) (b)

Fig. 6: (a) Straightforward vertex transformation using 32-bit floating point precision leads to jitter. The resulting misaligned structures of the model lead to wrong per-pixel lighting. (b) Our floating camera approach does not exhibit these issues.

The floating camera approach provides a simple and effective way to handle a high dynamic range of distances without precision issues. It achieves this by just adding a translation to the transformation matrices. In comparison, the Dynamic Scene Graph requires the camera to be moved around in the scene graph, and its position to be taken into account when computing the transformations applied to all nodes.

3.3 Relativistic Effects

Gaia Sky implements a couple of relativistic effects, namely relativistic aberration and the apparent visual distortion due to gravitational waves. Both effects are implemented in an integrated way, which minimizes the general overhead to zero when the effects are turned off.

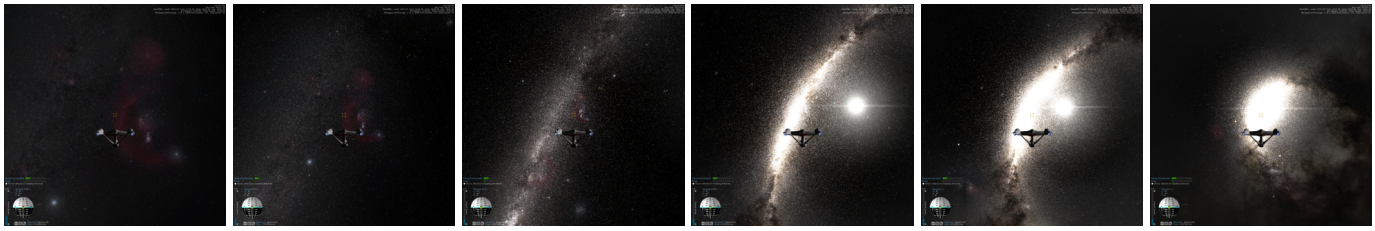
Relativistic Aberration. The relativistic aberration of light is a special-relativity effect that arises when an observer moves at a velocity v close to the speed of light c. In that case, the rays of light from each source that reach the observer are tilted towards the observer's direction of motion, which yields an apparent reduction of the effective angle ϑ between the velocity direction and the light source to the apparent angle ϑ' according to the aberration formula

$$\cos\vartheta' = \frac{\cos\vartheta + \beta}{1 + \beta\cos\vartheta}, \quad \beta = \frac{v}{c}. \quad (2)$$

This has a strong apparent minification effect in the direction of motion and a strong magnification effect in the opposite direction. As a result, it is even possible that objects, which are actually behind the observer, appear to be in front of her.

Figure 7 shows Gaia Sky in relativistic spacecraft mode. While the spacecraft keeps its position relative to the camera, the velocity is increased from zero to nearly the absolute speed limit c. The relativistic aberration is non-linear, which means that even for high velocities, the effect is rather small, and only for velocities near the speed of light, the tilting toward the direction of motion increases dramatically. In that case, even objects which are actually behind the spacecraft, like the Sun in Figure 7d, appear in front of the observer. The increasing brightness of the stars is due to the additive blending of the star shading. Although this does not show the correct behavior, it is quite similar to the searchlight effect, which causes an increasing brightness in the direction of motion and a strong decreasing brightness in the opposite direction. The searchlight effect, as well as the relativistic Doppler shift, are, however, not implemented yet. A detailed discussion of the relativistic effects is out of the scope of this work, and we refer the reader to, e.g., Müller et al. [22].

Gravitational Waves. Gravitational waves are a general relativistic effect, which produces a disturbance in the curvature of spacetime, generated by accelerated masses. These waves propagate outward from their source at the speed of light. Gaia Sky implements the visual effects that an observer would perceive in such an event, based on the model by Klioner [15]. There are a few parameters (4 amplitudes/polarization and the wave frequency) which can be adjusted according to the source. The current model used in Gaia Sky has a few caveats. First, it is only valid for slowly moving observers, with velocities much smaller than the speed of light. Second, the usual amplitude (strain) of



(a) 0 c (b) 0.5 c (c) 0.9 c (d) 0.99 c (e) 0.999 c (f) 0.9999 c

Fig. 7: Gaia Sky in relativistic spacecraft mode, with increasing velocity. Even with half the speed of light, $v = 0.5$ c, the relativistic aberration is rather weak. The closer the spacecraft approaches the absolute speed limit c, the stronger the aberration becomes.

(a) (b) (c)

Fig. 8: Apparent distortion due to a gravitational wave on the view of Mars and its moon Phobos.

the gravitational wave is very small, but can be artificially exaggerated for visualization purposes. And third, such sources of gravitational waves of the model—supermassive binary black holes in the centers of galaxies—are typically short-living (few thousands of years), so adjusting the parameters according to the simulation time might be in order. Figure 8 shows the influence of the apparent distortion due to a gravitational wave on the view of Mars and its moon Phobos.

