1 Contents 3
  1.1 Installation ........................................................................................................ 3
    1.1.1 System requirements .................................................................................. 3
    1.1.2 Download ..................................................................................................... 3
    1.1.3 Installation and uninstallation ..................................................................... 3
  1.2 System Folders .................................................................................................... 5
    1.2.1 Linux ............................................................................................................ 5
    1.2.2 Windows and macOS ................................................................................. 5
  1.3 Run instructions ................................................................................................ 5
    1.3.1 Running regular installation ....................................................................... 5
    1.3.2 Running from tar.gz package ..................................................................... 6
    1.3.3 Running from source .................................................................................. 6
    1.3.4 CLI arguments .............................................................................................. 7
    1.3.5 Packaging the software .............................................................................. 8
  1.4 Usage, data and options ..................................................................................... 8
    1.4.1 Settings and configuration ......................................................................... 8
    1.4.2 The configuration file ................................................................................ 12
    1.4.3 Graphics configuration .............................................................................. 12
    1.4.4 User Interface ............................................................................................... 14
    1.4.5 Controls ........................................................................................................ 18
    1.4.6 Camera modes ............................................................................................. 24
    1.4.7 Cinematic camera ........................................................................................ 26
    1.4.8 Stereoscopic (3D) mode ............................................................................. 27
    1.4.9 Planetarium mode ....................................................................................... 30
    1.4.10 Panorama mode .......................................................................................... 32
    1.4.11 Datasets .................................................................................................... 34
    1.4.12 External views ........................................................................................... 37
    1.4.13 Connecting Gaia Sky instances ................................................................. 37
    1.4.14 Camrecorder ............................................................................................... 39
    1.4.15 Scripting ..................................................................................................... 42
    1.4.16 Capturing videos ....................................................................................... 47
    1.4.17 Taking screenshots .................................................................................... 48
    1.4.18 SAMP integration ...................................................................................... 48
  1.5 Additional information ....................................................................................... 51
    1.5.1 Performance ................................................................................................ 51
    1.5.2 Graphics performance ................................................................................ 52
These are the official documentation pages of Gaia Sky. Find below the contents table to navigate around.

- Visit our [home page](#)
- Download [Gaia Sky](#)
- Submit a [bug or a feature request](#)
- Follow development news at [@GaiaSky_Dev](#)

Gaia Sky is described in the paper [Gaia Sky: Navigating the Gaia Catalog](#).
1.1 Installation

In the sections below is the information on the minimum hardware requirements and on how to install the software.

1.1.1 System requirements

Here are the minimum requirements to run this software:

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Linux / Windows 7+ / macOS, 64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Core i5 3rd Generation or similar. 4 core or higher recommended</td>
</tr>
<tr>
<td>GPU</td>
<td>Support for OpenGL 3.2 (4.x recommended), 1 GB VRAM</td>
</tr>
<tr>
<td>Memory</td>
<td>4+ GB RAM</td>
</tr>
<tr>
<td>Hard drive</td>
<td>1+ GB of free disk space (depends on datasets)</td>
</tr>
</tbody>
</table>

1.1.2 Download

Gaia Sky packages are available for Linux, macOS and Windows. You can either download the Gaia Sky build for your operating system (recommended) or browse and build the source code.

- Gaia Sky downloads page

1.1.3 Installation and uninstallation

Depending on your system and your personal preferences the installation procedure may vary. Below is a description of the various installation methods available.
Linux

We provide 3 packages for Linux systems (deb for Debian, Ubuntu and derivatives, rpm for RedHat, Fedora and derivatives and a linux installer which works on all distros) plus an AUR package for Arch and derivatives.

**deb**

This is the package for Debian-based distros (Debian, Ubuntu, Mint, SteamOS, etc.). Download the gaiasky_<version>.deb file and run the following command. You will need root privileges to install a deb package in your system.

```
$ dpkg -i gaiasky_<version>.deb
```

This will install the application in the /opt/gaiasky/ folder and it will create the necessary shortcuts and .desktop files. The package depends on the default-jre package, which will be installed if it is not yet there.

In order to **uninstall**, just type:

```
$ apt-get remove gaiasky
```

**rpm**

This is the package for RPM-based distributions (Red Hat, Fedora, Mandriva, SUSE, CentOS, etc.) Download the gaiasky_linux_<version>.rpm file and run the following command. You will need root privileges to install a rpm package in your system.

```
$ yum install gaiasky_linux_<version>.rpm
```

This will install the application in the /opt/gaiasky/ folder and it will create the necessary shortcuts.

In order to **uninstall**, just type:

```
$ yum remove gaiasky-x86
```

**Install from AUR**

If you have Arch, Manjaro, Antergos or any other Arch Linux derivative, you can install the package from AUR using any tool able to install AUR software. For example:

```
$ yay -S gaiasky
```

This will download the package and install it in the /opt/gaiasky/ folder. It also creates the necessary shortcuts.

**Unix/Linux installer**

We also provide a Unix/Linux installer that will trigger a graphical interface where you can choose the installation location and some other settings. Download the file gaiasky_unix_<version>.sh to your disk. Then run the following to start the installation.

```
$ ./gaiasky_unix_[version].sh
```

Follow the on-screen instructions to proceed with the installation.

In order to **uninstall**, just execute the uninstall file in the installation folder.

**Windows**

We offer a Windows installer for 64-bit systems. gaiasky_windows-x64_<version>.exe.
To install the Gaia Sky, just double click on the installer and then follow the on-screen instructions. You will need to choose the directory where the application is to be installed.

In order to **uninstall** the application you can use the Windows Control Panel or you can use the provided uninstaller in the Gaia Sky folder.

### macOS

For macOS we provide a `gaiasky_macos_<version>.dmg` file. To install, double-click on it to mount it and then click on the icon in the window that pops up. Follow the on-screen instructions, which will guide you through the installation process. Once finished, it is safe to unmount the dmg volume.

### Compressed (TGZ) package

A `gaiasky-[version].tgz` package file is also provided [here](#). It will work in all systems but you need to unpack it yourself and create the desired shortcuts. In **Windows**, use an archiver software (7zip, iZArc, etc.) to unpack it.

In Linux and macOS, you can use:

```bash
$ tar xzvf gaiasky-<version>.tgz -C [target_folder]
```

### 1.2 System Folders

In this documentation we will repeatedly refer to two different folders that Gaia Sky uses to store data and configuration settings: `$GS_DATA` and `$GS_CONFIG`. Their location depends on the Operating System and is specified below.

#### 1.2.1 Linux

As of version **2.2.0**, the Linux release of Gaia Sky uses the XDG base directory specification.

- `$GS_DATA` = `~/.local/share/gaiasky/`
- `$GS_CONFIG` = `~/.config/gaiasky/`

#### 1.2.2 Windows and macOS

Windows and macOS use the old `.gaiasky` folder in the user home directory for both locations, so:

- `$GS_DATA` = `$GS_CONFIG` = `[User.Home]/.gaiasky/`

### 1.3 Run instructions

This section contains information on how to run the software depending on the install method.

#### 1.3.1 Running regular installation

If you installed the software using an installer or a package manager system (`rpm`, `deb`), you just need to use the standard running procedures of your Operating System to run the application.
**Linux**

Just type `gaiasky` in a terminal or use your favourite desktop environment search engine to run the Gaia Sky launcher.

**Windows**

In windows, this means clicking on **Start** and then browsing the start menu folder **Gaia Sky**. You can run the executable from there.

**macOS**

Locate the launcher in your install directory (usually `/Applications`) and double click on it.

**Code and package**

However, if you are a maverick and do not like installers, you can also run the Gaia Sky directly from the source code in **GitHub** or using the **tgz** package.

### 1.3.2 Running from tar.gz package

You can also run the software using the **tar.gz** package without need for an installation. To do so, extract the archive to your desired location and run the `gaiasky` script (Linux, macOS), or the `gaiasky.cmd` batch file (Windows).

```
$ tar -xzvf gaiasky-[version].tar.gz
$ cd gaiasky-[version]
$ gaiasky #Linux/macOS
$ .\gaiasky.cmd #Windows
```

### 1.3.3 Running from source

**Requirements**

If you want to compile the source code, you will need the following:

- JDK11+
- git

Please, be aware that only tags are guaranteed to work (here). The master branch holds the development version and the configuration files are possibly messed up and not ready to work out-of-the-box. So remember to use a tag version if you want to run it right away from source.

First, clone the repository:

```
$ git clone https://gitlab.com/langurmonkey/gaiasky.git
```

**Getting the catalog data**

**Hint:** As of version 2.1.0, Gaia Sky provides a self-contained download manager to get all the data packs available.
The base-data pack is necessary for Gaia Sky to run, and contains the Solar System, the Milky Way model, etc. Catalog files are optional but recommended if you want to see any stars at all. You can bring up the download manager at any time by clicking on the button Data download in the preferences window, data tab. More information on the download manager can be found in Data download manager.

You can also download the data packs manually here.

### Compiling and running

To compile the code and run Gaia Sky run the following.

```bash
$ gradlew core:run
```

**Tip:** Gaia Sky will check that you are using a version compatible with Java 11 when running the build. You can still use a newer JDK version (e.g. JDK 13) by setting the following environment variable to `false` in the context of gradle:

```bash
$ export GS_JAVA_VERSION_CHECK=false
```

In order to pull the latest changes from the GitHub repository:

```bash
$ git pull
```

#### 1.3.4 CLI arguments

Gaia Sky offers a few command line arguments. Run `gaiasky -h` for more information.

```bash
$ gaiasky -h
```

Usage: gaiasky [options]

Options:
- `h, --help`
  Show program options and usage information.
- `v, --version`
  List Gaia Sky version and relevant information.
  Default: false
- `d, --ds-download`
  Display the data download dialog at startup. If no data is found, the download dialog is shown automatically.
  Default: false
- `c, --cat-chooser`
  Display the catalog chooser dialog at startup. This enables the selection of different available catalogs when Gaia Sky starts.
  Default: false
- `p, --properties`
  Specify the location of the properties file.
- `a, --assets`
  Specify the location of the assets folder. If not present, the default assets location (in the installation folder) is used.
- `vr, --openvr`
  Launch in Virtual Reality mode. Gaia Sky will attempt creating a VR context through OpenVR.
  Default: false
- `n, --noscript`
  Do not start the scripting server. Useful to run more than one Gaia Sky instance at once in the same machine.

(continues on next page)
1.3.5 Packaging the software

Gaia Sky can be exported to a folder to be run as a standalone app with the following command.

$ gradlew core:dist

That will create a new folder called releases/gaiasky-[version].[revision] with the exported application. Run scripts are provided with the name gaiasky (Linux, macOS) and gaiasky.cmd (Windows).

Also, to export Gaia Sky into a tar.gz archive file, run the following.

$ gradlew core:createTar

In order to produce the desktop installers for the various systems you need a licensed version of Install4j. Then, you need to run:

$ gradlew core:pack

These command will produce the different OS packages (.exe, .dmg, .deb, .rpm, etc.) of Gaia Sky into releases/packages-[version].[revision] folder.

1.4 Usage, data and options

1.4.1 Settings and configuration

Gaia Sky can be configured using the on-screen control panel and preferences window. Bring up the preferences window by clicking on the preferences icon in the Controls pane. However, very few features are not represented in the GUI, so you may need to dive deep into the properties file.

Graphics settings

Please refer to the Graphics configuration chapter.

Interface settings

The Interface settings section allows the user to set the language and the theme of the user interface.

- **Language** – Choose the language of the interface. Changes are applied immediately after clicking on Save preferences.

- **Interface theme** – Select the UI skin or theme. The available themes are:
  - dark-green, black and green theme.
  - dark-blue, black and blue theme.
  - dark-orange, orange and blue theme.
Fig. 1: The preferences window of Gaia Sky

- **bright-green**, a bright theme with greenish tones.
- **night-red**, a red theme for low-light environments.

- **Scale UI (HiDPI)** – All themes are also available for HiDPI (high pixel density) screens. To enable this, check the Scale UI (HiDPI) checkbox in the Preferences dialog, Interface tab, or add a the \(-x2\) suffix to the theme name in the properties file.

- **Display pointer coordinates** – Display the coordinates of the pointer, either sky coordinates (equatorial), or latitude and longitude on a planet. The coordinates are shown at the bottom and right edges of the screen, aligned with the pointer.

- **Minimap size** – Adjust the base size of the minimap frame buffers. You can bring up the minimap by clicking on the minimap icon or by pressing Tab.

- **Crosshair** – Adjust the visibility of the different crosshairs and markers.
  - **Focus marker** – Marks the location of the current focus object.
  - **Closest object marker** – Marks the location of the closest object to the camera.
  - **Home object marker** – Marks the location of the home object, defined in the configuration file.

**Performance**

- **Enable multithreading** – Enable using multiple threads.
- **Number of threads** – Adjust the maximum number of threads to use.
- **Smooth transitions between levels of detail** – Fade the contents of octree nodes as they approach the visibility threshold. Improves graphical fidelity and removes pop-ins.
- **Draw distance** – Adjusts the solid angle threshold for when octree nodes become visible. See the Draw distance (levels of detail) section for more information.
More detailed info on performance can be found in the performance section.

Controls

You can see the key-action bindings in the Controls tab. Controls are only editable by modifying the keyboard mappings file inside the mappings folder of your installation. Check out the Controls documentation to know more.

Screenshots

You can take screenshots anytime when the application is running by pressing F5. There are two screenshot modes available: * Simple, the classic screenshot of what is currently on screen, with the same resolution. * Advanced, where you can define the resolution of the screenshots. Additionally, the output format (either JPG or PNG) and the quality (in case of JPG format) can also be set in the configuration file, usually located in $GS_CONFIG/global.properties (see folders). The keys to modify are:

- screenshot.format
- screenshot.quality

- Screenshots save location – Choose the location on disk where the screenshots are to be saved.
- Mode – Choose the screenshot mode, either Simple or Advanced (see above).
- Screenshots size – Adjust the resolution for the Advanced screenshots mode.

Frame output

There is a feature in Gaia Sky that enables the output of every frame as a JPG or PNG image. This is useful to produce videos. In order to configure the frame output system, use the Frame output tab. There you can select the output folder, the image prefix name, the output image resolution (in case of Advanced mode) and the target frame rate. Additionally, the output format (either JPG or PNG) and the quality (in case of JPG format) can also be set in the configuration file, usually located in $GS_CONFIG/global.properties (see folders). The keys to modify are:

- graphics.render.format
- graphics.render.quality

Note: Use F6 to activate the frame output mode and start saving each frame as an image. Use F6 again to deactivate it.

When the program is in frame output mode, it does not run in real time but it adjusts the internal clock to produce as many frames per second as specified here. You have to take it into account when you later use your favourite video encoder (ffmpeg) to convert the frame images into a video.

- Frame save location – Choose the location on disk where the still frames are to be saved.
- Frame name prefix – Choose the prefix to prepend to the still frame files.
- Target FPS – Target framerate of the frame output system.
- Mode – Choose the frame mode, either Simple or Advanced (see above).
- Size of frames – Adjust the resolution for the Advanced mode.
- Reset sequence number – Resets the frame sequence number of the current session to 0. Useful if you need to re-capture frames.
Camrecorder

- **Target FPS** – Here you can set the desired **frames per second** to capture the camera paths. If your device is not fast enough in producing the specified frame rate, the application will slow down while recording so that enough frames are captured. Same behaviour will be uploading during camera playback.

- **Activate frame output automatically** – You can also enable **automatic frame recording** during playback. This will automatically activate the frame output system (see *Frame output*) during a camera file playback.

- **Keyframe preferences** – This button will bring up a new dialog to adjust some preferences of the camera keyframe system. See *this section* for more information.