4 System

This section provides a bird's eye view of the system parts, which are not novel, but nevertheless essential to the system as a whole. Gaia Sky is implemented using Java and OpenGL. The OpenGL bindings are provided by the Lightweight Java Game Library [20], and an additional framework layer, libGDX [18], is used for its caching, batching, and loading utilities. The choice of Java as a main platform is due to the abstraction provided by the JVM, which makes it easy to port to and maintain several platforms, one of the main requirements of the project. Furthermore, the language of choice of DPAC for the data processing of the Gaia data is Java, and there exists a vast code base with interdependent libraries and utilities, to compute, for example, the attitude quaternions of the satellite, which would need to be all translated to another language otherwise. Additionally, the default use of the just-in-time compiler, which gives the compiler access to runtime information not available to native languages, plus the avoidance of garbage collection as much as possible by using object pools and custom collections, helps minimizing the impact of using such a platform for a real-time graphics application where high frame rates are needed. Performance hot-spot functions, such as matrix operations, are implemented in plain C and called using the Java Native Interface.

4.1 Main Loop

At its highest level, the system implements a very simple update–render loop, in charge of updating the model objects and rendering them each frame. The loop also computes the amount of time elapsed since the last iteration, and passes it on to the update and render stages.

4.2 Structure

Gaia Sky is structured into modules, each with a defined set of responsibilities. Only five main modules compose the backbone of Gaia Sky, as seen in Figure 9: the user interface module, the event manager, the object model, the rendering system, and the scripting engine. Below,

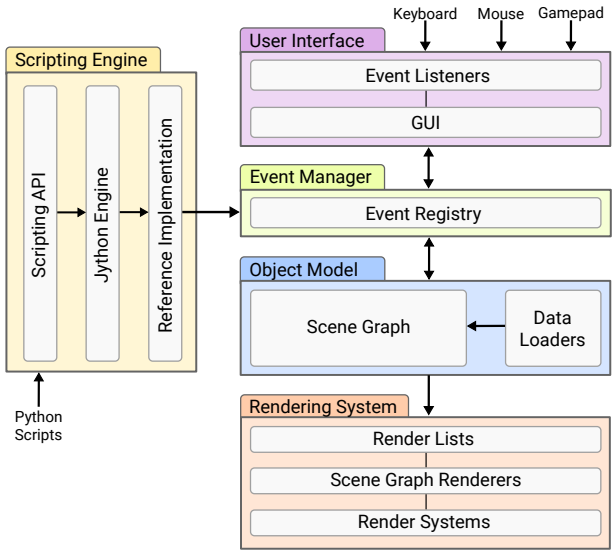


Fig. 9: Gaia Sky component structure. The interactions captured by the UI are processed and the relevant events are posted to the event manager, which broadcasts them to the relevant parties. The object model is updated continuously, and its entities are added to the render lists, which are used by the rendering system to display the scene.

we give a brief overview of the individual modules, and provide detailed descriptions in the subsequent sections.

User interface. This module is in charge of managing all interactions with the user. Among its responsibilities are generating and rendering the graphical user interfaces, as well as listening and processing the user's input events through the input listeners. The user interface is skinnable and supports HiDPI themes. The communication of the user interface module with other modules is accomplished by means of the event manager.

Event manager. This component is a generic registry, where event producers and consumers are connected via a centralized hub. Any entity can publish and subscribe to events of interest. The event manager reacts to events synchronously and sequentially. Actions can be specified to be processed after the current loop cycle has finished, ensuring thread safety and consistent state of the object model.

Object model. The object model contains and manages all the scene objects, and is in charge of updating their state during every iteration of the main loop.

Rendering system. This module is in charge of rendering the scene. Objects are routed from the object model to the rendering system using a number of render lists. These lists are used by the renderer objects, producing different kinds of visuals for each element.

Scripting engine. Finally, the scripting engine is a component that allows for the execution of Python scripts using an API to access and modify the internal state of Gaia Sky with ease and precision.

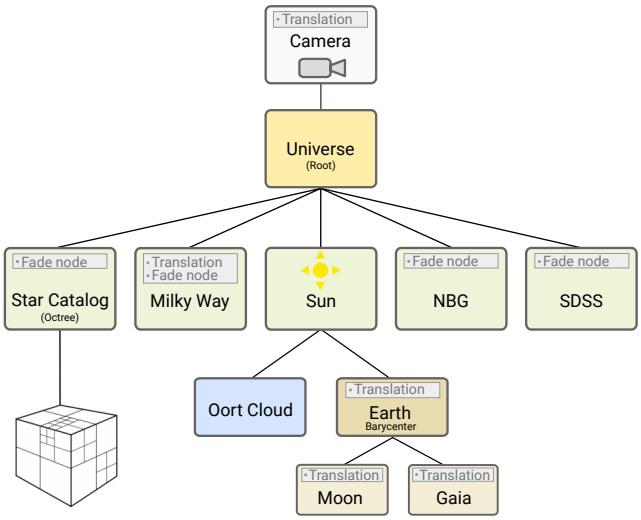


Fig. 10: Part of the scene graph used in Gaia Sky, with floating camera at the root. The global reference system is centered around the "Sun".

4.3 Object Model

The object model holds all the objects in the scene. These objects are organized into a scene graph (Figure 10), a tree-like data structure in which geometrical transformations are inherited from parent to children. All objects in the object model are categorized into component types, which organize the objects according to their typology. Available component types are stars, planets, moons, satellites, asteroids, clusters, labels, grids, orbits, atmospheres [24], constellations, constellation boundaries, galaxies, topological information, locations, arbitrary meshes, titles, and others. These component types are mainly used to implement the switch-on/off behavior, to prevent all data being displayed at once, and thus cluttering the viewport.

4.3.1 Data Loaders

Gaia Sky handles many different kinds of data, and their loading mechanisms can vary substantially. In a broad sense, the types can be organized into two large groups: *particle data* and *non-particle data*.

Particle data. Stars, galaxies, and essentially all data which are point-based, and come from astronomical catalogs, are particle data. They usually have a single node representing them in the scene graph. The different catalog formats are supported through the STIL Java library [31], including VOTable [23], FITS, ASCII, CSV, and GBIN. The STIL loader relies on Unified Content Descriptors (UCD) [6] defined by the International Virtual Observatory Association (IVOA) to assign units and semantics to each data column. Gaia Sky also uses an own binary format that is very compact and can be memory-mapped very easily for increased performance. As an example, Figure 11 shows the catalog descriptor file for the GDR2 particle dataset.

Non-particle data. Any non point-based data, such as planets, orbits, constellations, locations, satellites, grids, and others, have their own object in the data model, and their own node in the scene graph. These data are usually loaded from JSON files, using a specific set of loaders that match attributes by name via reflection. Each of the files defines a few one-to-many relationships which specify what files are to be loaded by which data loaders. The default loader (JSONLoader) is in charge of fetching the data in several JSON files, such as `meshes.json`, `planets.json`, or `locations.json`. Figure 12 shows, as an example, a snippet of `planets.json`, which defines the planet Earth.

4.3.2 Levels of Detail

Once the octree is constructed, Gaia Sky loads its structural metadata and traverses it at each frame, selecting the "visible" nodes and sending them to the relevant renderer.

```
"description" : "Gaia DR2 catalog (12.5% relative error). About 94 M stars",
"data" : [
   {
      "loader" : SunLoader,
      "files" : []
   },
   {
      "loader" : OctreeGroupLoader,
      "files" : [ "data/gdr2/particles/", "data/gdr2/metadata.bin" ]
   }]
```

Fig. 11: Example descriptor of the GDR2 catalog. It contains a Sun loader, which loads the Sun, and an octree loader, which contains the level-of-detail metadata file, and the folder where the actual catalog is.

```
   "name" : Earth,
   "wikiname" : Earth,
   "color" : [.13, .26, .89, 1.0],
   "size" : 6371.1,
   "ct" : Planets,
   "absmag" : -2.78, "appmag" : -3.1, "parent" : Sol, "impl" : Planet,

   "coordinates" : { "impl" : EarthVSOP87,
                     "orbitname" : Earth orbit },

   "rotation" : { "period" : 23.93447117,
                  "axialtilt" : -23.4392911,
                  "inclination" : 0.0,
                  "meridianangle" : 180.0 },

   "model" : {
      "type" : sphere,
      "params" : { "quality" : 180, "diameter" : 1.0, "flip" : false },
      "texture" : { "base" : earth-day*.jpg,
                    "specular" : earth-specular*.jpg,
                    "normal" : earth-normal*.jpg,
                    "night" : earth-night*.jpg }
   },

   "atmosphere" : {
      "size" : 6450.0,
      "wavelengths" : [0.650, 0.570, 0.475],
      "m_Kr" : 0.0025,
      "m_Km" : 0.001,
      "params" : { "quality" : 180, "diameter" : 2.0, "flip" : true }
   }
```

Fig. 12: JSON definition of the Earth object within the `planets.json` file. Images with an asterisk are given in different resolutions according to the quality defined in the configuration settings.

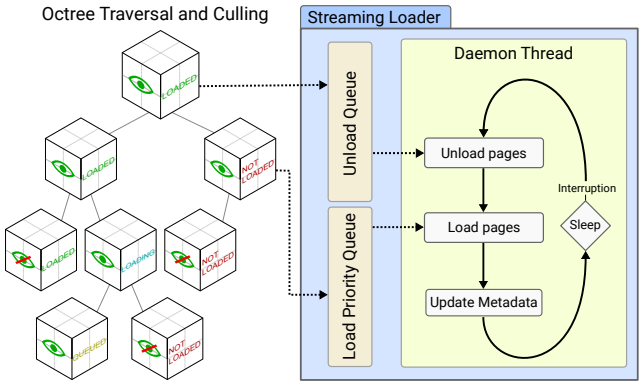


Fig. 13: Streaming catalog loader. Adding objects to the load queue causes the system to send an interruption to the daemon thread, which then unloads the objects in the unload queue and loads the objects in the load queue. Finally, metadata like constellations are updated.

If the catalog is large and does not fit into memory, as is the case of most GDR2-based catalogs, Gaia Sky implements a streaming loader solution, depicted in Figure 13. This loader relies on the octree data pages (octants) to be distributed in different files. The loader defines two queues, a load queue, which is a priority queue weighted by the inverse of the depth of the octant in the tree—lower-depth octants containing brighter stars are loaded first—and an unload queue, which

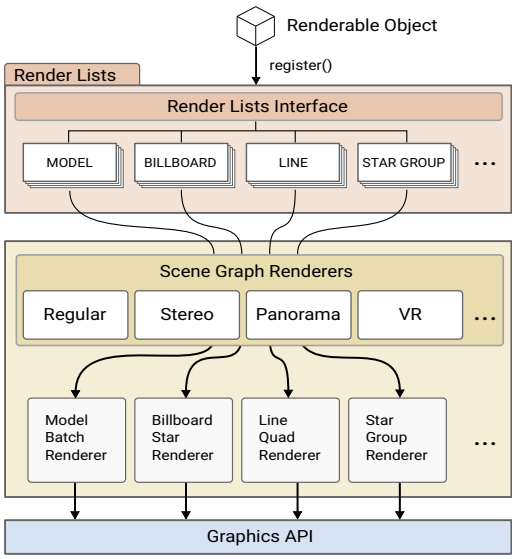


Fig. 14: Detailed view of the rendering system. The incoming render lists are routed through the scene graph renderers and to the actual renderer objects, which communicate directly with the graphics API.