Panorama mode

- **Cubemap side resolution** – Here you can define the **cube map side resolution** for the 360 mode. With this mode a cube map will be rendered (the whole scene in all direction +X, -X, +Y, -Y, +Z, -Z) and then it will be transformed into a flat image using an equirectangular projection. This allows for the creation of 360 (VR) videos.

Planetarium mode

- **Aperture angle** – Adjust the aperture angle to suit your dome setup.

- **Cubemap side resolution** – The planetarium mode also works with the cube map system used in Panorama mode, so here you can also adjust the cubemap side resolution.

Data

As of version 1.0.0 you can use the **Data** tab to select the catalogues to load. You can select as many catalogs as you want.

- **Use high accuracy positions** – Enable high accuracy positions, which uses all terms in the VSOPxx and other ephemerides algorithms.

- **Data source** – Choose the catalog to load at startup. The changes will be effective on restart.

- **Select catalog at startup** – Check this box if you want Gaia Sky to ask which catalog to load every time it starts.

- **Data download** – Bring up the data download dialog to get more datasets.

Gaia

- **Gaia attitude** – You can either use the **real satellite attitude** (takes a while to load but will ensure that Gaia points to where it should) and the NSL, which is an analytical implementation of the nominal attitude of the satellite. It behaves the same as the real thing, but the observation direction is not ensured.

System

- **Show debug info** – Enable and disable the debug info using the **Show debug info** checkbox. When the debug info is enabled, the program prints the frames per second and other useful information at the top-right of the screen.
• **Reset default settings** – Revert to the default settings. You will lose your current settings file and Gaia Sky will need to be relaunched for the changes to take effect.

### 1.4.2 The configuration file

There is a configuration file which stores most of the configuration settings of Gaia Sky. This section is devoted to these settings that are not represented in the GUI but are still configurable. The configuration file is located in `$GS_CONFIG/global.properties` (see `folders`). The file is annotated with comments specifying the function of most properties. However, here is an explanation of some of the properties found in this file that are not represented in the GUI.

- **data.limit.mag** – this contains the limiting magnitude above which stars shall not be loaded. Not all data loaders implement this. It is now deprecated.
- **scene.octree.maxstars** – the maximum number of stars loaded at a time from LOD (octree-backed) catalogs. Increase this to allow for more stars to be loaded. When this number is hit, the application unloads old, unobserved octants.
- **program.debuginfo** – if this property is set to true, some debug information will be shown at the top right of the window. This contains information such as the number of stars rendered as a quad, the number of stars rendered as a point or the frames per second. This can be activated in real time by pressing `Ctrl + d`.
- **controls.blacklist** – list of game controllers to ignore.

### 1.4.3 Graphics configuration

Most of the graphics settings can be adjusted using the Preferences dialog.

**Resolution and mode**

You can find the Resolution and mode configuration under the Graphics tab.

- **Display mode** – select between fullscreen mode and windowed mode. In the case of full screen, you can choose the resolution from a list of supported resolutions in a drop down menu. If you choose windowed mode, you can enter the resolution you want. You can also choose whether the window should be resizable or not. In order to switch from full screen mode to windowed mode during the execution, use the key `F11`.
- **V-sync** – enable v-sync to limit the frame rate to the refresh rate of your monitor. In some cases this may help reducing tearing.
- **Maximum frame rate** – it is possible to set a maximum frame rate by ticking this checkbox and entering a positive integer value. The frame rate will be capped to that value.

**Graphics settings**

**Graphics quality**

This setting governs the size of the textures, the complexity of the models and also the quality of the graphical effects (light glow, lens flare, etc.). Here are the differences:

- **Low** – very low resolution textures, mostly 1K (1024x512), and fewer sample counts for the visual effects than in higher quality settings.
- **Normal** – moderately low resolution textures (2K when available). The graphical effects use a reasonable amount of quality for nice visuals without compromising the performance too much.
• **High** – high-resolution 4K (3840x2160) textures. Graphical effects use a large number of samples.

• **Ultra** – very high resolution textures (8K, 16K, etc.).

**Antialiasing**

In the **Graphics** tab you can also find the antialiasing configuration. Applying antialiasing removes the jagged edges of the scene and makes it look better. However, it does not come free of cost, and usually has a penalty on the frames per second (FPS). There are four main options, described below.

Find more information on antialiasing in the **Antialiasing** section.

**No Antialiasing**

If you choose this no antialiasing will be applied, and therefore you will probably see jagged edges around models. This has no penalty on either the CPU or the GPU. If want you enable antialiasing with **override application settings** in your graphics card driver configuration program, you can leave the application antialiasing setting to off.

**FXAA – Fast Approximate Antialiasing**

This is a post-processing antialiasing filter which is very fast and produces very good results. The performance hit depends on how fast your graphics card is, but it is generally low. Since it is a post-processing effect, this will work also when you take screenshots or output the frames. As of Gaia Sky 2.2.5, FXAA is activated by default. Here is more info on FXAA.

**NFAA – Normal Field Antialiasing**

This is yet another post-processing antialiasing technique. It is based on generating a normal map to detect the edges for later smoothing. It may look better on some devices and the penalty in FPS is small. It will also work for the screenshots and frame outputs.

**Orbit style**

Style to render the orbits.

• **Use line style** – use whatever style is defined in the **Line style** setting.

• **GPU VBOs** – very fast, but slightly inaccurate and bad-looking. Enable for performance, as everything is done in the GPU.

**Line style**

Select the line rendering backend.

• **GL lines** – use the line primitives offered by the graphics driver. Quite fast.

• **Polyline quadstrips** – use polygon lines. Better looking but slower.

**Bloom effect**

Select the amount of bloom to apply to the scene.
Lens flare

Activate the pseudo lens flare effect.

Motion blur

Activate the motion blur effect.

Elevation representation

Choose the way elevation (height) is represented in Gaia Sky.

- **Tessellation** – use geometry subdivision.
- **Parallax mapping** – use parallax mapping in the fragment shaders.
- **None** – do not represent elevation.

Shadows

Enable or disable shadows, and choose their properties.

- **Shadow map resolution** – choose the resolution of the shadow map textures to use.
- **# shadows** – how many shadows are active at a time in the scene.

Image levels

Control the image levels

- **Brightness** – overall brightness of the image.
- **Contrast** – overall contrast of the image.
- **Hue** – hue value of the image.
- **Saturation** – saturation value of the image.
- **Gamma correction** – gamma correction value of the image. This should be calibrated with your monitor.
- **HDR tone mapping type** – tone mapping algorithm to use. Choose *Automatic* to use a real-time adjusting mode based on the overall lightness of the image. All the others are static algorithms.

1.4.4 User Interface

Data download manager

As of version 2.1.0 Gaia Sky provides an integrated download manager to help visualize and obtain the available data packs and catalogs. Chances are that the download manager is the first thing you see when you launch Gaia Sky for the first time.

The download manager pops up automatically when Gaia Sky is started if no base data or no catalog files are detected. It can also be launched manually from the preferences window, data tab.
Using the download manager, the user can select whatever datasets she wants, then click download and wait for the download and extract processes to finish. Once done, the data will be available to Gaia Sky the next time it starts.

**GUI window**

The Gaia Sky GUI is divided into seven panes, *Time, Camera, Type visibility, Visual settings, Datasets, Objects*, and *Music*.

<table>
<thead>
<tr>
<th>Controls window with all panes, except the Time pane, collapsed.</th>
</tr>
</thead>
</table>

| Controls window with the Time pane and the Camera pane expanded. |

The seven panes, except for the Time pane, are collapsed at startup. To expand them and reveal its controls just click on the little plus icon at the right of the pane title. Use the minus icon to collapse it again. Panes can also be detached to their own window. To do so, use the detach icon.

**Time**

You can play and pause the simulation using the Play/Pause buttons in the Controls window to the left. You can also use *Space* to play and pause the time. You can also change time warp, which is expressed as a factor. Use , and . to divide by 2 and double the value of the time warp.

**Camera**

In the camera options pane on the left you can select the type of camera. This can also be done by using the *Numpad 0-4* keys.

There are five camera modes:

- **Free mode** – the camera is not linked to any object and its velocity is exponential with respect to the distance to the origin (Sun).
- **Focus mode** – the camera is linked to a focus object and it rotates and rolls with respect to it.
- **Game mode** – a game mode which maps the controls *wasd* + mouse look.
- **Gaia scene** – outside view of the Gaia satellite. The camera can not be rotated or translated in this mode.
- **Spacecraft** – take control of a spacecraft and navigate around at will.
- **Gaia FOV** – the camera simulates either of the fields of view of Gaia, or both.

For more information on the camera modes, see the *Camera modes* section.

Additionally, there are a number of sliders for you to control different parameters of the camera:

- **Field of view** – control the field of view angle of the camera. The bigger it is, the larger the portion of the scene represented.
- **Camera speed** – control the longitudinal speed of the camera.
- **Rotation speed** – control the transversal speed of the camera, how fast it rotates around an object.
- **Turn speed** – control the turning speed of the camera.
You can lock the camera to the focus when in focus mode. Doing so links the reference system of the camera to that of the object and thus it moves with it.

**Hint:** Lock the camera so that it stays at the same relative position to the focus object.

Finally, we can also lock the orientation of the camera to that of the focus so that the same transformation matrix is applied to both.

**Hint:** Lock the orientation so that the camera also rotates with the focus.

Additionally, we can also enable the crosshair, which will mark the currently focused object.

### Type visibility

Most graphical elements can be turned off and on using these toggles. For example you can remove the stars from the display by clicking on the stars toggle. The object types available are the following:

- Stars
- Planets
- Moons
- Satellites, the spacecrafts
- Asteroids
- Labels, all the text labels
- Equatorial grid
- Ecliptic grid
- Galactic grid
- Orbits, the orbit lines
- Atmospheres, the atmospheres of planets
- Constellations, the constellation lines
- Boundaries, the constellation boundaries
- Milky way
- Others

By checking the proper motion vectors checkbox we can enable the representation of star proper motions if the currently loaded catalog provides them. Once proper motions are activated, we can control the number of displayed proper motions and their length by using the two sliders that appear.

### Visual settings

Here are a few options to control the lighting of the scene:

- **Star brightness** – control the brightness of stars.
- **Star size** – control the size of point-like stars.
• **Min. star opacity** – set a minimum opacity for the faintest stars.

• **Ambient light** – control the amount of ambient light. This only affects the models such as the planets or satellites.

• **Label size** – control the size of the labels.

• **Elevation multiplier** – scale the height representation.

**Objects**

There is a list of focus objects that can be selected from the interface. When an object is selected the camera automatically centers it in the view and you can rotate around it or zoom in and out. Objects can also be selected by double-clicking on them directly in the view or by using the search box provided above the list. You can also invoke a search dialogue by pressing `Ctrl + f` or simply `f`.

**Datasets**

This tab contains all the datasets currently loaded. For each dataset, a highlight color can be defined. The dataset properties window can be used to define arbitrary filters on any of the properties of the elements of the dataset. Datasets can be highlighted by clicking on the little crosshair next to the name, below the color picker.

**Music**

Since version 0.800b, Gaia Sky also offers a music player in its interface. By default it ships with only a few *spacey* melody, but you can add your own by dropping them in the folder `$GS_DATA/music` (see folders).

**Hint:** Drop your mp3, ogg or wav files in the folder `$GS_DATA/music` and these will be available during your Gaia Sky sessions to play.

In order to start playing, click on the Play button. To pause the track, click on the Pause icon. To skip to the next track, click on the Forward icon. To go to the previous track, click on the Backward icon. The volume can be controlled by scrolling up and down, using the mouse wheel, over the volume indicator. The default volume value is 5%.

**Bottom buttons**

The buttons at the bottom of the control panel are described here.

**Minimap**

With this button you can toggle the minimap window on or off. You can also use Tab.
Load dataset

Use this button to load a new VOTable file (.vot) into Gaia Sky. Refer to the STIL data provider section for more information on the metadata needed for Gaia Sky to parse the dataset correctly.

Preferences window

You can launch the preferences window any time during the execution of the program. To do so, click on the Preferences button at the bottom of the GUI window. For a detailed description of the configuration options refer to the Configuration Instructions.

Log

The log button brings up the log window, which displays the Gaia Sky log for the current session. The log can be exported to a file by clicking on the Export to file button. The location of the exported log files is $GS_DATA (see folders).

About/help

The help button brings up the help dialog, where information on the current system, OpenGL settings, Java memory, updates and contact can be found.

Exit

Click on the cross icon to exit Gaia Sky. You can also use Esc.

Spacecraft UI controls

The spacecraft mode UI is described in the Spacecraft mode section.

1.4.5 Controls

This section describes the controls of Gaia Sky.

Keyboard controls

To check the most up-to-date controls go to the Controls tab in the preferences window. Here are the default keyboard controls depending on the current camera mode. Learn more about camera modes in the Camera modes section.
Keyboard mappings

The keyboard mappings are stored in an internal file called `keyboard.mappings` (link). If you want to edit the keyboard mappings, copy the file into `$GS_CONFIG/mappings/` and edit it. This overrides the default internal mappings file. The file consists of a series of `<ACTION>=<KEYS>` entries. For example:

```
# Help
action.help = F1
action.help = H

# Exit
action.exit = ESC

# Home
action.home = HOME

# Preferences
action.preferences = P

#action.playcamera = C
```

The available actions are defined in this I18n file here.

Focus and free camera modes

These keyboard controls apply to the focus mode and also to the free mode.