contains all loaded octants sorted by access date—octants that have not been visited for the longest time are in the head. In this case, each time an octant is visited, the culling algorithm is run to determine its visibility. If it is visible, we query its state (*not_loaded*, *loaded*, *queued*, or *loading*). If the octant is not yet loaded, it is added to the load priority queue. If it is loaded, its state in the unload queue is updated (it is removed and added to the head). Depending on I/O bandwidth and latency of the file system, the streaming loader may cause pop-ins if the loading of a data page is slower than the camera. The actual loading and unloading of data is handled by a background daemon thread.

4.4 Rendering System

The rendering system, depicted in Figure 14, is in charge of rendering the scene to the active render targets. Possible render targets are the display, image files, or the VR API. The main rendering pipeline consists of four main stages executed in a sequential manner: the pre-passes, the scene rendering, and the post-processing stage. They are described in the following.

4.4.1 Pre-Pass Stage

The first stage is the pre-pass stage. In this stage, the scene is rendered to FBOs using different techniques, to be used in later stages.

Shadow map pass. The shadow map pass gathers the relevant model objects, positions the camera at a certain distance in the direction of the light source, and renders a depth map. This will later be used by the models to compute the shadows. Gaia Sky implements an adaptive shadow mapping solution instead of a global one because the depth map resolutions needed for such a sparse scene, where there are huge distances between the objects, which are themselves very small, are simply not feasible with current hardware.

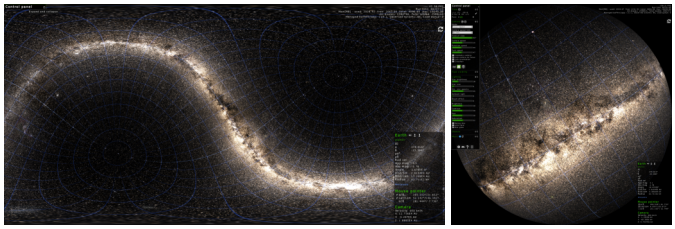
Occlusion pass. The occlusion pass renders close-up stars and models using a solid black color into a single low-resolution FBO. This is later used in the light glow post-processing effect, which *estimates* the number of visible fragments of each star and overlays a light glow effect, which spills over the rest of the geometry.

4.4.2 Scene Rendering Stage

The next stage is the actual scene rendering. After the update phase, the render lists are prepared to be used by the relevant scene graph renderer (SGR) object, once the pre-passes have been rendered. The responsibility of the SGR objects is to prepare the render environment (viewport, post-processing FBOs, etc.), call each render system with the relevant render list, and run the post-processing stage. Since the rendering and the post-processing may need to happen more than once—for instance, the stereoscopic mode renders the scene twice—these two stages are handled by the SGR objects.

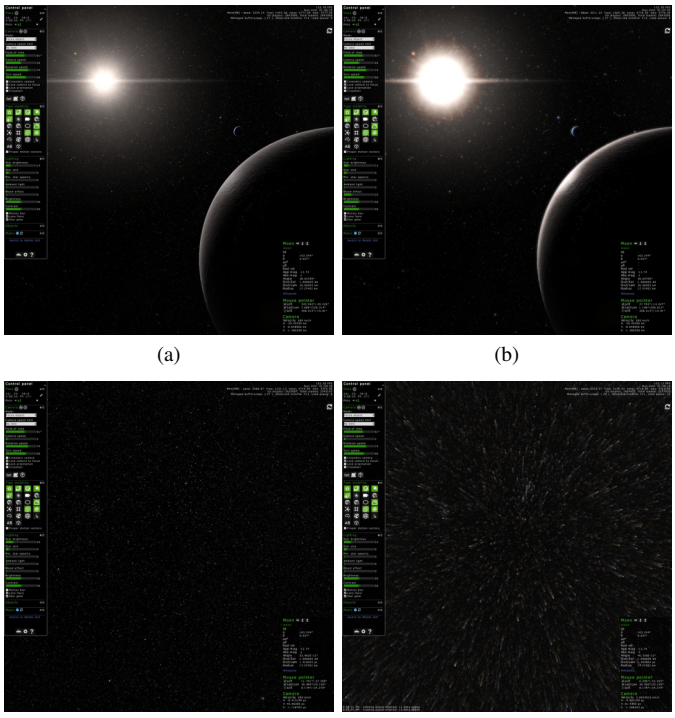
Gaia Sky has four SGR objects. The normal SGR, the 360° panorama SGR (Figure 15a), the Gaia FOV SGR, which projects both fields of view of Gaia on a single viewport, and the stereoscopic SGR. Additionally, the VR spinoff adds an OpenVR SGR.

The render systems are in charge of handling and rendering a render list each, corresponding to a render group. The render systems are constructed and managed automatically, and the routing of the correct render list to the relevant render system is also done in a manner transparent to the rest of the program. Each render system, then, corresponds to a render group.



(a) (b)

Fig. 15: The panorama scene graph renderer (a) generates images, which can be encoded into 360°-videos, whereas the output of the planetarium mode (b) is dedicated to fulldome planetarium shows.