<table>
<thead>
<tr>
<th>Key(s)</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑</td>
<td>Camera forward</td>
</tr>
<tr>
<td>↓</td>
<td>Camera backward</td>
</tr>
<tr>
<td>←</td>
<td>Rotate/yaw left</td>
</tr>
<tr>
<td>→</td>
<td>Rotate/yaw right</td>
</tr>
<tr>
<td>Home</td>
<td>Back to Earth (or any other home object)</td>
</tr>
<tr>
<td>Tab</td>
<td>Toggle minimap</td>
</tr>
<tr>
<td>Ctrl + r</td>
<td>Reset time to current</td>
</tr>
<tr>
<td>Num 0 or 0</td>
<td>Free camera</td>
</tr>
<tr>
<td>Num 1 or 1</td>
<td>Focus camera</td>
</tr>
<tr>
<td>Num 2 or 2</td>
<td>Game mode</td>
</tr>
<tr>
<td>Num 3 or 3</td>
<td>Gaia scene mode</td>
</tr>
<tr>
<td>Num 4 or 4</td>
<td>Spacecraft mode</td>
</tr>
<tr>
<td>Num 5 or 5</td>
<td>Gaia FOV1 camera</td>
</tr>
<tr>
<td>Num 6 or 6</td>
<td>Gaia FOV2 camera</td>
</tr>
<tr>
<td>Num 7 or 7</td>
<td>Gaia FOV1 and FOV2 camera</td>
</tr>
<tr>
<td>Ctrl + w</td>
<td>New keyframe</td>
</tr>
<tr>
<td>Ctrl + k</td>
<td>Panorama mode</td>
</tr>
<tr>
<td>Space</td>
<td>Pause/resume time</td>
</tr>
<tr>
<td>F1</td>
<td>Help dialog</td>
</tr>
<tr>
<td>F5</td>
<td>Take screenshot</td>
</tr>
<tr>
<td>F6</td>
<td>Start/stop frame output mode</td>
</tr>
<tr>
<td>F11</td>
<td>Toggle fullscreen/windowed mode</td>
</tr>
<tr>
<td>Ctrl + f or f</td>
<td>Search dialog</td>
</tr>
<tr>
<td>Esc or q</td>
<td>Quit application</td>
</tr>
</tbody>
</table>

Continued on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Key(s)</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>Open preferences dialog</td>
</tr>
<tr>
<td>$h$</td>
<td>Open help dialog</td>
</tr>
<tr>
<td>$r$</td>
<td>Run script dialog</td>
</tr>
<tr>
<td>$c$</td>
<td>Run camera path file dialog</td>
</tr>
<tr>
<td>$-$</td>
<td>Decrease limiting magnitude</td>
</tr>
<tr>
<td>$+$</td>
<td>Increase limiting magnitude</td>
</tr>
<tr>
<td>$.$</td>
<td>Divide time warp by two</td>
</tr>
<tr>
<td>$.$</td>
<td>Double time warp</td>
</tr>
<tr>
<td>*</td>
<td>Reset limiting magnitude</td>
</tr>
<tr>
<td>$\text{Shift} + b$</td>
<td>Toggle constellation boundaries</td>
</tr>
<tr>
<td>$\text{Shift} + c$</td>
<td>Toggle constellation lines</td>
</tr>
<tr>
<td>$\text{Shift} + e$</td>
<td>Toggle ecliptic grid</td>
</tr>
<tr>
<td>$\text{Shift} + g$</td>
<td>Toggle galactic grid</td>
</tr>
<tr>
<td>$\text{Shift} + l$</td>
<td>Toggle labels</td>
</tr>
<tr>
<td>$\text{Shift} + m$</td>
<td>Toggle moons</td>
</tr>
<tr>
<td>$\text{Shift} + o$</td>
<td>Toggle orbits</td>
</tr>
<tr>
<td>$\text{Shift} + p$</td>
<td>Toggle planets</td>
</tr>
<tr>
<td>$\text{Shift} + q$</td>
<td>Toggle equatorial grid</td>
</tr>
<tr>
<td>$\text{Shift} + s$</td>
<td>Toggle stars</td>
</tr>
<tr>
<td>$\text{Shift} + t$</td>
<td>Toggle satellites</td>
</tr>
<tr>
<td>$\text{Shift} + v$</td>
<td>Toggle star clusters</td>
</tr>
<tr>
<td>$\text{Shift} + h$</td>
<td>Toggle meshes</td>
</tr>
<tr>
<td>$\text{Shift} + u$</td>
<td>Expand/collapse controls window</td>
</tr>
<tr>
<td>$\text{Ctrl} + u$</td>
<td>Toggle UI completely (hide/show user interface)</td>
</tr>
<tr>
<td>$\text{Ctrl} + d$</td>
<td>Toggle debug info</td>
</tr>
<tr>
<td>$\text{Ctrl} + s$</td>
<td>Toggle stereoscopic mode</td>
</tr>
<tr>
<td>$\text{Ctrl} + \text{Shift} + s$</td>
<td>Switch between stereoscopic profiles</td>
</tr>
<tr>
<td>$\text{Ctrl} + k$</td>
<td>Toggle 360 panorama mode</td>
</tr>
<tr>
<td>$\text{Ctrl} + \text{Shift} + k$</td>
<td>Switch between 360 projections</td>
</tr>
<tr>
<td>$\text{Ctrl} + \text{Shift} + g$</td>
<td>Toggle galaxy renderer</td>
</tr>
</tbody>
</table>

**Spacecraft mode**

These controls apply only to the spacecraft mode.

<table>
<thead>
<tr>
<th>Key(s)</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w$</td>
<td>Apply forward thrust</td>
</tr>
<tr>
<td>$s$</td>
<td>Apply backward thrust</td>
</tr>
<tr>
<td>$a$</td>
<td>Roll to the left</td>
</tr>
<tr>
<td>$d$</td>
<td>Roll to the right</td>
</tr>
<tr>
<td>$k$</td>
<td>Stop spaceship automatically</td>
</tr>
<tr>
<td>$l$</td>
<td>Stabilize spaceship automatically</td>
</tr>
<tr>
<td>$\uparrow$</td>
<td>Decrease pitch angle</td>
</tr>
<tr>
<td>$\downarrow$</td>
<td>Increase pitch angle</td>
</tr>
<tr>
<td>$\leftarrow$</td>
<td>Increase yaw angle</td>
</tr>
<tr>
<td>$\rightarrow$</td>
<td>Decrease yaw angle</td>
</tr>
<tr>
<td>$\text{PgUp}$</td>
<td>Increase engine power by a factor of 10</td>
</tr>
<tr>
<td>$\text{PgDown}$</td>
<td>Decrease engine power by a factor of 10</td>
</tr>
</tbody>
</table>
Mouse controls

Here are the default mouse controls for the focus and free Camera modes. The other modes do not have mouse controls.

Focus mode

<table>
<thead>
<tr>
<th>Mouse + keys</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-MOUSE DOUBLE CLICK</td>
<td>Select focus object</td>
</tr>
<tr>
<td>L-MOUSE SINGLE CLICK</td>
<td>Stop all rotation and translation movement</td>
</tr>
<tr>
<td>L-MOUSE + DRAG</td>
<td>Apply rotation around focus</td>
</tr>
<tr>
<td>L-MOUSE + Shift + DRAG</td>
<td>Camera roll</td>
</tr>
<tr>
<td>R-MOUSE + DRAG</td>
<td>Pan view freely from focus</td>
</tr>
<tr>
<td>M-MOUSE + DRAG or WHEEL</td>
<td>Move towards/away from focus</td>
</tr>
</tbody>
</table>

Free mode

<table>
<thead>
<tr>
<th>Mouse + keys</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-MOUSE DOUBLE CLICK</td>
<td>Select object as focus (changes to focus mode)</td>
</tr>
<tr>
<td>L-MOUSE SINGLE CLICK</td>
<td>Stop all rotation and translation movement</td>
</tr>
<tr>
<td>L-MOUSE + DRAG</td>
<td>Pan view</td>
</tr>
<tr>
<td>L-MOUSE + Shift + DRAG</td>
<td>Camera roll</td>
</tr>
<tr>
<td>M-MOUSE + DRAG or WHEEL</td>
<td>Forward/backward movement</td>
</tr>
</tbody>
</table>

Game mode

Use the mouse to look around and wasd to move.

Gamepad controls

Gaia Sky supports (as of version 1.5.0) mappings for different controller types. However, so far only the mappings files for the Xbox 360 controller and the PS3 controller are provided.

Sometimes there are differences between the axes and buttons codes for the same controller device between operating systems. To solve this issue, we offer a way to describe operating system specific mappings. To do so, create a new mappings file with the format [controller_name].[os_family].controller, where os_family is linux, win, macos, unix or solaris. If the mappings for the given file name and OS family are found, they will be used. Otherwise, the file defined in the configuration file is used. For example, if we have the file xbox360. controller is defined in the configuration file, the system will look up xbox360.win.controller if on Windows, xbox360.linux.controller if on Linux, and so on. If found, the file is used. Otherwise, the default xbox360.controller file is used. Gaia Sky provides the default xbox360.controller file, which defines the Linux mappings, and also the Windows mappings xbox360.win.controller.

The mappings files (see here) must be in the $GS_CONFIG/mappings (see folders) folder, and basically assign the button and axis codes for the particular controller to the actions.
The actions depend on the current camera mode (focus, free, spacecraft), and are described below.

**Creating mappings files for new controllers**

As of version 1.5.1 a new controller debug mode has been added to help create new mappings files. This mode prints to the log all key press and release events with their respective key codes, as well as trigger events, values and codes. It also prints controller connection and disconnection events.

In order to enable the controller debug mode, set the property `controls.debugmode=true` in the `$GS_CONFIG/global.properties` file.

Put your new files in `$GS_CONFIG/mappings/`. The name of the file should be `controller brand and model`.mappings. For example, `xboxone.mappings` or `logitech_f310.mappings`.

Please, if you create mappings files for new game controllers, create a pull request in the gaiasky gitlab so that the community can benefit.

### Focus mode

<table>
<thead>
<tr>
<th>Property</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>button.velocityhalf</code></td>
<td>Hold to apply 0.5 factor to speed</td>
</tr>
<tr>
<td><code>button.velocitytenth</code></td>
<td>Hold to apply 0.1 factor to speed</td>
</tr>
<tr>
<td><code>axis.velocitydown</code></td>
<td>Move away from focus</td>
</tr>
<tr>
<td><code>axis.velocityup</code></td>
<td>Move towards focus</td>
</tr>
<tr>
<td><code>axis.yaw</code></td>
<td>Horizontal rotation around focus</td>
</tr>
<tr>
<td><code>axis.pitch</code></td>
<td>Vertical rotation around focus</td>
</tr>
<tr>
<td><code>axis.roll</code></td>
<td>Roll right and left</td>
</tr>
<tr>
<td><code>axis.move</code></td>
<td>Move towards or away from focus</td>
</tr>
<tr>
<td><code>button.velocityup</code></td>
<td>Move towards focus</td>
</tr>
<tr>
<td><code>button.velocitydown</code></td>
<td>Move away from focus</td>
</tr>
</tbody>
</table>
Fig. 2: Xbox 360 controller in focus mode
Free camera mode

<table>
<thead>
<tr>
<th>Axis/button</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>button.velocityhalf</td>
<td>Hold to apply 0.5 factor to speed</td>
</tr>
<tr>
<td>button.velocitytenth</td>
<td>Hold to apply 0.1 factor to speed</td>
</tr>
<tr>
<td>axis.velocitydown</td>
<td>Move away from focus</td>
</tr>
<tr>
<td>axis.velocityup</td>
<td>Move towards focus</td>
</tr>
<tr>
<td>axis.yaw</td>
<td>Yaw right and left</td>
</tr>
<tr>
<td>axis.pitch</td>
<td>Pitch up and down</td>
</tr>
<tr>
<td>axis.roll</td>
<td>Move sideways</td>
</tr>
<tr>
<td>axis.move</td>
<td>Move forward and backward</td>
</tr>
<tr>
<td>button.velocityup</td>
<td>Move towards focus</td>
</tr>
<tr>
<td>button.velocitydown</td>
<td>Move away from focus</td>
</tr>
</tbody>
</table>

Spacecraft mode

<table>
<thead>
<tr>
<th>Axis/button</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>button.velocityhalf</td>
<td>Stabilise spacecraft rotations</td>
</tr>
<tr>
<td>button.velocitytenth</td>
<td>Stop spacecraft</td>
</tr>
<tr>
<td>axis.velocitydown</td>
<td>Apply backward thrust</td>
</tr>
<tr>
<td>axis.velocityup</td>
<td>Apply forward thrust</td>
</tr>
<tr>
<td>axis.yaw</td>
<td>Yaw right and left</td>
</tr>
<tr>
<td>axis.pitch</td>
<td>Pitch up and down</td>
</tr>
<tr>
<td>axis.roll</td>
<td>Roll right and left</td>
</tr>
<tr>
<td>axis.move</td>
<td>None</td>
</tr>
<tr>
<td>button.velocityup</td>
<td>Increase engine power</td>
</tr>
<tr>
<td>button.velocitydown</td>
<td>Decrease engine power</td>
</tr>
</tbody>
</table>

Touch controls

No mobile version yet.

1.4.6 Camera modes

Gaia Sky offers five basic camera modes.

Focus mode

This is the default mode. In this mode the camera movement is locked to a focus object, which can be selected by double clicking or by using the find dialog (Ctrl + F). There are two extra options available. These can be activated using the checkboxes at the bottom of the Camera panel in the GUI Controls window:

- **Lock camera to object** – the relative position of the camera with respect to the focus object is maintained. Otherwise, the camera position does not change.

- **Lock orientation** – the camera rotates with the object to keep the same perspective of it at all times.

The description of the controls in focus mode can be found here:

- Keyboard controls in focus mode
• Mouse controls in focus mode
• Gamepad controls in focus mode

**Hint:** Numpad 1 – Enter focus mode

---

**Free mode**

This mode does not lock the camera to a focus object but it lets it roam free in space.

• Keyboard controls in free mode
• Mouse controls in free mode
• Gamepad controls in free mode

**Hint:** Numpad 0 – Enter free mode

---

**Game mode**

This mode maps the standard control system for most games (wasd + Mouse look) in Gaia Sky. Additionally, it is possible to add gravity to objects, so that when the camera is closer to a planet than a certain threshold, gravity will pull it to the ground. Quit

**Hint:** Numpad 2 – Enter game mode

---

**Gaia scene mode**

In this mode the camera can not be controlled. It provides a view of the Gaia satellite from the outside.

**Hint:** Numpad 3 – Enter Gaia scene mode

---

**Spacecraft mode**

In this mode you take control of a spacecraft. In the spacecraft mode, the GUI changes completely. The Options window disappears and a new user interface is shown in its place at the bottom left of the screen.

• **Attitude indicator** – it is shown as a ball with the horizon and other marks. It represents the current orientation of the spacecraft with respect to the equatorial system.

• – indicates the direction the spacecraft is currently headed to.

• – indicates direction of the current velocity vector, if any.
• – indicates inverse direction of the current velocity vector, if any.

• **Engine Power** – current power of the engine. It is a multiplier in steps of powers of ten. Low engine power levels allow for Solar System or planetary travel, whereas high engine power levels are suitable for galactic and intergalactic exploration. Increase the power clicking on and decrease it clicking on .

• – stabilises the yaw, pitch and roll angles. If rotation is applied during the stabilisation, the stabilisation is cancelled.

• – stops the spacecraft until its velocity with respect to the Sun is 0. If thrust is applied during the stopping, the stopping is cancelled.

• – return to the focus mode.

Additionally, it is possible to adjust three more parameters:

• **Responsiveness** – this slider controls how fast the spacecraft reacts to the user’s yaw/pitch/roll commands. It could be seen as the power of the thrusters.

• **Drag** – this slider controls the friction force applied to all the forces acting on the spacecraft (engine force, yaw, pitch, and roll). Set it to zero for a real zero G simulation.

• **Force velocity to heading direction** – this checkbox, when checked, makes the spacecraft to always move in the direction it is facing, instead of using the regular momentum-based motion. Even though physically inaccurate, this makes it much easier to control and arguably more fun to play with.

• **Keyboard controls in spacecraft mode**

• **Gamepad controls in spacecraft mode**

---

**Hint:** NUMPAD_4 – Enter spacecraft mode

---

**Field of View mode**

This mode simulates the Gaia fields of view. You can select FoV1, FoV2 or both.

**Hint:** Numpad 5 – Enter Field of View 1 mode

* Numpad 6 – Enter Field of View 2 mode

* Numpad 7 – Enter Field of View 1 and 2 mode

---

**1.4.7 Cinematic camera**

Since version 1.5.0 a new option is available in the user interface to control the behaviour of the camera, the cinematic mode toggle. The cinematic mode is in fact the same exact behaviour the camera has had in Gaia Sky since the first release. If cinematic mode is not enabled, however, the camera adopts a new behaviour which is much more responsive.
Cinematic behaviour

This behaviour makes the camera use acceleration and momentum, leading to very smooth transitions and movements. This is the ideal camera to use when recording camera paths or when showcasing to an audience.

Non-cinematic behaviour

In this behaviour the camera becomes much more responsive to the user’s commands and inputs. There is no longer an acceleration factor, and momentum is very minimal. This is the default behaviour as of version 1.5.0 and probably better meets the expectations of new users.

1.4.8 Stereoscopic (3D) mode

Gaia Sky includes a stereoscopic mode or 3D mode which outputs two images each intended for each eye, creating the illusion of depth.

Hint:

or Ctrl + S – Activate the stereoscopic mode
Ctrl + Shift + S – Switch between 3D profiles
Stereoscopic profiles

Usually, as the images are placed side by side (even though most 3DTVs also support up and down), the right image is intended for the right eye and the left image is intended for the left eye. This works with 3DTVs and VR head sets (such as the Oculus Rift, Google cardboard, etc.). In 3DTVs, however, the image is distorted because each half of the TV will be stretched back to the whole TV area when the 3D mode is on.

Note: As of version 1.5.0, the head tracking is not yet implemented, so you won’t be able to use Gaia Sky interactively with a VR headset. The integration with OpenVR will come soon enough.

Additionally, there are a couple of techniques called cross-eye 3D (you can find some examples here, and here is a very nice video teaching the concept and how to achieve it) and parallel view. These work without any extra equipment and consist on trying to focus your eyes some distance before or after the actual image so that each eye receives the correct image. In cross-eye this case the right images goes to the left eye and the left image goes to the right eye. The opposite is true for parallel view images.

In order to manage all these parameters, we have created 6 stereoscopic profiles which can be selected by the user and are described below.

- **VR HEADSET** – The left image goes to the left eye. Lens distortion is applied to be viewed with VR glasses.
- **Crosseye** – The left image goes to the right eye. No distortion is applied.
- **Parallel view** – The left image goes to the left eye. No distortion is applied.
- **3DTV Horizontal** – The left image goes to the left eye. The left and right images are stretched to fit in a half of the screen.
- **3DTV Vertical** – The top image goes to the left eye. Top and bottom images are stretched to fit in half of the screen.
- **Anaglyph 3D** – To use with red-cyan glasses. Displays both the left and right images at full resolution. Left image contains the red channel, right image contains the green and blue channels.

Hint: Ctrl + Shift + S – Switch between 3D profiles
<table>
<thead>
<tr>
<th>Profile</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>VR_HEADSET</td>
<td><img src="image1" alt="VR_HEADSET Image" /> <img src="image2" alt="VR_HEADSET Image" /></td>
</tr>
<tr>
<td>Crosseye</td>
<td><img src="image3" alt="Crosseye Image" /> <img src="image4" alt="Crosseye Image" /></td>
</tr>
<tr>
<td>Parallel View</td>
<td><img src="image5" alt="Parallel View Image" /> <img src="image6" alt="Parallel View Image" /></td>
</tr>
</tbody>
</table>
1.4.9 Planetarium mode

Gaia Sky supports a couple of planetarium modes, depending on the projector setup.

Single-projector setup

Gaia Sky can output a true fisheye stream suitable for single-projector dome setups. If you need to separate the UI from the planetarium render window, you have two options:

- Create an external view: External views.
- Connect two instances running (possibly) on different computers: Connecting Gaia Sky instances.

This planetarium mode runs by rendering the scene into a cubemap (using separate renders for all directions +X, -X, +Y, -Y, +Z, -Z) and renders it using a fisheye projection. A few things need to be considered:

- **Cubemap side resolution** – The resolution of each of the sides of the cubemap can be adjusted in the preferences window, planetarium mode tab.
- **Aperture angle** – The default aperture is 180, corresponding to a half-sphere. However this angle can be adjusted to suit different dome types in the planetarium mode tab of the preferences window.
- **View skew** – in focus mode, the view is skewed about 50 degrees downwards. This setting is not adjustable as of now.

**Hint:** To activate the planetarium mode, click on the 🌌 icon in the camera section of the controls window. Exit by clicking 🌌 again. You can also use the shortcut *CTRL + P*.

Multi-projector setup

Gaia Sky offers support for multi-projector setups, where a number of slave instances (each with its own viewport, field of view, warp and blend settings), are synchronized with a master (presenter) instance. Each slave is in charge of producing the image for a single projector and has a different view setup, geometry warp and blend mask. The current section only deals with the configuration of the view, warp and blend parameters for each slave.

**Hint:** The configuration and setup of the connection between master and slave instances is documented in the “Connecting Gaia Sky instances” section.

Additionally to the configuration needed to connect master and slaves, the slave instances need a special view, resolution, warp and blend configuration. These depend on the specifications, location and orientation of each projector, as well as the projection surface.

The following settings can be configured:

- The yaw angle – turn the camera right
- The pitch angle – turn the camera up
- The roll angle – rotate the camera clock-wise
- The field of view angle
- The geometry warp file – a PFM file that contains the destination location for each source location in normalized coordinates
Fig. 4: Planetarium mode
• The blend mask – a 8-bit RGB or RGBA PNG file with the blending mask

Gaia Sky offers two ways to configure these settings for each slave instance:

• Using the MPCDI standard file format
• Using the configuration file of Gaia Sky directly

**MPCDI**

Gaia Sky partially supports the MPCDI format in order to configure each instance. You will need a single .mpcdi file for each projector/gaia sky instance. Each file contains the resolution, the yaw, pitch and roll angles, the field of view angle and optionally a PFM warp file and a PNG blend mask. Gaia Sky does not support the MPCDI format fully, here are some caveats.

• Only the ‘3d’ profile is supported
• Only one buffer per display is supported
• Only one region per buffer is supported, and this region must cover the full frame
• Only linear interpolation is supported for the warp file

In order to set the .mpcdi file for an instance, set/edit the following property in the instance’s configuration file:

```
program.net.slave.config=[path_to_file]/instance_config.mpcdi
```

**Gaia Sky configuration file**

If you do not have the MPCDI files for your projector setup, you can also configure each instance directly using the Gaia Sky properties file for that instance.

Usually, each instance has a configuration file with the name global.slave[n].properties, without the brackets, where n is the slave number. Open this file for each instance and set/edit the following properties.

```
# If you don't have an mpcdi file, use these next properties to
# configure the orientation. In order for this to work, you also
# need to set fullscreen=true, the right fullscreen resolution
# and the right field of view angle.

# Yaw angle (turn head right)
program.net.slave.yaw=[yaw angle in degrees]

# Pitch angle (turn head up)
program.net.slave.pitch=[pitch angle in degrees]

# Roll angle (rotate head cw)
program.net.slave.roll=[roll angle in degrees]

# Warp pfm file
program.net.slave.warp=[path to PFM warp file]

# Blend png file
program.net.slave.blend=[path to 8-bit RGB or RGBA PNG file to use as blend mask]
```

**1.4.10 Panorama mode**

Gaia Sky includes a panorama mode where the scene is rendered in all directions to a cube map. This cube map is then projected onto a flat image. The projection to use can be an equirectangular (spherical) projection, a cylindrical projection or a Hammer projection. The final image can be used to create 360 videos with head tracking (see here)
Hint: To activate the panorama mode, click on the icon in the camera section of the controls window. Exit by clicking again.

Hint: Ctrl + k – Toggle the panorama mode.

Hint: Ctrl + Shift + k – Cycle between the projections (spherical, cylindrical and Hammer).

Configuration

Please, see the Panorama mode section.

Creating panorama images

In order to create panorama images that can be viewed with a VR device or simply a 360 viewer, we need to take into consideration a few points.

- You should probably use the equirectangular (spherical) projection, as it is the simplest and the one most programs use.

- Panoramas work best if their aspect ratio is 2:1, so a resolution of 5300x2650 or similar should work. (Refer to the Screenshots section to learn how to take screenshots with an arbitrary resolution).

- Some services (like Google) have strong constraints on image properties. For instance, they must be at least 14 megapixels and in jpeg format. Learn more here.

- Some metadata needs to be injected into the image file.

Injecting panorama metadata to 360 images

To do so, we can use ExifTool in Linux, MacOS and Windows. To inject the metadata which describes a 360 degrees 4K image (3840x2160) we need to run the following command:

```
$ exiftool -UsePanoramaViewer=True -ProjectionType=equirectangular -
   PoseHeadingDegrees=360.0 -CroppedAreaLeftPixels=0 -FullPanoWidthPixels=3840 -
   CroppedAreaImageHeightPixels=2160 -FullPanoHeightPixels=2160 -
   CroppedAreaImageWidthPixels=3840 -CroppedAreaTopPixels=0 -
   LargestValidInteriorRectLeft=0 -LargestValidInteriorRectTop=0 -
   LargestValidInteriorRectWidth=3840 -LargestValidInteriorRectHeight=2160 image_name.jpg
```
Now we can enjoy our image in any 360 panorama viewer like Google Street View app or the Cardboard Camera! Find some examples in this album.

![Panorama image captured with Gaia Sky](image)

**Fig. 5: Panorama image captured with Gaia Sky**

### Creating spherical (360) videos

First, you need to capture the 360 video. To do so, capture the images and use ffmpeg to encode them or capture the video directly using a screen recorder. See the *Capturing videos* section for more information. Once you have the .mp4 video file, you must use the spatial media project to inject the spherical metadata so that video players that support it can play it correctly.

First, clone the project.

```
$ git clone https://github.com/google/spatial-media.git
$ cd spatial-media/
```

Then, inject the spherical metadata with the following command. Python 2.7 must be used to run the tool, so make sure to use that version.

```
$ python spatialmedia -i <input_file> <output_file>
```

You are done, your video can now be viewed using any 360 video player or even uploaded to YouTube.

To check whether the metadata has been injected correctly, just do:

```
$ python spatialmedia <file>
```

### 1.4.11 Datasets

Gaia Sky supports the loading and visualization of datasets of different nature.
Loading datasets

Datasets can be loaded into Gaia Sky by three different means:

- Via SAMP (see this section).
- Via scripting (see this section).
- Directly using the UI. See the next paragraph.

In order to load a dataset, click on the folder icon in the controls window and choose the file you want to load. Right now, .csv (Comma-Separated Values) and .vot (VOTable) formats are supported through the STIL library. Please, see the STIL data provider section for more information on how to prepare the datasets for Gaia Sky.

Managing datasets

![Fig. 6: Dataset entry in the controls window](image)

All datasets currently loaded are displayed in the Datasets pane of the controls window. A few actions can be carried out directly from that pane:

- Toggle the visibility of the dataset
- Open the dataset preferences window
- Delete (unload) dataset
- Highlight the dataset using the current color and particle size. The color can be changed by clicking on the icon just above of this icon (blue square in the image), and the particle size factor can be adjusted in the dataset preferences window.

Dataset highlight color

Above the icon is the color icon. Use it to define the highlight color for the dataset. The color can either be a plain color or a color map.

A plain color can be chosen using the color picker dialog that appears when clicking on the “Plain color” radio button. The “Color map” radio button displays the screen shown below. From there, you can choose the color map type, as well as the attribute to use for the mapping and the maximum and minimum mapping values.

The available attributes depend on the dataset type and loading method. Particle datasets have coordinate attributes (right ascension, declination, ecliptic longitude and latitude, galactic longitude and latitude) and distance distance. Star datasets have, additionally, apparent and absolute magnitudes, proper motions (in alpha and delta) and radial velocity. For all datasets loaded from VOTable either directly or through samp, all the numeric attributes are also available.

Dataset preferences

Clicking on the preferences icon, a few extra settings can be adjusted:
Fig. 7: The plain color picker dialog

Fig. 8: The color map dialog
• **Size increase factor** - Scale factor to apply to the particles when the dataset is highlighted.

• **Make all particles visible** - Raises the minimum opacity to a non-zero value when the dataset is highlighted.

• **Filters** - Allows for the creation of arbitrary selection filters by setting conditions (rules) on the particle attributes. Several rules can be defined, but only one type of logical operator (AND, OR) is possible. The available attributes depend on the dataset and are covered in the dataset color settings.

### 1.4.12 External views

Gaia Sky offers a mode to create an additional window with an external view of the current scene and no user interface controls. This may be useful when presenting or in order to project the view to an external screen or dome.

In order to create the view, just use the `-e` or `--externalview` flags when launching Gaia Sky.

```
$ gaiasky -e
```

The external view contains a copy of the same frame buffer rendered in the main view. The scene is not re-rendered (for now), so increasing the size of the external window won’t increase its base resolution. The original aspect ratio is maintained in the external view to avoid stretching the image.

**Hint:** Enable the external view at launch with the flag `-e` or `--externalview`.

![External view using planetarium mode](image.png)

**Fig. 9:** External view using planetarium mode

### 1.4.13 Connecting Gaia Sky instances

Gaia Sky offers a method to connect different instances together so that their internal state is synchronized. The model uses a **master-slave** scenario, where one (and only one) instance acts as a master and one or more instances act as
slaves, getting their internal states updated over a network. The user interacts with the master and all slaves are updated accordingly.

This section describes only how to configure the master and the slave instances in order to connect them together. This method is used to provide multi-projector rendering support (i.e. planetarium domes), but extra steps are needed in order to configure the orientation, distortion warp and blend settings for each slave instance.

**Hint:** Multi-projector configuration is covered in the “Planetarium mode multi-projector setup” section.

### Configuration

The configuration is easy and painless. You will need to launch each instance of Gaia Sky using a different configuration file (global.properties). You can run Gaia Sky with a specific configuration file by using the -p or --properties command line flags:

```sh
$ gaiasky -p ~/.config/gaiasky/global.master.properties
```

The next sections explain how to configure the master and the slaves.

#### Configuration: slave instance(s)

You can have as many slave instances as you want, but here we’ll explain the process of setting up two slaves.

1. Copy the current global.properties file in your config folder (see folders) into global.slave0.properties. The name is irrelevant, but choose something meaningful. Repeat with global.slave1.properties.

2. Set the property program.net.slave=true in each file and make sure that program.net.master is set to false.

3. Set the desired port to listen to in program.restport. For example, to use the port 13900 just set the property program.restport=13900. Use a different port for each slave (i.e. slave 0 listens to 13900, slave 1 listens to 13901, etc.).

For example, to set up a slave in port 13900, make sure that the following lines are in its configuration file:

```properties
program.net.master=false
program.net.slave=true
program.restport=13900
```

The slave instances should be launched before the master. Launch the slave(s) with:

```sh
$ # Launch slave 0
$ gaiasky -p /path/to/global.slave0.properties
$ # Launch slave 1
$ gaiasky -p /path/to/global.slave1.properties
```

Once the slave(s) have been launched, you can verify that the API is working by visiting http://localhost:13900/api/help with your browser. Modify the port with whatever port you are using.

**Hint:** Only the master is starting the scripting server. The slaves are automatically forbidden to do so!
**Configuration: master instance**

Copy the current `global.properties` file into `global.master.properties` and edit the following lines.

1. Set the property `program.net.master=true` and make sure that `program.net.slave` is set to `false`.

2. Add the locations of all desired slaves under the properties `program.net.master.slaves.n=URL`, where `n` is an integer that goes from 0 to the number of slaves - 1.

For example, in order to connect the master with two slaves, both running locally (`localhost`) on ports 13900 and 13901, add the following to the `global.master.properties` file:

```plaintext
program.net.master=true
program.net.slave=false
program.net.master.slave.0=http://localhost:13900/api/
program.net.master.slave.1=http://localhost:13901/api/
```

Then, just launch the master *(after the slaves are running!)*:

```
$ gaiasky -p /path/to/global.master.properties
```

**Caveats**

Even though this offers a very flexible system to connect several instances of Gaia Sky together, each instance is a fully-fledged application with its own copy of the scene graph and the data structures. This means that, if you run them locally, the data and scene graph will be replicated several times in memory, possibly consuming lots of gigabytes. Handle it with care.

**1.4.14 Camrecorder**

Gaia Sky offers the possibility to record camera paths out of the box and later play them. These camera paths are saved in a `.gsc` (for Gaia Sky Camera) file in `$GS_DATA/camera` (see folders).

**Camera path file format**

The format of the file is pretty straightforward. It consists of a `.gsc` file with white spaces as delimiters, each row containing the state of the camera and the time for a given frame. The state of the camera consists of 9 double-precision floating point numbers, 3 for the position and 3 for the direction vector and 3 for the up vector.

The reference system used is explained in the *Internal reference system* section. The units are $1 \times 10^{-9} m$.