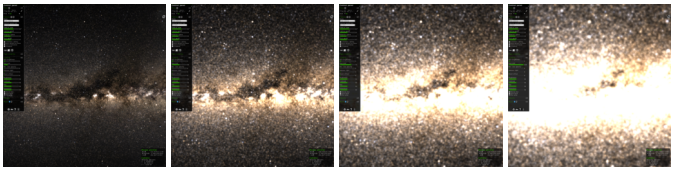


(a) (b) (c) (d)

Fig. 16: Bloom ((a) and (b)) and motion blur ((c) and (d)) are carried out in the post-processing pass.

4.4.3 Post-Processing Stage

The final stage, which is actually handled by the SGR objects, is the post-processing stage, where a series of filters is defined and runs on the result of the rendering stage. Filters can be easily linked and composited by the use of ping-pong buffers. The lens flare, anti-aliasing (FXAA [19] and NFAA [29]), light glow, motion blur (Figure 16d), and bloom effects (Figure 16b) are all run as post-processing filters.



(a) $\sigma = 0.25$ (b) $\sigma = 0.5$ (c) $\sigma = 0.75$ (d) $\sigma = 1.0$

Fig. 17: View toward the center of the Milky Way rendered with user-defined, increasing star pseudo-size σ. The larger σ the larger the stars appear, but due to the blending, the central region becomes overdense.

4.4.4 Shaders

Gaia Sky features a wide variety of shaders, which are in charge of producing the final picture. The models are rendered using per-pixel lighting. Lines can be rendered either as primitives or using polyline quadstrips. In order to render stars as realistically as possible, we determine their visual appearance and RGB color using their measured apparent magnitude and photometric intensities in the two Gaia photometric bands. To achieve a realistic and physically based star rendering, we first compute the absolute magnitude, which is the brightness of a star at a distance of 10 parsecs, using the apparent magnitude. Then, we correct it using the extinction value found in the catalog, if present, or we apply a simple analytic galactic dust model otherwise. Finally, we convert the absolute magnitude to a flux, from which we can compute a pseudo size. This pseudo size is used for rendering purposes only. The size is passed into the star shader and used in conjunction with the star brightness, the star size, and the star minimum opacity settings to determine its representation in pixels, see Figure 17. If the star is close enough, it is shaded using a textured billboard. The star colors are computed from the effective temperature value in the catalog, if present, or using the conversion from BP-RP (blue minus red photometric intensity values) to effective temperature as discussed by Jordi et al. [13]. Finally, we convert the effective temperature to RGB using an algorithm based on best fits by Helland [10].

4.4.5 Stars and Proper Motions Rendering

Star catalogs are represented by a single object in the octree. This means that the floating camera approach is not applied to every single star at each frame. Instead, stars are sent to the GPU as VBOs of float attributes. A configurable set of background threads continuously index and update the stars, so that close-by stars are always well indexed and quickly accessible. Accuracy problems would become apparent only when the camera gets close to a particular star. Thus, for the few stars which are close to the camera, Gaia Sky switches to a billboard-based rendering and applies the floating camera transformation, eliminating the need to transform all stars at every frame.

Proper motions are instantaneous angular velocities of stars in the sky. Tangential velocity vectors can be calculated when combining proper motions with parallaxes. Additionally, when radial velocities are also available, 3D velocity vectors can be obtained. Gaia Sky features support for both tangential velocities and full 3D vectors. The representation of proper motions in Gaia Sky is done via a straight-line integration in the vertex stage of the star shader, given the instantaneous proper motion vector and the simulation time. This enables real-time star movement when the time speed is set sufficiently high. Since the represented star velocities are only a first-order approximation to the real motion, we limit the explorable time range, allowing us to use a static octree structure.

4.4.6 Relativistic Effects Rendering

Both relativistic aberration and gravitational waves act at vertex level in Gaia Sky. The relativistic effects manager is in charge of updating the parameters and matrices needed for the relativistic effects. It is updated every frame with the camera and the current time. During the rendering stage, the shader programs are chosen depending on the active relativistic effects, the respective uniform values are set (velocity direction vector and speed for the relativistic aberration and transformation matrix, wave frequency, wave parameters, and time for the gravitational waves) and the draw call is issued normally. The shader programs include functions to apply the relativistic transformations to the positions of vertices according to the current state, passed via the uniform values. In the vertex shader, the uniform values are gathered and passed on to these functions, which compute a new position for each vertex. This is done for every vertex rendered by Gaia Sky that needs to be affected by the relativistic effects.

```
from gaia.cu9.ari.gaiaorbit.script import EventScriptingInterface

gs = EventScriptingInterface.instance()
gs.disableInput()
gs.cameraStop()

gs.setRotationCameraSpeed(20)
gs.setTurningCameraSpeed(20)
gs.setCameraSpeed(20)

gs.goToObject("Sol", 20.0, 4.5)
gs.setHeadlineMessage("This is the Sun, our star")

gs.sleep(4)

gs.clearAllMessages()
gs.goToObject("Earth", 20.0, 6.5)
gs.setHeadlineMessage("This is the Earth, our home")

gs.enableInput()
```

Fig. 18: Simple Gaia Sky script. First, the environment is prepared, and the values for certain camera properties are set. Then, we move the camera to the Sun and to the Earth, printing some messages on-screen. The API is fully documented in the official Gaia Sky documentation [28].