The format of each row is as follows:

- **long** - Time as defined by the `getTime()` function of `java.util.Date` (here).
- **double x3** - Position of the camera.
- **double x3** - Direction vector of the camera.
- **double x3** - Up vector of the camera.

**Recording camera paths**

Gaia Sky offers two possibilities as to how to record a camera path: real time recording and keyframes.
Real time recording

In order to **start recording** the camera path, click on the REC button next to the Camera section title in the GUI Controls window. The REC button will turn red, which indicates the camera is being recorded.

In order to **stop the recording** and write the file, click again on the red REC button. The button will turn grey and a notification will pop up indicating the location of the camera file. Camera files are by default saved in the $GS_DATA/camera directory (see folders).

**Hint:** Mind the FPS! The camera recording system stores the position of the camera for every frame! It is important that recording and playback are done with the same (stable) frame rate. To set the target recording frame rate, edit the “Target FPS” field in the camrecorder settings of the preferences window. That will make sure the camera path is using the right frame rate. In order to play back the camera file at the right frame rate, you can edit the “Maximum frame rate” input in the graphics settings of the preferences window.

Keyframe system

The keyframe system offers the possibility to create keyframes at specific positions from which the camera file will be generated. In order start creating a keyframed camera path, click on the REC button in the camera pane, controls window. A new window will pop up from which you’ll be able to create and manage the keyframes.

**Keyframe and camera files**

Keyframes can be saved and loaded to and from .gkf files. These files only contain the information on the keyframes themselves. Once the keyframes have been created, they can be exported to a .gsc camera path file. Both keyframe files and camera path files are stored by default in the $GS_DATA/camera folder (see folders).

**Creating and editing keyframes**

A graphical representation of keyframes is displayed in the 3D world (see screenshot above). Make sure that the visibility of the component Others is enabled. The yellow lines join the keyframe positioins with straight lines, whereas the green line shows the true camera path which will be generated.

Keyframes can be selected and dragged with the right mouse button. The currently selected keyframe is highlighted in the keyframes list and also in the scene, using a magenta color. Here are the basic controls:

- **RIGHT_MOUSE** – select keyframes (click) and move them around (drag).
- **Shift + RIGHT_MOUSE** – drag to rotate the keyframe orientation around the up vector (in blue).
- **Ctrl + RIGHT_MOUSE** – drag to rotate the keyframe orientation around the direction vector (in red).
- **Ctrl + Shift + RIGHT_MOUSE** – drag to rotate the keyframe orientation around the perpendicular to the up and the direction vector (not represented in the scene).

Additionally, some actions can be performed directly on the keyframes list in the keyframes window. You can edit both the keyframe name and time (as elapsed time since previous) by clicking on them.
• \( \Rightarrow \) – go to the keyframe. Puts the camera in the state specified by the keyframe
• \( \Rightarrow \) – mark the keyframe as seam. In case the spline interpolation method is chosen, this will break the spline path.
• \( \Rightarrow \) – add a new keyframe after this one, interpolating position, orientation and time with the next one.
• \( \Rightarrow \) – remove the keyframe.

At the bottom of the keyframes window, there are a few buttons useful to load, export and save keyframes projects.

• **Normalize times to constant speed** – recompute all keyframe times so that speed is constant. The total length and distance are unaltered.
• **Load keyframes file** – loads a new .gkf keyframes file.
• **Save keyframes to file** – saves the current project to a .gkf file in $GS_DATA/camera.
• **Export to camera path** – exports the current project to a camera path using the settings defined in the settings dialog. See next section.
• **Preferences** – see next section, Keyframes preferences.

**Keyframes preferences**

The **Preferences** button (lower right in the Keyframes window) opens a window which contains some settings related to the keyframes system:
• **Target FPS** – the target frame rate to use when generating the camera file from the keyframes.

• **Interpolation type** – used for generating the positions and/or the orientations. The time is always interpolated linearly to prevent unwanted speed-ups and slow-downs. Two types of interpolation are available:
  
  – **Catmull-Rom splines** – produce smoothed paths which hit every keyframe. In this mode, keyframes can be seams, meaning that the path is broken at that point.
  
  – **Linear interpolation** – keyframe positions are joined by straight lines. In this mode, the yellow and green lines above are the same.

### Playing camera paths

In order to play a camera file, click on the PLAY icon next to the REC icon. This will prompt a list of available camera files in the $GS\_DATA/camera folder (see folders).

You can also combine the camera file playback with the frame output system to save each frame to a JPEG image during playback. To do so, enable the **Activate frame output automatically** checkbox in the preferences dialog as described in the Camrecorder section.

### 1.4.15 Scripting

As of version 2.2.0, Gaia Sky uses Py4J instead of Jython as a mechanism to run Python scripts.

You can find lots of example scripts in the scripts folder of the project.

#### Quick start

Gaia Sky scripts must be run with the system Python interpreter. They connect to a gateway service offered by a running instance of Gaia Sky. As of version 2.1.8, Gaia Sky is **not** responsible anymore to run the scripts.

#### Requirements

In order to connect to the gateway server, you need a Python interpreter and the Py4J package. Find out how to install it here, or simply do, if you use pip:

```
$ pip install py4j
```

Note that you may need to call this with elevated privileges.

You may also use your distribution or operating system package manager to install Py4J. Please, refer to your distribution or operating system documentation for more information. Find more information on the library at the Py4J homepage.

#### Running a test script

Then, launch Gaia Sky, download this script, open a terminal window (PowerShell in Windows) and run:

```
$ python asteroids-tour.py
```
The location from which you run the script does not matter. If all goes well, you should see no crashes anywhere, and Gaia Sky should be showing a nice tour of the asteroids in the DR2 catalog.

Have a look at the script. All lines which start with `gs.` are API calls which call methods in the Gaia Sky gateway server. What are API calls, you ask? See next section.

**The scripting API**

The scripting API is a set of methods which may be called from Python scripts to interact with Gaia Sky. The available methods differ depending on the version of Gaia Sky.

**API documentation**

The only up-to-date API documentation for each version is in the interface header files themselves. Below is a list of links to the different APIs.

- API Gaia Sky master (development branch)
- API Gaia Sky 2.2.4-1
- API Gaia Sky 2.2.4
- API Gaia Sky 2.2.3
- 2.2.2, 2.2.1, 2.2.0, 2.1.7, 2.1.6, 2.1.5, 2.1.4, 2.1.3, 2.1.2, 2.1.1, 2.1.0, 2.0.3, 2.0.2, 2.0.1, 2.0.0

**Writing scripts for Gaia Sky**

Gaia Sky uses the single-threaded model of Py4J. In order to connect to Gaia Sky from Python, import `ClientServer` and `JavaParameters`, and then create a gateway and get its entry point. The entry point is the object you can use to call API methods on. Since Gaia Sky uses a server per script, the gateway must be shut down at the end of the script so that the Python program can terminate correctly and Gaia Sky can create a new server to deal with further scripts listening to the Py4J port.
from py4j.clientserver import ClientServer, JavaParameters

gateway = ClientServer(java_parameters=JavaParameters(auto_convert=True))
gs = gateway.entry_point

[...]
gateway.shutdown()

The `JavaParameters(auto_convert=True)` is not strictly necessary, but if you don’t use it you need to convert Python lists to Java arrays yourself before calling the API.

Now, we can start calling API methods on `gs`.

```python
# Disable input
gs.disableInput()
gs.cameraStop()
gs.minimizeInterfaceWindow()

# Welcome
gs.setHeadlineMessage("Welcome to the Gaia Sky")
gs.setSubheadMessage("Explore Gaia, the Solar System and the whole Galaxy!")
[...]
```

Find lots of example scripts here.

**Logging to Gaia Sky and Python**

When printing messages, you can either log to Gaia Sky or print to the standard output of the terminal where Python runs:

```python
gs.print("This goes to the Gaia Sky log")
print("This goes to the Python output")
```

In order to log messages to both outputs, you can define a function which takes a string and prints it out to both sides:

```python
def pprint(text):
    gs.print(text)
    print(text)

pprint("Hey, this is printed in both Gaia Sky AND Python!")
```

**Method and attribute access**

Py4J allows accessing public class methods but not public attributes. In case you get objects from Gaia Sky, you can’t directly call public attributes, but need to access them via public methods:

```python
# Get the Mars model object
body = gs.getObject("Mars")
# Get spherical coordinates
radec = body.getPosSph()

# DO NOT do this, it crashes!
gs.print("RA/DEC: %f / %f" % (radec.x, radec.y))
```
# DO THIS instead

gs.print("RA/DEC: \%f / \%f" % (radec.x(), radec.y()))

## Strict parameter types

Please, be strict with the parameter types. Use floats when the method signature has floats and integers when it has integers. The scripting interface still tries to perform conversions under the hood but it is better to do it right from the beginning. For example, for the API method:

double[] galacticToInternalCartesian(double l, double b, double r);

may not work if called like this from Python:

gs.galacticToInternalCartesian(10, 43.5, 2)

Note that the first and third parameters are integers rather than floating-point numbers. Call it like this instead:

gs.galacticToInternalCartesian(10.0, 43.5, 2.0)

## Loading datasets from scripts

Gaia Sky supports data loading from scripts using the STIL data provider. It is really easy to load a VOTable file from a script:

```python
from py4j.clientserver import ClientServer, JavaParameters
gateway = ClientServer(java_parameters=JavaParameters(auto_convert=True))
gs = gateway.entry_point

# Load dataset
gs.loadDataset("dataset-name", "/path/to/dataset.vot")
# Async insertion, let's make sure the data is available
gs.sleep(2)

# Now we can play around with it
gs.hideDataset("dataset-name")

# Show it again
gs.showDataset("dataset-name")

# Shutdown
gateway.shutdown()
```

Find an example of data loading in a script [here](#).

## Synchronizing with the main loop

Sometimes, when updating animations or creating camera paths, it is necessary to sync the execution of scripts with the thread which runs the main loop (main thread). However, the scripting engine runs scripts in separate threads asynchronously, making it a non-obvious task to achieve this synchronization. In order to fix this, a new mechanism has been added in Gaia Sky 2.0.3. Now, runnables can be parked so that they run at the end of the update-render
processing of each loop cycle. A runnable is a class which extends `java.lang.Runnable`, and implements a very simple `public void run()` method.

Runnables can be **posted**, meaning that they are run only once at the end of the current cycle, or **parked**, meaning that they run until they stop or they are unparked. Parked runnables must provide a name identifier in order to be later accessed and unparked.

Let’s see an example of how to implement a frame counter in Python using `py4j`:

```python
from py4j.clientserver import ClientServer, JavaParameters, PythonParameters

class FrameCounterRunnable(object):
    def __init__(self):
        self.n = 0

    def run(self):
        self.n = self.n + 1
        if self.n % 30 == 0:
            gs.print("Number of frames: %d" % self.n)

class Java:
    implements = ["java.lang.Runnable"]

gateway = ClientServer(java_parameters=JavaParameters(auto_convert=True),
        python_parameters=PythonParameters())
gs = gateway.entry_point

# We park a runnable which counts the frames and prints the current number
# of frames every 30 of them
gs.parkRunnable("frame_counter", FrameCounterRunnable())
gs.sleep(15.0)

# We unpark the frame counter
gs.unparkRunnable("frame_counter")
gateway.shutdown()
```

In this example, we park a runnable which counts frames for 15 seconds. Note that here we need to pass a `PythonParameters` instance to the `ClientServer` constructor.

A more useful example can be found [here](#). In this one, a polyline is created between the Earth and the Moon. Then, a parked runnable is used to update the line points with the new positions of the bodies. Finally, time is started so that the bodies start moving and the line positions are updated correctly and in sync with the main thread.

**More examples**

As we said, you can find more examples in the `scripts` folder in the repository.

**Running and debugging scripts**

In order to run scripts, you need a Python interpreter with the `python-py4j` module installed in your system. Load up Gaia Sky, open a new terminal window and run your script:
$ python script.py

Please, note that Gaia Sky needs to be running before the script is started for the connection to succeed.

To debug a script in the terminal using pudb run this:

$ python -m pudb script.py

### 1.4.16 Capturing videos

In order to capture videos there are at least two options which differ significantly.

**Frame output system + ffmpeg**

The frame output system enables automatic saving of every frame to an image file to disk with an arbitrary resolution and a user-defined frame rate. The image files can later be encoded into a video using video encoder software such as ffmpeg.

**Note:** Use F6 to activate the frame output mode and start saving each frame as an image. Use F6 again to deactivate it. When the frame output mode is active, the icon 📺 is displayed at the top-right corner of the screen.

When the frame output system is active, each frame is saved as a JPG or PNG image to disk. Refer to the Frame output section to learn how to configure the frame output system.

Once you have the image frames you can encode a video using a ffmpeg preset (slow, veryslow, fast, etc.) with the following command:

```bash
$ ffmpeg -framerate 60 -start_number [start_img_num] -i [prefix]%05d.jpg -vframes \n  → [num_images] -s 1280x720 -c:v libx264 -preset [slower|veryslow|placebo] -r 60 [out_ \n  → _video_filename].mp4
```

Please note that if you don’t want scaling, the --framerate input framerate, -r output framerate and -s resolution settings must match the settings defined in the frame output system preferences in Gaia Sky. You can also use a constant rate factor -crf setting:

```bash
$ ffmpeg -framerate 60 -start_number [start_img_num] -i [prefix]%05d.jpg -vframes \n  → [num_images] -s 1280x720 -c:v libx264 -pix_fmt yuv420p -crf 23 -r 60 [out_video_ \n  → filename].mp4
```

You need to obviously change the prefix and start number, if any, choose the right resolution, frame rate and preset and modify the output format if you need to.

ffmpeg is quite a complex command which provides a lot of options, so for more information please refer to the official ffmpeg documentation. Also, here is a good resource on encoding videos from image sequences with ffmpeg.

**OpenGL/Screen recorders**

There are several available options to record the screen or OpenGL context, in all systems. Below are some of these listed. These methods, however, will only record the scene as it is displayed in the screen and are limited to its window resolution.

1.4. Usage, data and options 47
Linux

- **OBS Studio** - Amazing open source capturing and streaming solution.
- **glc/glcs** - Command-line interface applications. The documentation and user guides can be found in this [wiki](#).
- **Simple Screen Recorder** - The name says it all.
- **Gamecaster** - Front end to glc.
- **Soul Capture** - Front end to glc.

Windows

- **OBS Studio** - Amazing open source capturing and streaming solution.
- **FRAPS** - 3rd party Direct3D and OpenGL recording software.
- **NVIDIA Shadowplay** - Only for NVIDIA cards.

1.4.17 Taking screenshots

Gaia Sky has an in-built screenshot capturing feature. To take a screenshot press **F5** any time during the execution of the program. By default, screenshots are saved in the `$GS_DATA/screenshots` (see [folders](#)) folder. The screenshots are in **PNG** format with high quality settings, so they can grow quite big if the resolution is large.

**Hint:** Take a screenshot with **F5**.

Screenshot modes

- **Simple mode** - This mode saves the current screen buffer to a file. It captures also the GUI and it does so at the current display resolution.

- **Advanced mode** - This mode renders the current scene to an off-screen buffer with an arbitrary resolution. The resolution can be configured in the config dialog, **Screenshots** tab. The advanced mode will **NOT** capture the GUI or any additional elements that are not part of the scene.

1.4.18 SAMP integration

As of commit 4d0d133, or version 2.0.0, Gaia Sky supports interoperability via **SAMP**. However, due to the nature of Gaia Sky, not all functions are yet implemented and not all types of data tables are supported.

Since Gaia Sky only displays 3D positional information there are a few restrictions as to how the integration with SAMP is implemented.

The current implementation only allows using Gaia Sky as a SAMP client. This means that when Gaia Sky is started, it automatically looks for a preexisting SAMP hub. If it is found, then a connection is attempted. If it is not found, then Gaia Sky will attempt further connections at regular intervals of 10 seconds. Gaia Sky will never run its own SAMP hub, so the user always needs a SAMP-hub application (Topcat, Aladin, etc.) to use the interoperability that SAMP offers.

Also, the only supported format in SAMP is VOTable through the STIL data provider described below.
STIL data provider

Gaia Sky supports the loading of data in VOTable, CSV, ASCII, etc. using the STIL library. It tries to make educated guesses using UCDs and column names to attribute semantics to columns. Here is what this provider can work with:

IDs

Columns with the ucd `meta.id` are recognized as generic identifiers. Otherwise, the actual matching is done by column name. The following are recognized:

- **id** – Generic ID
- **hip** – HIP number
- **source_id** – Gaia source ID
- **tycho2_id** – Tycho-2 ID

Names

Names are taken from the columns `name`, `proper`, `proper_name`, `common_name` and `designation`. The loader supports multiple names in a single value. The connecting character used is `|`, so that if multiple names are to be loaded, they must be in a column with one of the above names and the format `name-1|name-2|...|name-n`.

Positions

For the positional data, Gaia Sky will look for spherical and cartesian coordinates. In the case of spherical coordinates, the following are supported:

- **Equatorial**: `pos.eq.ra`, `pos.eq.dec`
- **Galactic**: `pos.galactic.lon`, `pos.galactic.lat`
- **Ecliptic**: `pos.ecliptic.lon`, `pos.ecliptic.lat`

To work out the distance, it looks for `pos.parallax` and `pos.distance`. If either of those are found, they are used. Otherwise, a default parallax of 0.04 mas is used. With respect to cartesian coordinates, it recognizes `pos.cartesian.x|y|z`, and they are interpreted in the equatorial system by default. If no UCDs are available, only equatorial coordinates (ra, dec) are supported, and they are looked up using the column names.

Proper motions and radial velocities

Proper motions are supported using only the UCDs `pm.eq.ra` and `pm.eq.dec`. Otherwise, the following column names are checked, assuming the units to be in mas/yr.

- **RA**: `pmra`, `pmalpha`, `pm_ra`
- **DEC**: `pmdec`, `pmdelta`, `pm_dec`, `pm_de`

Radial velocities are supported through the UCD `dopplerVeloc` and through the column names `radvel` and `radial_velocity`.
Magnitudes

Magnitudes are supported using the phot.mag or phot.mag;stat.mean UCDs. Otherwise, they are discovered using the column names mag, bmag, gmag, phot_g_mean_mag. If no magnitudes are found, the default value of 15 is used.

Colors

Colors are discovered using the phot.color UCD. If not present, the column names b_v, v_i, bp_rp, bp_g and g_rp are used, if present. If no color is discovered at all, the default value of 0.656 is used.

Others

Other physical quantities (mass, flux, T_eff, radius, etc.) are not yet supported via SAMP.

Implemented features

The following SAMP features are implemented:

- Load VOTable (table.load.votable) - The VOTable will be loaded into Gaia Sky if it adheres to the format above.
- Highlight row (table.highlight.row) - The row (object) is set as the new focus if the table it comes from is already loaded. Otherwise, Gaia Sky will not load the table lazily.
- Broadcast selection (table.highlight.row) - When a star of a table loaded via SAMP is selected, Gaia Sky broadcasts it as a row highlight, so that other clients may act on it.
- Point at sky (coord.pointAt.sky) - Puts camera in free mode and points it to the specific direction.
- Multi selection (table.select.rowList) - Gaia Sky does not have multiple selections so far, so only the first one is used right now.

Unimplemented features

The following SAMP functions are not yet implemented:

- table.load.* - Only VOTable supported.
- image.load.fits
- spectrum.load.ssa-generic
- client.env.get
- bibcode.load
- voresource.loadlist
- coverage.load.moc.fits
1.5 Additional information

1.5.1 Performance

The performance of the application may vary significantly depending on the characteristics of your system. This chapter describes what are the factors that have an impact in a greater or lesser degree in the performance of the Gaia Sky and explains how to tweak them. It is organised in two parts, namely GPU performance (graphics performance) and CPU performance.

Graphics performance

Refer to the Graphics performance chapter.

CPU performance

The CPU also plays an obvious role in updating the scene state (positions, orientations, etc.), managing the input and events, executing the scripts and calling and running the rendering subsystem, which streams all the texturing and geometric information to the GPU for rendering. This section describes what are the elements that can cause a major impact in CPU performance and explains how to tune them.

Multithreading

Gaia Sky uses background threads to index and update meta-information on the stars that are currently in view. The multithreading option controls the number of threads devoted to these indexing and updating tasks. If multithreading is disabled, only one background thread is used. Otherwise, it uses the defined number of threads in the setting.

Limiting FPS

As of version 2.0.1, Gaia Sky offers a way to limit the frames per second. This will ease the CPU of some work, especially if the max FPS is set to a value lower than 60. To do it, just edit the value in the preferences dialog, performance tab.

Draw distance (levels of detail)

These settings apply only when using a catalog with levels of detail like Gaia DR2+. We can configure whether we want Smooth transitions between the levels (fade-outs and fade-ins) and also the draw distance, which is represented by a range slider. The draw distance is a solid angle threshold against which we compare the octree nodes to determine their visibility.

Fig. 11: Draw distance slider in preferences dialog
Basically, the slider sets the view angle above which a particular octree node (axis aligned cubic volume) is marked as observed and thus its stars are processed and drawn.

- Set the knob to the right to lower the draw distance and increase performance.
- Set the knob to the left to higher the draw distance at the expense of performance.

Fig. 12: Octree and levels of detail. Image: Wikipedia.

Smooth transitions

This setting controls whether particles fade in and out depending on the octree view angle. This will prevent pop-ins when using a catalog backed by an octree but it will have a hit on performance due to the opacity information being sent to the GPU continuously. If smooth transitions are enabled, there is a fade-in between the draw distance angle and the draw distance angle + 0.4 rad.

1.5.2 Graphics performance

The Gaia Sky uses OpenGL to render advanced graphics and thus its performance may be affected significatively by your graphics card. Below you can find some tips to improve the performance of the application by tweaking or deactivating some graphical effects.

Graphics quality setting

Please see the Graphics quality section.
Antialiasing

Antialiasing is a term to refer to a number of techniques for reducing jagged edges, stairstep-like lines that should be smooth. It reduces the jagged appearance of lines and edges, but it also makes the image smoother. The result are generally better looking images, even though this depends on the resolution display device.

There are several groups of antialiasing techniques, some of them implemented in the Gaia Sky and available for you to choose from the graphics settings. They all come at a cost, which may vary depending on your system.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Antialiasing</td>
<td>No antialiasing</td>
<td>This has no cost since it does not apply any antialiasing technique.</td>
</tr>
<tr>
<td>FXAA</td>
<td>Post-processing</td>
<td>This has a mild performance cost and produces reasonably good results. If you have a good graphics card, this is super-fast.</td>
</tr>
<tr>
<td>NFAA</td>
<td>Post-processing</td>
<td>Based on the creation of a normal map to identify edges, this is slightly costlier than FXAA but it may produce better results in some devices.</td>
</tr>
</tbody>
</table>

Here are some sample images.
Some graphics drivers allow you to override the anti-aliasing settings of applications with some default configuration (usually MSAA or FXAA). You can also use this feature with the Gaia Sky.

Find more information on antialiasing in the *Antialiasing* section.

**Star brightness**

The *star brightness* setting has an effect on the graphics performance because it will cause more or less stars to be rendered as quads instead of points, which means multiplying the number of vertices to send to the GPU. Quads are basically flat polygons to which a texture is applied (in this case their appearance is controlled by a shader).

The star brightness can be increased or decreased from the *Star brightness* slider in the *Visual settings* section.

**Hint:** *Ctrl + d* - activate *debug mode* to get some information on how many stars are currently being rendered as points and quads as well as the frames per second, frame time and more.
Model detail

Some models (mainly spherical planets, planetoids, moons and asteroids) are automatically generated when the Gaia Sky is initialising and accept parameters which tell the loader how many vertices the model should have. These parameters are set in the json data files and can have an impact on devices with low-end graphics processors. Let’s see an example:

```json
{
   "model" : {
      "args" : [true],
      "type" : "sphere",
      "params" : {
         "quality" : 150,
         "diameter" : 1.0,
         "flip" : false
      },
      "texture" : {
         "base" : "data/tex/neptune.jpg",
      }
   }
}
```

The **quality** parameter specifies here the number of both vertical and horizontal divisions that the sphere will have.

Additionally, some other models, such as that of the Gaia spacecraft, come from a binary model file .g3db. These models are created using a 3D modelling software and then exported to either .g3db (bin) or .g3dj (JSON) using fbx-conv. You can create your own low-resolution models and export them to the right format. Then you just need to point the json data file to the right low-res model file. The attribute’s name is **model**.

```json
{
   "model" : {
      "args" : [true],
      "model" : "data/models/gaia/gaia.g3db"
   }
}
```

Bloom, lens flare and light glow

All post-processing algorithms (those algorithms that are applied to the image after it has been rendered) take a toll on the graphics card and can be disabled.

**Hint:** Disable the **light glow** effect for a significant performance boost in low-end graphics cards

- The **bloom** is not very taxing on the GPU.
- The **lens flare** effect is a bit harder on the GPU, but most modern cards should be able to handle it with no problems.
- The **light glow** effect is far more demanding, and disabling it can result in a significant performance gain in some GPUs. It samples the image around the principal light sources using a spiral pattern and applies a light glow texture which is rather large.

To disable these post-processing effects, find the controls in the UI window, as described in the [[lighting|User-interface#lighting]] section of the [[User interface]] chapter.
Labels

Object labels or names in the Gaia Sky are rendered using a special shader which implements distance field fonts. This means that labels look great at all distances but it is costlier than the regular method.

The label factor basically determines the stars for which a label will be rendered if labels are active. It is a real number between 1 and 5, and it will be used to scale the threshold point angle (which determines the boundary between rendering as points or as quads) to select whether a label should be rendered or not.

The label is rendered if the formula below yields true.

\[
\text{viewAngle} > \frac{\text{threshold\_angle\_point}}{\text{label\_factor}}
\]

Currently there is no GUI option for modifying the label factor, so you must directly edit the configuration file in the [[Scene properties|Configuration-files#scene-properties]] section of the [[Configuration files]] chapter.

1.5.3 Internal reference system

The internal cartesian reference system is described as follows: $XZ$ is the equatorial plane. $Z$ points towards the vernal equinox point, Aries (where $ra = 0^\circ$). $Y$ points towards the north celestial pole ($dec = 90^\circ$). $X$ is perpendicular to both $Z$ and $Y$ and points to $ra = 90^\circ$.

![Gaia Sky reference system](image)

Fig. 13: Gaia Sky reference system
All the positions and orientations of the entities in the scene are at some point converted to this reference system for representation. The same happens with the orientation sensor data in mobile devices.

### 1.5.4 Data: catalogues and formats

Gaia Sky needs to first load data in order to display it. The internal structure of these data is a scenegraph, which is basically a tree with nodes. The objects that are displayed in a scene are all nodes in this scene graph and are organized in a hierarchical manner depending on their geometrical and spatial relations.

**Hint:** The data nodes in the scene graph are of multiple natures and are loaded differently depending on their type. Here we can make the first big distinction in the data nodes depending on where they come from.

The different types of data are:

- **Catalogue data** – usually stars which come from a star catalogue. In this group we have two different approaches: single particles and particle groups. The TLDR version says that the single particles method is fundamentally slower and CPU-bound, while the particle groups method is faster and GPU-based. Therefore, single particles are deprecated.

- **Rest of data** – planets, orbits, constellations, grids and everything else qualifies for this category.

Data belonging to either group will be loaded differently into the Gaia Sky. The sections below describe the data format in detail:

#### General information on the data loading mechanisms

Gaia Sky implements a very flexible an open data mechanism. The data to be loaded is defined in a couple of keys in the global.properties configuration file, which is usually located in the $GS_CONFIG folder (see folders). The keys are:

- **data.json.catalog** – contains a comma-separated list of data files which point to the catalogs to load. These files have usually the data/catalog-*.json format.

- **data.json.objects** – contains a comma-separated list of data files which point to the files with the rest of the data. By default, only the data/data-main.json file is there.

Now, all the files in either properties have a very similar format, and nothing prevents you from putting catalogues into the objects file. However, the distinction is a semantic one, since the data defined in each file are fundamentally different. Also, Gaia Sky includes an option to choose the catalog(s) to load at startup using a GUI window (set property program.dataset.dialog to true to enable), and in this manner only the catalogue files can be modified.

Additionally, any file with the format autoload-*.json in the data folder will be automatically parsed and loaded using the default JSON loader.

#### catalog-*.json example files

```
{
    "name" : "TGAS+HYG (GPU)",
    "description" : "Gaia DR1 TGAS catalog, GPU version. About 1.5 million stars.",
    "data" : [{
        "loader": "gaiasky.data.JsonLoader",
        "files": [ "data/particles-tgas.json" ]
    }]
}
```
The format in all files is the same. There is a "data" property, which is a list of pairs containing [loader: files] correspondences. Each "loader" contains the classes that will load the list of files under the corresponding "files" property. Obviously, each loader needs to know how to load the provided files.

As of version 2.1.0, any descriptor file with the format autoload-*.json dropped into the data folder will be loaded by default without need to be referenced from any of the properties.

Fig. 14: Gaia Sky data loading diagram

The files are sent to the Scene Graph JSON Loader, which iterates on each loader-files pair in each file, instantiates the loader and uses it to load the files. All loaders need to adhere to a contract, defined in the interface ISceneGraphLoader – here –. The loadData() method of each loader must return a list of Scene Graph objects,
which is then added to a global list containing all the previously loaded files. At the end, we have a list with all the objects in the scene. This list is passed on to the Scene Graph instance, which constructs the scene graph tree structure which will contains the object model.

As we said, each loader will load a different kind of data; the JSONLoader—here—loads non-catalog data (planets, satellites, orbits, etc.), the STILDataProvider—here—loads VOTables, FITS, CSV and other files through the STIL library, ConstellationsLoader—here—and ConstellationsBoundariesLoader—here—load constellation data and constellation boundary data respectively and so on.

**Particle data**

Particle data refers to the loading of particles (stars, galaxies, etc.) where each gets an object in the internal scene graph model. This allows for selection, labeling, levels of detail, etc.

There are several off-the-shelf options to get local data in various formats into Gaia Sky.

In order to load **local data** there are a series of default options which can be combined. As described in the general data loading section, multiple catalogue loaders can be used at once. Each catalog loader will get a list of files to load. A description of the main local catalog loaders follows.

**Particle groups**

As of version 1.5.0, Gaia Sky offers a new data type, the particle group. Particle groups can be either point particles or stars (defined by star groups). Particle data are read from a file using a certain particle/star group provider implementation, and these data are sent to GPU memory where they reside. This approach allows for these objects to be composed of hundreds of thousands of particles and still have a minimal impact on performance.