4.5 Time Control

There are two principal time frames in Gaia Sky. The real wall clock time t and the simulation time frame τ are linked using a warp speed factor, which applies to t and indicates how fast the simulation time passes with respect to the wall clock time. The warp speed factor is user-controlled and enables time-lapse effects. While t is used to update all the entities that need a direct connection to the real time, such as the integration of the forces acting on a spacecraft, τ is used to update all properties of entities that need a temporal reference frame such as the proper motions of stars or the planets' positions and orientations.

4.6 Scripting Engine

The scripting engine enables the execution of user scripts to manage and control the internal state of Gaia Sky. The script manager handles the high-level asynchronous execution of scripts and defines a maximum number of concurrent scripts by a cap in the thread pool size. Each new script runs in a separate thread and it is interpreted alongside the main thread of Gaia Sky by the interpreter. The scripting interface is of paramount importance for the production of outreach material and the creation of stories. Figure 18 shows an example for a camera motion between the Sun and the Earth with some text overlays.

4.7 Camera Recording and Playback

Gaia Sky offers a native method for recording and playing camera files in order to aid in the production of audiovisual material. Camera files contain a series of values defining the time and the camera state at each frame. Each row in a camera file contains the current τ, as well as the camera's position, direction, and up vector. Camera files can be played back directly from the UI or using the scripting API.

4.8 Camera Modes

There are four main camera modes in Gaia Sky, which affect camera behavior and capabilities in various ways.

The *focus mode* makes the camera automatically track a focus object. The camera normally points at the object, even though the view direction can be changed, remaining relative to the focus' position. It

is possible to lock the position to the focus, so that the relative position of the camera with respect to the focus is maintained, and also its orientation so that the camera rotates with the object. The *free mode* allows for free roaming in the scene. For maximum movement freedom, a game controller with two analog joysticks can be used. The velocity scaling depends on the distance to either the closest model object, or to the origin of the reference system. The *Gaia FOV mode* projects both fields of view of Gaia on the screen and overlays the CCD configuration in focal plane. This allows the user to see what the current simulation of the Gaia satellite sees. The *spacecraft mode*, shown in Figure 7, puts the user in command of a spacecraft with a realistic engine model with thrust, a power multiplier, and an optional drag. The mode brings up the spacecraft GUI, which contains visual elements to control the spacecraft parameters, as well as an attitude indicator widget with the positions of the current velocity and anti-velocity vectors. Finally, there are two camera behaviors available. The *cinematic camera* behavior implements camera movement using the steering behavior principles laid out by Raynolds [26], which yields very smooth camera movements suitable for creating videos and camera paths. The *non-cinematic camera* behavior is a much more responsive implementation, more suited for interactive use.

4.9 Video Modes

There are five video output modes implemented in Gaia Sky, each mode corresponding to a different scene graph renderer, as discussed in Section 4.4.

The *normal* mode (the default) produces the output to screen as a single scene. The *planetarium* mode, shown in Figure 15b, is coupled with the frame output system to produce image sequences in azimuthal equidistant projection which can be encoded into fulldome videos. The *stereoscopic* mode renders the scene twice, where either parallel view, cross-eye, anaglyphic red-cyan, 3DTV, or VR profile can be selected. The eye separation is a function of the distance to the focus object or can be set to a fixed value. The *VR* mode is implemented in the VR spinoff branch of Gaia Sky, and makes use of the OpenVR API, which is responsible for the necessary image transformations for the supported headsets. Finally, in the *panorama* mode (Figure 15a), the whole scene around the camera is pre-rendered six times with $90^\circ \times 90^\circ$ field-of-view into a cubemap to create 360°-videos.

4.10 SAMP Integration

Gaia Sky also provides a basic integration with the Simple Application Messaging Protocol (SAMP) [30], which enables interoperability of astronomical software by sharing data and providing linked views. On startup, Gaia Sky looks for a preexisting SAMP hub. If found, a connection is attempted. If not found, Gaia Sky attempts further lookups at regular intervals of ten seconds. The only format supported by Gaia Sky through SAMP is VOTable, through the STIL data loader. The implemented features are *VOTable loading*, *row highlighting*, *selection broadcast*, and *point at sky coordinate*. Since Gaia Sky only represents 3D data, some assumptions are made upon receiving tables from other programs via SAMP. If Gaia Sky does not find enough information in the table metadata to reconstruct positions, the table is discarded. SAMP enables complex use cases like loading a table into Topcat from VO, creating histograms and plots, and sending the table to Gaia Sky for closer inspection in a galactic context (even VR), while providing inter-application linked views.

5 Performance

Gaia Sky targets common everyday systems such as laptops and desktop computers. The minimum system requirements are rather forgiving, due to the possibility to set low settings for graphics quality. A system with an Intel Core i3 CPU, Intel HD 4000 or NVIDIA GeForce 8400 GS with 1 GB of VRAM, 4 GB RAM, and at least 1 GB of free disc space should suffice. The frame rate of Gaia Sky depends largely on the used graphics card, the graphics settings, and the type of objects that are enabled. For instance, certain graphics quality settings will fetch higher resolution textures and use higher sample counts for some effects, which will lead to lower frame rates on most systems. In the LOD approach, the draw distance has obviously a huge impact on the frame rate, as it determines the amount of data that needs to be fetched in the background, as well as the number of stars on screen. In general, the application runs at very high frame rates on gaming-grade GPUs such as NVIDIA's GTX lines. The GTX 1070, for example, is able to pull between 150 and 200 fps with 5 million stars on screen. Low-power integrated Intel GPUs, such as the UHD 620, common in ultrabooks, run at between 30 and 60 fps, depending on what elements are enabled and/or visible. It is, however, difficult to assess the performance in a general manner, due to the sheer number of options given to the user in terms of visuals and quality settings.

The different SGRs have an obvious impact on the performance. Every loop cycle, the object model is updated only once, but depending on the SGR in use, it may be rendered more than once. The stereoscopic SGR renders the scene twice with different cameras, as does the VR SGR. The panorama SGR renders the scene six times and then processes the equirectangular projection.

The GDR2 raw dataset contains around 1.7 billion sources and weights 700 GB. In the preparation step, which is done only once, the octree is generated. Using the 90.0% parallax relative error, yielding a catalog of roughly 600 million stars, and a subdivision threshold $N_\sigma = 100,000$, a system with 2 TB of RAM (only 500 GB of memory were allocated to the generation process) takes about six hours, 2.8% of which are spent reading the catalog into memory, 2.7% are spent actually processing and generating the octree, and the rest for writing the results. In the case of the geometric Bayesian distances catalog, the resulting dataset contains 1.33 billion stars. The generation of this octree took about 14.5 hours, of which 5.8 hours were spent loading the data, 7.7 hours generating the octree, and the rest writing the results.

5.1 Performance Comparison

We have evaluated the performance of other similar systems yielding the following results. All evaluations were carried out using the same system, with an i7-7700, an NVIDIA GTX 1070, and 16 GB of RAM.

Open Space. The default star catalog in Open Space contains hundreds of thousands of stars. The frame rate we experienced was around 30–40 fps when inside the Solar System, and around 150 fps when navigating the stars. The frame rate in Open Space is inversely dependent on the speed of time, reaching 1 fps when time speed is 200 days/second. Gaia Sky can handle many more stars at higher frame rates, and a dependency on the time scale does not exist.

Celestia. Using the default settings, Celestia runs at a stable 60 fps. This includes, however, a limiting magnitude of 7.0 mag, which contains only 14 K stars. When the limiting magnitude is increased to the maximum setting (only 12.0 mag), the frame rate decreased to 40–50 fps. This may pose problems regarding scalability to larger star catalogs. Gaia Sky is able and ready to handle catalogs orders of magnitudes larger, without frame rate issues.

Space Engine. The performance of Space Engine is difficult to compare to that of Gaia Sky due to the heavy usage of graphical effects in the former. Most of the time, Space Engine runs at 60 fps. However, Gaia Sky can handle many more objects at high frame rates.

6 Conclusion

We presented a system for visualizing the star catalog acquired by the Gaia mission, with a special focus on its Data Release 2 (GDR2), comprising 1.3 billion stars. Due to its large size, this is the first star catalog that requires effective handling of such data. The fact that GDR2 includes brightness information and, in particular, high precision three-dimensional positional data, motivated our approach of magnitude–space level-of-detail, a data access mechanism specially tailored for the requirements of interactive stellar visualization. To handle the large range of scales numerically, we presented a respective CPU/GPU approach for camera computations, partially based on arbitrary-precision floating point calculation. We also presented our recent additions to the system, including relativistic aberration, and simulation of the visual effects of gravitational waves.

REFERENCES

[1] 4D2U-Project-Team. Mitaka. http://4d2u.nao.ac.jp/html/program/mitaka. [Online; accessed 31-March-2018].

[2] E. Axelsson, J. Costa, C. Silva, C. Emmart, A. Bock, and A. Ynnerman. Dynamic scene graph: Enabling scaling, positioning, and navigation in the Universe. *Computer Graphics Forum*, 36(3):459–468, 2017.

[3] C. A. L. Bailer-Jones, J. Rybizki, M. Fouesneau, G. Mantelet, and R. Andrae. Estimating distances from parallaxes IV: Distances to 1.33 billion stars in Gaia Data Release 2. *arXiv:1804.10121*, Apr. 2018.

[4] A. Bock, E. Axelsson, K. Bladin, J. Costa, G. Payne, M. Territo, J. Kilby, E. Myers, M. Kuznetsova, C. Emmart, and A. Ynnerman. OpenSpace: An open-source astrovisualization framework. *The Journal of Open Source Software*, 2(15):281, 2017. doi: 10.21105/joss.00281

[5] M. A. Breddels and J. Veljanoski. Vaex: Big data exploration in the era of Gaia. *arXiv:1801.02638*, 2018.

[6] S. Derriere, N. Gray, J. C. McDowell, R. Mann, F. Ochsenbein, P. Osuna, A. Preite Martinez, G. Rixon, and R. Williams. UCD in the IVOA context. 314:315, 2004.

[7] Evans and Sutherland. Digistar. https://www.es.com/Digistar. [Online; accessed 31-March-2018].

[8] FMJ-Software. StarStrider. http://www.starstrider.com. [Online; accessed 31-March-2018].

[9] Gaia-Data-Team. A description of the anticipated contents of Gaia DR2. https://www.cosmos.esa.int/web/gaia/dr2. [Online; accessed 31-March-2018].

[10] T. Helland. Teff to RGB conversion. http://www.tannerhelland.com/4435/convert-temperature-rgb-algorithm-code/. [Online; accessed 31-March-2018].