Let’s see an example of the definition of one of such particle groups in the Oort cloud:

```json
{
    "name": "Oort cloud",
    "position": [0.0, 0.0, 0.0],
    "color": [0.9, 0.9, 0.9, 0.8],
    "size": 2.0,
    "labelcolor": [0.3, 0.6, 1.0, 1.0],
    "labelposition": [0.0484814, 0.0, 0.0484814],
    "ct": "Others",
    "fadein": [0.0004, 0.004],
    "fadeout": [0.1, 15.0],
    "profiledecay": 1.0,
    "parent": "Universe",
    "impl": "gaiasky.sceneograph.ParticleGroup",
    "provider": "gaiasky.data.group.PointDataProvider",
    "factor": 149.597871,
    "datafile": "data/oort/oort_10000particles.dat"
}
```

Let’s go over the attributes:

- **name** – The name of the particle group.
- **position** – The mean cartesian position (see internal reference system) in parsecs, used for sorting purposes and also for positioning the label. If this is not provided, the mean position of all the particles is used.
• **color** – The color of the particles as an *rgba* array.

• **size** – The size of the particles. In a non HiDPI screen, this is in pixel units. In HiDPI screens, the size will be scaled up to maintain the proportions.

• **labelcolor** – The color of the label as an *rgba* array.

• **labelposition** – The cartesian position (see *internal reference system*) of the label, in parsecs.

• **ct** – The ComponentType – here –. This is basically a string that will be matched to the entity type in ComponentType enum. Valid component types are Stars, Planets, Moons, Satellites, Atmospheres, Constellations, etc.

• **fadein** – The fade in interpolation distances, in parsecs. If this property is defined, there will be a fade-in effect applied to the particle group between the distance `fadein[0]` and the distance `fadein[1]`.

• **fadeout** – The fade out interpolation distances, in parsecs. If this property is defined, there will be a fade-in effect applied to the particle group between the distance `fadein[0]` and the distance `fadein[1]`.

• **profiledecay** – This attribute controls how particles are rendered. This is basically the opacity profile decay of each particle, as in \((1.0 - \text{dist})^{\text{profiledecay}}\), where dist is the distance from the center (center dist is 0, edge dist is 1).

• **parent** – The name of the parent object in the scenegraph.

• **impl** – The full name of the model class. This should always be `gaiasky.scenegraph.ParticleGroup`.

• **provider** – The full name of the data provider class. This must extend `gaiasky.data.group.IParticleGroupDataProvider` (see here).

• **factor** – A factor to be applied to each coordinate of each data point. If not specified, defaults to 1.

• **datafile** – The actual file with the data. It must be in a format that the data provider specified in `provider` knows how to load.

### Star groups

As of version 1.5.0, entire star catalogs can also be provided as a special type of particle groups: star groups. The stars in a star group will function very much like their single particles counterparts. They are rendered using the magnitude and color information, they are selectable and focusable, they can render labels and proper motions, and they get close-up detail quads. Since most of the rendering is GPU-based using VBOs, and there’s only one object in the scene graph for the whole star group, this method is much more performant than the single particles method. Also, to update some model information a background thread is spawned for every star group which sorts the particles in the background according to their current view angle.

To define a catalog containing a star group, we need to create a pointer and load it using the regular `JsonLoader`:

```javascript
{
   "name": "TGAS+HYG (GPU)",
   "description": "Gaia DR1 TGAS catalog, GPU version. About 1.5 million stars.",
   "data": [{
      "loader": "gaiasky.data.JsonLoader",
      "files": [ "data/tgas-pg.json" ]
   }]
}
```

The file `tgas-pg.json` contains a single object with the actual star group definition:
In this case, the data file, `tgashyg.bin`, is a binary file which contains java objects serialized. These can be loaded using the `SerializedDataProvider`. However, anyone can implement a new provider to load any other kind of catalog file by implementing the `IStarGroupDataProvider` interface.

Star groups can also be combined with octrees (levels of detail method) to allow for huge catalogs like DR2 (hundreds of millions of points). This option is still not implemented.

### Octree catalog loader

As of version 1.5.0, a new on-demand catalog loader exists, called Octree multfile loader. This is a version of the octree catalog loader specially designed for very large datasets. This version does not load everything at startup. It needs the catalog to be organised into several files, each one corresponding to a particular octree node. This is an option in the `OctreeGeneratorTest`. Back to the loader, it can pre-load files down to a certain depth level; the rest of the files will be loaded when needed and unloaded if necessary. This offers a convenient way in which the data is streamed from disk to the main memory as the user explores the dataset. It also results in a very fast program startup.

This loader is called `OctreeMultiFileLoader` and is implemented here.

Some discussion on memory issues and the streaming loader can be found here.

### STIL data provider

As of version v0.704 the Gaia Sky supports all formats supported by the STIL library. Since the data held by the formats supported by STIL is not of a unique nature, this catalog loader makes a series of assumptions. More information can be found in STIL data provider.

Particularly, it is possible to directly load a `VOTable` or a `csv` file into Gaia Sky using the `Open file` icon at the bottom of the control panel.

### Non-particle data: Planets, Moons, Asteroids, etc.

Most of the entities and celestial bodies that are not stars in the Gaia Sky scene are defined in a series of `json` files and are loaded using the `JsonLoader` here. The format is very flexible and loosely matches the underneath data
model, which is a scene graph tree.

**Top-level objects**

All objects in the json files must have at least the following 5 properties:

- **name**: The name of the object.
- **color**: The colour of the object. This will translate to the line colour in orbits, to the colour of the point for planets when they are far away and to the colour of the grid in grids.
- **ct**: The ComponentType – here –. This is basically a string that will be matched to the entity type in ComponentType enum. Valid component types are Stars, Planets, Moons, Satellites, Atmospheres, Constellations, etc.
- **impl**: The package and class name of the implementing class.
- **parent**: The name of the parent entity.

Additionally, different types of entities accept different additional parameters which are matched to the model using reflection. Here are some examples of these parameters:

- **size**: The size of the entity, usually the radius in km.
- **appmag**: The apparent magnitude.
- **absmag**: The absolute magnitude.

Below is an example of a simple entity, the equatorial grid:

```json
{
    "name" : "Equatorial grid",
    "color" : [1.0, 0.0, 0.0, 0.5],
    "size" : 1.2e12,
    "ct" : "Equatorial",
    "parent" : "Universe",
    "impl" : "gaiasky.scenegraph.Grid"
}
```

**Planets, moons, asteroids and all rigid bodies**

Planets, moons and asteroids all use the model object Planet - here -. This provides a series of utilities that make their json specifications look similar.

**Coordinates**

Within the coordinates object one specifies how to get the positional data of the entity given a time. This object contains a reference to the implementation class (which must implement IBodyCoordinates - here-) and the necessary parameters to initialize it. There are currently a bunch of implementations that can be of use:

- **OrbitLintCoordinates** – The coordinates of the object are linearly interpolated using the data of its orbit, which is defined in a separated entity. See the [[Orbits|Non-particle-data-loading#orbits]] section for more info. The name of the orbit entity must be given. For instance, the Hygieia moon uses orbit coordinates.
• **StaticCoordinates** – For entities that never move. A position is required. For instance, the Milky Way object uses static coordinates:

```
{
    "coordinates" : {
        "impl" : "gaiasky.util.coord.StaticCoordinates",
        "position" : [-2.169e17, -1.257e17, -1.898e16]
    }
}
```

• **AbstractVSOP87** – Used for the major planets, these coordinates implement the VSOP87 algorithms. Only the implementation is needed. For instance, the Earth uses these coordinates.

```
{
    "coordinates" : {
        "impl" : "gaiasky.util.coord.vsop87.EarthVSOP87"
    }
}
```

• **GaiaCoordinates** – Special coordinates for Gaia.

• **MoonAACoordinates** – Special coordinates for the moon using the algorithm described in the book Astronomical Algorithms by Jean Meeus.

**Rotation**

The rotation object describes, as you may imagine, the rigid rotation of the body in question. A rotation is described by the following parameters:

• **period** – The rotation period in hours.

• **axialtilt** – The axial tilt is the angle between the equatorial plane of the body and its orbital plane. In degrees.

• **inclination** – The inclination is the angle between the orbital plane and the ecliptic. In degrees.

• **ascendingnode** – The ascending node in degrees.

• **meridianangle** – The meridian angle in degrees.

For instance, the rotation of Mars:

```
{
    "rotation": {
        "period" : 24.622962156,
        "axialtilt" : 25.19,
        "inclination" : 1.850,
        "ascendingnode" : 47.68143,
        "meridianangle" : 176.630
    }
}
```
Model

This object describes the model which must be used to represent the entity. Models can have two origins:

- They may come from a **3D model file**. In this case, you just need to specify the file.

```json
{
  "model": {
    "args": [true],
    "model": "data/models/gaia/gaia.g3db"
  }
}
```

- They may be **generated on the fly**. In this case, you need to specify the type of model, a series of parameters and the material.

```json
{
  "model": {
    "args": [true],
    "type": "sphere",
    "params": {
      "quality": 180,
      "diameter": 1.0,
      "flip": false
    },
    "material": {
      "base": "data/tex/base/earth-day*.jpg",
      "specular": "data/tex/base/earth-specular*.jpg",
      "normal": "data/tex/base/earth-normal*.jpg",
      "night": "data/tex/base/earth-night*.jpg",
      "height": "data/tex/base/earth-height*.jpg",
      "heightScale": 8.12,
      "reflection": [1.0, 1.0, 0.0],
      "elevation": {
        "noiseType": "opensimplex",
        "noiseSize": 10.0,
        "noiseOctaves": [[1.0, 4.0, 8.0], [1.0, 0.25, 0.125]],
        "noisePower": 1.0
      }
    }
  }
}
```

- **type** – the type of model. Possible values are `sphere`, `disc`, `cylinder` and `ring`.
- **params** – parameters of the model. This depends on the type. The `quality` is the number of both horizontal and vertical divisions. The `diameter` is the diameter of the model and `flip` indicates whether the normals should be flipped to face outwards. The `ring` type also accepts `innerradius` and `outerradius`.
- **material** – properties of the material, such as textures, reflections, elevation, etc.
  - **base** – the diffuse texture to use.
  - **specular** – the specular map to produce specular reflections.
  - **normal** – normal map to produce extra detail in the lighting.
  - **night** – texture applied to the part of the model in the shade.
  - **height** – height map which will be represented with tessellation or parallax mapping (see `graphics configuration`) and whose scale is defined in `heightScale` (in Km). It can also contain the keyword "generate". In such case, a new subobject "elevation" is expected.
* noiseType – type of noise. Accepted values are opensimplex and perlin.
* noiseSize – extent of the noise map to generate.
* noiseOctaves – a 2xN matrix containing pairs of [frequency, amplitude] for the noise octaves.
* noisePower – exponent of a power operation on the generated noise (n=n^power).

– reflection – specifies an index or a color. If this is present, the default skymap will be used to generate reflections on the surface of the material. Hint: look up the Reflections object in Gaia Sky. It is defined in satellites.json.

**Atmosphere**

Planet atmospheres can also be defined using this object. The atmosphere object gets a number of physical quantities that are fed in the atmospheric scattering algorithm (Sean O’Neil, GPU Gems).

```json
{
    "atmosphere" : {
        "size" : 6600.0,
        "wavelengths" : [0.650, 0.570, 0.475],
        "m_Kr" : 0.0025,
        "m_Km" : 0.001,

        "params" : {
            "quality" : 180,
            "diameter" : 2.0,
            "flip" : true
        }
    }
}
```

**Orbits**

When we talk about orbits in this context we talk about orbit lines. In the Gaia Sky orbit lines may be created from two different sources. The sources are used by a class implementing the IOrbitDataProvider –here– interface, which is also specified in their orbit object.

- An orbit data file. In this case, the orbit data provider is OrbitFileDataProvider.
- The orbital elements, where the orbit data provider is OrbitalParametersProvider.

If the orbit is pre-sampled it comes from an orbit data file. In the Gaia Sky the orbits of all major planets are pre-sampled, as well as the orbit of Gaia. For instance, the orbit of Venus.

```json
{
    "name" : "Venus orbit",
    "color" : [1.0, 1.0, 1.0, 0.55],
    "ct" : "Orbits",

    "parent" : "Sol",
    "impl" : "gaiasky.scenegraph.Orbit",
    "provider" : "gaiasky.data.orbit.OrbitFileDataProvider",

    "orbit" : {
        "source" : "data/orb.VENUS.dat",
    }
}
```
If you prefer to define the orbit using the orbital elements, you need to specify these parameters in the orbit object. For example, the orbit of Phobos.

```json
{
    "name": "Phobos orbit",
    "color": [0.7, 0.7, 1.0, 0.4],
    "ct": "Orbits",

    "parent": "Mars",
    "impl": "gaiasky.scenegraph.Orbit",
    "provider": "gaiasky.data.orbit.OrbitalParametersProvider",

    "orbit": {
        "period": 0.31891023,
        "epoch": 2455198,
        "semimajoraxis": 9377.2,
        "eccentricity": 0.0151,
        "inclination": 1.082,
        "ascendingnode": 16.946,
        "argofpericenter": 157.116,
        "meananomaly": 241.138
    }
}
```

**Grids and other special objects**

There are a last family of objects which do not fall in any of the previous categories. These are grids and other objects such as the Milky Way (inner and outer parts). These objects usually have a special implementation and specific parameters, so they are a good example of how to implement new objects.

```json
{
    "name": "Galactic grid",
    "color": [0.3, 0.5, 1.0, 0.5],
    "size": 1.4e12,
    "ct": "Galactic",
    "transformName": "equatorialToGalactic",

    "parent": "Universe",
    "impl": "gaiasky.scenegraph.Grid"
}
```

For example, the grids accept a parameter `transformName`, which specifies the geometric transform to use. In the case of the galactic grid, we need to use the `equatorialToGalactic` transform to have the grid correctly positioned in the celestial sphere.

**Creating your own catalogue loaders**

If you want to load data into Gaia Sky, changes are that the STIL data provider can already do it. It supports VOTable, FITS, ASCII, CSV, etc. and it loads the data making educated guesses on the UCDs (if present) or on the column names.
If you still need to create your own loader, keep reading.

In order to create a loader for your catalogue, one only needs to provide an implementation to the `ISceneGraphLoader` interface.

```java
public interface ISceneGraphLoader {
    public List<? extends SceneGraphNode> loadData() throws FileNotFoundException;
    public void initialize(String[] files) throws RuntimeException;
}
```

The main method to implement is `List<? extends SceneGraphNode> loadData()` – here –, which must return a list of elements that extend `SceneGraphNode`.

But how do we know which file to load? You need to create a `catalog-*.json` file, add your loader there and create the properties you desire. Usually, there is a property called `files` which contains a list of files to load. Once you’ve done that, implement the `initialize(String[])` – here – method knowing that all the properties defined in the `catalog-*.json` file with your catalogue loader as a prefix will be passed in the `Properties` object without prefix.

Also, you will need to connect this new catalog file with the Gaia Sky configuration so that it is loaded at startup. To do so, locate your `global.properties` file (usually under `$GS_CONFIG`, see `folders`) and add your new file to the property `data.json.catalog`.

Add your implementing `jar` file to the `classpath` (usually putting it in the `lib/` folder should do the trick) and you are good to go.

Take a look at already implemented catalogue loaders such as the `OctreeCatalogLoader` – here – to see how it works.

**Loading data using scripts**

Data can also be loaded at any time from a Python script. See the scripting section for more info.

### 1.5.5 Data streaming: Levels of detail

This section discusses the Levels of detail (LOD) datasets (from Gaia DR2 on) where not all data fits into the CPU memory (RAM) and especially the GPU memory (VRAM). In order to solve the issue, Gaia Sky implements a LOD structure based on the spatial distribution of stars into an octree. The culling of the octree is determined using a draw distance setting, called `alpha`. Alpha is actually the minimum solid angle from the camera that an octant must have for it to be observed and its stars to be rendered. Larger `alpha` values lead to less octants being observed, and smaller `nu` values lead to more octants being observed.