[11] E. Høg, C. Fabricius, V. V. Makarov, S. Urban, T. Corbin, G. Wycoff, U. Bastian, P. Schwekendiek, and A. Wicenec. The Tycho-2 catalogue of the 2.5 million brightest stars. *Astronomy and Astrophysics*, 355:L27–L30, 2000.

[12] K. Jardine. Making a galaxy map. http://galaxymap.org/drupal/node/256. [Online; accessed 26-June-2018].

[13] C. Jordi, M. Gebran, J. M. Carrasco, J. de Bruijne, H. Voss, C. Fabricius, J. Knude, A. Vallenari, R. Kohley, and A. Mora. Gaia broad band photometry. *Astronomy and Astrophysics*, 523:A48, 2010.

[14] S. Klashed, P. Hemingsson, C. Emmart, M. Cooper, and A. Ynnerman. Uniview - visualizing the universe. In *Proceedings of Eurographics 2010 - Areas Papers*, 2010.

[15] S. A. Klioner. Gaia-like astrometry and gravitational waves. *Classical and Quantum Gravity*, 35(4):045005, 2018.

[16] A. Kunder, G. Kordopatis, M. Steinmetz, T. Zwitter, P. J. McMillan, L. Casagrande, H. Enke, J. Wojno, M. Valentini, C. Chiappini, G. Matijevi, A. Siviero, P. de Laverny, A. Recio-Blanco, A. Bijaoui, R. F. G. Wyse, J. Binney, E. K. Grebel, A. Helmi, P. Jofre, T. Antoja, G. Gilmore, A. Siebert, B. Famaey, O. Bienaym, B. K. Gibson, K. C. Freeman, J. F. Navarro, U. Munari, G. Seabroke, B. Anguiano, M. Žerjal, I. Minchev, W. Reid, J. Bland-Hawthorn, J. Kos, S. Sharma, F. Watson, Q. A. Parker, R.-D. Scholz, D. Burton, P. Cass, M. Hartley, K. Fiegert, M. Stupar, A. Ritter, K. Hawkins, O. Gerhard, W. J. Chaplin, G. R. Davies, Y. P. Elsworth, M. N. Lund, A. Miglio, and B. Mosser. The radial velocity experiment (RAVE): Fifth data release. *The Astronomical Journal*, 153(2):75, 2017.

[17] Y. Li, C. w. Fu, and A. Hanson. Scalable WIM: Effective exploration in large-scale astrophysical environments. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1005–1012, 2006.

[18] LibGDX-Development-Team. libGDX. https://libgdx.badlogicgames.com. [Online; accessed 31-March-2018].

[19] T. Lottes. FXAA white paper. http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf, 2009.

[20] LWJGL-Development-Team. LWJGL. https://www.lwjgl.org. [Online; accessed 31-March-2018].

[21] A. Moitinho, A. Krone-Martins, H. Savietto, M. Barros, C. Barata, A. Falco, T. Fernandes, J. Alves, A. F. Silva, M. Gomes, J. Bakker, A. G. A. Brown, J. Gonzlez, G. Gracia-Abril, R. Gutierrez-Sanchez, J. Hernandez, S. Jordan, X. Luri, B. Mern, A. Vallenari, and A. Sagristà. Gaia data release 1: The archive visualisation service. 2017.

[22] T. Müller, A. King, and D. Adis. A trip to the end of the Universe and the twin paradox. *American Journal of Physics*, 76(4):360–373, 2008.

[23] F. Ochsenbein, R. Williams, C. Davenhall, M. Demleitner, D. Durand, P. Fernique, D. Giaretta, R. Hanish, T. McGlynn, A. Szalay, M. Taylor, and A. Wicenec. VOTable format definition. http://www.ivoa.net/documents/VOTable/, 2013.

[24] S. O'Neil. *Accurate Atmospheric Scattering, GPU Gems 2*. GPU Gems. Pearson Addison Wesley Prof, 2005.

[25] M. A. C. Perryman, L. Lindegren, J. Kovalevsky, E. Hoeg, U. Bastian, P. L. Bernacca, M. Crézé, F. Donati, M. Grenon, M. Grewing, F. van Leeuwen, H. van der Marel, F. Mignard, C. A. Murray, R. S. Le Poole, H. Schrijver, C. Turon, F. Arenou, M. Froeschlé, and C. S. Petersen. The HIPPARCOS Catalogue. *Astronomy and Astrophysics*, 323:L49–L52, 1997.

[26] C. Raynolds. Steering behaviors for autonomous characters. In *Proceedings of Game Developers Conference*, pp. 763–782, 1999.

[27] A. Sagristà. Gaia Sky. http://bit.ly/2FulIs0. [Online; accessed 31-March-2018].

[28] A. Sagristà. Gaia Sky official online documentation. https://gaia-sky.readthedocs.io. [Online; accessed 31-March-2018].

[29] Styves. NFAA – a post-process anti-aliasing filter. http://bit.ly/2FNvozp. [Online; accessed 31-March-2018].

[30] M. Taylor, T. Boch, M. Fitzpatrick, A. Allan, L. Paioro, J. Taylor, and D. Tody. IVOA recommendation: SAMP - Simple application messaging protocol version 1.3. 2011.

[31] M. B. Taylor. TOPCAT & STIL: Starlink table/VOTable processing software. In *Proceedings of Astronomical Data Analysis Software and Systems XIV*, vol. 347 of *Astronomical Society of the Pacific Conference Series*, 2005.