Balancing the loading of data depends on several parameters:

- The maximum java heap memory (set to 4 Gb by default), let’s call it `maxheap`.
- The available graphics memory (VRAM, video ram). It depends on your graphics card. Let’s call it `VRAM`.
- The draw distance setting , `alpha`.
- The maximum number of loaded stars, ‘nu’. This is in the configuration file (`$GS_CONFIG/global.properties`) under the key `scene.octree.maxstars`. The default value is balancing the 4 Gb of `maxheap` and the default data set.

So basically, a low `alpha` (below 50-60 degrees) means lots of observed octants and lots of stars. Setting `alpha` very low causes Gaia Sky to try to load lots of data, eventually overflowing the heap space and creating an `OutOfMemoryError`. To mitigate that, one can also increase the `maxheap` setting (`gaiasky` script in the download package, `core/build.gradle`, `run` task if running from source. The JVM argument is called `-Xmx`. More info).
Finally, there is the maximum number of loaded stars, $nu$. This is a number is set according to the **maxheap** setting. When the number of loaded stars is larger than $nu$, the loaded octants that have been unobserved for the longest time will be unloaded and their memory structures will be freed (both in GPU and CPU). This poses a problem if the draw distance setting is set so that the observed octants at a single moment contain more stars than than $nu$. That is why high values for **alpha** are recommended. Usually, values between 60 and 80 are fine, depending on the dataset and the machine.

### 1.6 Gaia Sky VR

**Gaia Sky VR** is the VR version of Gaia Sky. It runs on multiple headsets and operating systems thanks to Valve’s **OpenVR**, also implemented by **OpenOVR**. It is developed in the framework of **ESA’s Gaia mission** to chart about 1 billion stars of our Galaxy.

#### 1.6.1 Running Gaia Sky VR

The Gaia Sky VR project is the Virtual Reality version of Gaia Sky. At the moment, only **OpenVR** is supported, but nothing prevents us from supporting other APIs (like the Kronos Group’s **OpenXR**) in the future if it makes sense. Our tests have only been carried out with the Oculus Rift CV1 headset in direct mode under Windows. Supporting Linux is a top priority for us, and the HTC Vive should work well under Linux, even though the state of **OpenVR** in the platform is a bit rough. We have reports indicating that the HTC Vive VR controllers’ mappings are not fully working.

Also, we want to point out that Linux support for the Oculus Rift was dropped for the CV1 and it is not expected to be continued any time soon, unfortunately.

Gaia Sky VR is heavily under development, and it is not guaranteed to work. Currently, no binaries are provided, but it can still be run by compiling the source. Just keep in mind that this is the development branch.

### Pre-requisites

The minimum system requirements for running Gaia Sky VR are as following:

<table>
<thead>
<tr>
<th>VR Headset</th>
<th>OpenVR-compatible (Oculus Rift, HTC Vive)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Linux (only supported headsets) / Windows 10</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel Core i5 3rd Generation or similar, 4 core or higher recommended</td>
</tr>
<tr>
<td>GPU</td>
<td>VR-capable GPU (GTX 970 or above)</td>
</tr>
<tr>
<td>Memory</td>
<td>8+ GB RAM</td>
</tr>
<tr>
<td>Hard drive</td>
<td>1 GB of free disk space (depending on datasets)</td>
</tr>
</tbody>
</table>

From now on, this guide will assume you aim at running Gaia Sky VR with the Oculus Rift in Windows, as that’s the only headset we currently have access to, which is unfortunately Windows-only. You can still run it on Linux if you have a compatible headset like the HTC Vive or the Valve Index.

You will need the following:

1. Download and install **Git for Windows** or **cygwin** and get used to the unix-like command line interface.
2. Follow the provided vendor instructions and install the Oculus app with the runtime. If using a SteamVR headset (HTC Vive/Pro, Valve Index, etc.), just get Steam and download SteamVR.
3. For the Oculus, you need a translation layer from OpenVR to LibOVR. You can either use SteamVR (slower) or OpenOVR (faster). We recommend using OpenOVR, as it is much simpler and faster.
1. **OpenOVR OpenComposite** - Download OpenOVR’s OpenComposite Launcher, launch it and select ‘Switch to OpenComposite’. That’s it.

2. **SteamVR** - Download and install Steam and then install SteamVR and launch it. The SteamVR runtime must be running alongside the Oculus Runtime for it to work.

4. OpenJDK 11+.

5. A setup VR-ready rig.

**Cloning the repository**

First, open the Git or Cygwin CLI and clone the Gitlab repository. Right now only the master branch contains the version which shares the codebase with the desktop application, but starting with 2.2.1, you’ll need to check out a tag to get a guaranteed working version. This should give you a working version:

```
$ git clone https://github.com/langurmonkey/gaiasky.git
$ cd gaiasky
$ git checkout master
```

**Running**

To run Gaia Sky VR, make sure that both the Oculus runtime and Steam VR are running. Then, run Gaia Sky through gradle. The first time it will pull lots of dependencies and compile the whole project, so it may take a while.

```
$ gradlew.bat core:runvr
```

**CLI arguments**

You can use the provided bypass script instead of gradle. Use the `--vr` command line argument to launch Gaia Sky in VR mode.

```
$ gaiasky --vr
```

Run `gaiasky -h` or `man gaiasky` to find out about how to launch Gaia Sky and what arguments are accepted.

**1.6.2 Getting the data**

The download manager should show up automatically at startup. If it does not, force it with the `--d` argument (or using `gradlew core:rund` if running from sources). Just select the data packs and catalogs that you want to download, press `Download now` and wait for the process to finish.

```
$ gaiasky --d
```

You can also download the data packs manually here.

**1.6.3 Controls**

On the Oculus Rift controller the mappings are as follow:

- **Joystick (move)** - move around.
- **Trigger** - hold it to select an object and enter focus mode.
• **Jystick (push)** - return to free mode.
• **A or X** - toggle visibility of labels
• **B or Y** - enable/disable on-screen information
• **A + B or X + Y** - show usage info on screen
• **Grip + Joystick (move)** - rotate around focus, only in focus mode

### 1.6.4 Common problems

• If you are using an Optimus-powered laptop, make sure that the `java.exe` you are using to run Gaia Sky VR is set up properly in the Nvidia Control Panel to use the discrete GPU.
• If you experience low frame rates with an Oculus headset, try using OpenOVR OpenComposite instead of SteamVR.
• If you don’t see the VR controllers, check the output log for a line that starts with `VRContext - WARN: Could not [...]` and attach it or the full log file to a [bug report](https://github.com/gaia-sky/GaiaSky/issues).
• Make sure you are using Java 11+.

### 1.6.5 More info

The project’s main README file is [here](https://github.com/gaia-sky/GaiaSky).

### 1.7 Tutorials and workshops

This page gathers a list of the tutorials and workshops organised around Gaia Sky.

#### 1.7.1 Video production workshop

**The DPAC meeting has been cancelled due to the Coronavirus.**

This page contains the workshop on video production with Gaia Sky given at the DPAC plenary meeting in Heidelberg on March 4, 2020.

The main aim of this workshop is to provide a birds eye view of the capabilities of Gaia Sky when it comes to producing high-quality videos.

*Estimated duration:* 1.5 hours

**Before starting...**

In order to follow the workshop it is recommended, albeit not required, to have a local installation of Gaia Sky so that you can explore and try out the teachings for yourself. In order to install Gaia Sky, follow the instructions in [Installation](https://github.com/gaia-sky/GaiaSky#installation).

When you start Gaia Sky for the first time you will be greeted with a data downloader window. Please download the following data packs:

• **default-data** - Contains the Solar System and some other basic data
• **catalog-dr2-verysmall** or **catalog-hipparcos-newred** - The star catalog
- catalog-nbg – Nearby Galaxies Catalog
- catalog-sdss-12 (small) or catalog-sdss-14 (large) – Sloan Digital Sky Survey catalog
- catalog-nebulae – Some nebulae images

Then click on ‘Start’.

**The User Interface of Gaia Sky**

The User Interface of Gaia Sky consists of basically two components: Keyboard shortcuts and a Graphical User Interface in the form of the controls window.

![Gaia Sky User Interface](image)

**Fig. 15:** Gaia Sky User Interface with the most useful functions

- The UI is described in detail in *User Interface*
- The most common key bindings are listed in *Controls*

**Loading datasets**

Gaia Sky supports the loading of additional datasets at runtime. Right now, only the VOTable and CSV formats are tested, even though others may also be found to work in some cases. Gaia Sky needs some metadata, in the form of UCDs or column names, in order to parse the data correctly. Refer to the *STIL data provider* section for more information on how to prepare your dataset for Gaia Sky.

The datasets loaded in Gaia Sky at a certain moment can be found in the *Datasets* pane of the controls window.

There are three main ways to load new datasets into Gaia Sky:

- The user interface button
- Through SAMP
- Via scripting (addressed later on in the workshop)
Loading a dataset from the UI – Go ahead and remove the star catalog (either DR2 dataset or Hipparcos) by clicking on the icon. Now, download a raw Hipparcos dataset from here, click on the icon and select the file to start the loading. In a few moments the Hipparcos new reduction dataset should be loaded into Gaia Sky.

Loading a dataset via SAMP – This section presupposes that Topcat is installed in the machine and that the user knows how to use it to connect to the VO to get some data. The following video demonstrates how to do this (mirror):

Camrecorder: recording and playing back camera files

Gaia Sky includes a camrecorder feature to make recording and playing back camera files very easy. This comes in handy if you want to showcase a certain itinerary through a dataset, for example.
Recording a camera path – The camrecorder will capture the camera state at every frame and save it into a .gsc (for Gaia Sky camera) file. You can start a recording by clicking on the icon in the camera pane of the controls window. Once the recording mode is active, the icon will turn red. Click on it again in order to stop recording and save the camera file to disk with an auto-generated file name (default location is $GS_DATA/camera (see folders).

Playing a camera path – In order to playback a previously recorded .gsc camera file, click on the icon and select the desired camera path. The recording will start immediately.

**Hint: Mind the FPS!** The camera recording system stores the position of the camera for every frame! It is important that recording and playback are done with the same (stable) frame rate. To set the target recording frame rate, edit the “Target FPS” field in the camrecorder settings of the preferences window. That will make sure the camera path is using the right frame rate. In order to play back the camera file at the right frame rate, you can edit the “Maximum frame rate” input in the graphics settings of the preferences window.

Fig. 18: Location of the controls of the camrecorder in Gaia Sky

More information on camera paths in Gaia Sky can be found in the camera paths section.

**Keyframe system**

The camrecorder offers an additional way to define camera paths based on keyframes. Essentially, the user defines the position and orientation of the camera at certain times and the system generates the camera path from these definitions. Gaia Sky incorporates a whole keyframe definition system, which is out of the scope of this tutorial.

As a very short preview, in order to bring up the keyframes window to start defining a camera path, click on the icon.

More information on the keyframe system can be found in the keyframe system subsection of the docs.
Frame output system

In order to create high-quality videos, Gaia Sky offers the possibility to export every single still frame to an image file. The resolution of these still frames can be set independently of the current screen resolution.

You can start the frame output system by pressing F6. Once active, the frame rate will go down (each frame is being saved to disk). The save location of the still frame images is, by default, $GS_DATA/frames/[prefix]_[num].jpg, where [prefix] is an arbitrary string that can be defined in the preferences. The save location, mode (simple or advanced), and the resolution can also be defined in the preferences.

![Configuration screen for the frame output system]

Fig. 19: The configuration screen for the frame output system

Once we have the still frame images, we can convert them to a video using ffmpeg or any other encoding software. Additional information on how to convert the still frames to a video can be found in the capturing videos section.

Scripting

This section includes a hands-on session inspecting pre-existing scripts and writing new ones to later run them on Gaia Sky.

More information on the scripting system can be found in the scripting section.

- Scripting API specification:
  - 2.2.4-1.
  - development branch.

- Interesting showcase scripts can be found here.

- Basic testing scripts can be found here.

1.8 Frequently Asked Questions

1.8.1 Q: What is the base-data package?

The base-data package is required for Gaia Sky to run and contains basically the Solar System data (textures, models, orbits and attitudes of planets, moons, satellites, etc.). You can’t run Gaia Sky without the base-data
1.8.2 Q: Why do you have two different download pages?

We list the most important downloads in the official webpage of Gaia Sky (here) for convenience. The server listing (here) provides access to current and old releases.

At the end of the day, if you use the download manager of Gaia Sky, you will never see any of these. If you want to download the data manually, you can do so using either page.

1.8.3 Q: Why so many DR2 catalogs?

We offer 9 different catalogs based on DR2. Only one should be used at a time, as they are different subsets of the same data, meaning that smaller DR2 catalogs are contained in larger DR2 catalogs. For example, the catalog dr2-default is contained in dr2-large. dr2-verysmall is contained in dr2-default, and so on. We offer so many to give the opportunity to explore the DR2 data to everyone. Even if you have a potato PC, you can still run Gaia Sky with the dr2-verysmall dataset, which only contains the very best stars in terms of parallax relative error. If you have a more capable machine, you can explore larger and larger slices and get more stars in.

1.8.4 Q: Gaia Sky crashes on start-up, what to do?

We need a log. If you are using version 2.2.0 or newer, or the development version on the master branch, crash reports are saved in ~/.local/share/gaiasky/crashreports (Linux) or ~/.gaiasky/crashreports (Windows and macOS). Files starting with gaiasky_crash_* are crash reports, while files starting with gaiasky_log_* are full logs. Send us the relevant files.

If you are using another version of Gaia Sky (2.1.7 or older), getting a log differs depending on your Operating System.

On Linux, just run Gaia Sky from the command line and copy the log.

```bash
$ gaiasky
```

On Windows, files named output.log and error.log should be created in the installation folder of Gaia Sky. Check if they exist and, if so, attach them to the bug report. Otherwise, just open Power Shell, navigate to the installation folder and run the gaiasky.cmd script. The log will be printed in the Power Shell window.

On macOS, open a Terminal window and write this:

```bash
$ cd /Applications/Gaia\ Sky.app/Contenst/java/app
$ chmod u+x ./gaiasky
$ ./gaiasky
```

This will launch Gaia Sky in the terminal. Copy the log and paste it in the bug report. Here is a video demonstrating how to do this on macOS.

Once you have a log, create a bug report here, attach the log, and we’ll get to it ASAP.

1.8.5 Q: I can’t see the elevation data on Earth or other planets

First, make sure you are using at least version 2.2.0. Then, make sure that your graphics card supports tessellation (OpenGL 4.x). Then, download the High-resolution texture pack using the download manager and select High or Ultra in graphics quality. This is not strictly necessary, but it is much better to use higher resolution data if possible.
1.8.6 **Q: Can I contribute?**

Yes. You can contribute translations (currently EN, DE, CA, FR, SK, ES are available) or code. Please, have a look at the contributing guidelines.

1.8.7 **Q: I like Gaia Sky so much, can I donate to contribute to the project?**

Thanks, but no.

1.9 **Javadoc**

You can browse the Gaia Sky javadoc here:

- Gaia Sky javadoc.

1.10 **Changelog**

- Comprehensive version history
- Detailed changelog
- Full commit history

1.11 **About**

1.11.1 **Contact**

If you have doubts or issues you can contact us using one of the following methods.

- Submit an issue to our bug tracking system.
- Drop us a line in tsagrista@ari.uni-heidelberg.de.

Do not forget to visit our Homepage@ARI.

1.11.2 **Author**

**Toni Sagristà Sellés** – tonisagrista.com

1.11.3 **Acknowledgements**

The most up to date list of acknowledgements is always in the ACKNOWLEDGEMENTS.md file.

Funding for the project is provided by the following agencies:

- ZAH
- DLR
- BMWi