# Gaia Sky Documentation

**Antoni Sagristà Sellés**

**Apr 22, 2024**

# CONTENTS


**1 Contents** **3**

1.1 Installation and running . . . 3

1.1.1 System requirements . . . 4

1.1.2 Download . . . 4

1.1.3 Installation procedure . . . 4

1.1.4 Run from source . . . 6

1.1.5 CLI arguments . . . 8

1.1.6 Packaging the software . . . 9

1.2 System Directories . . . 9

1.2.1 Datasets location . . . 10

1.2.2 Logs and crash reports . . . 10

1.3 Quick start guide . . . 10

1.3.1 Before starting... . . . 11

1.3.2 Welcome window . . . 11

1.3.3 Dataset manager . . . 11

1.3.4 Basic controls . . . 13

1.3.5 The user interface . . . 16

1.3.6 Camera modes . . . 20

1.3.7 Special render modes . . . 20

1.3.8 Type visibility . . . 21

1.3.9 Visual settings . . . 24

1.3.10 External datasets . . . 25

1.3.11 External information . . . 27

1.3.12 Scripting . . . 28

1.3.13 Camera paths . . . 30

1.3.14 Frame output mode . . . 31

1.3.15 Conclusion . . . 32

1.4 User manual . . . 32

1.4.1 Dataset manager . . . 32

1.4.2 Controls . . . 36

1.4.3 User interface . . . 47

1.4.4 Camera settings . . . 62

1.4.5 Search objects . . . 65

1.4.6 Camera info panel . . . 66

1.4.7 Object visiblity . . . 70

1.4.8 Datasets . . . 70

1.4.9 Bookmarks . . . 85

1.4.10 Location log . . . 89

1.4.11 System information . . . 89

1.4.12 Camera paths . . . 91

1.4.13 Settings and configuration . . . 98
1.4.14 Scripting . . . 110
1.4.15 Frames and screenshots . . . 121
1.4.16 Stereoscopic (3D) mode . . . 122
1.4.17 Planetarium mode . . . 124
1.4.18 Panorama mode . . . 129
1.4.19 Orthosphere view mode . . . 131
1.4.20 Eclipse representation . . . 132
1.4.21 Bounding shapes . . . 132
1.4.22 External views . . . 134
1.4.23 Connecting Gaia Sky instances . . . 134
1.4.24 REST API . . . 137
1.4.25 Capturing videos . . . 140
1.4.26 SAMP integration . . . 141
1.4.27 Procedural planetary surfaces . . . 142
1.4.28 System logs . . . 150
1.5 Advanced topics . . . 151
1.5.1 The configuration file . . . 151
1.5.2 Proxy configuration . . . 153
1.5.3 Performance . . . 156
1.5.4 Graphics performance . . . 159
1.5.5 Internal reference system . . . 164
1.5.6 Data format . . . 166
1.5.7 STIL data loader . . . 189
1.5.8 Star catalog formats . . . 191
1.5.9 Particle catalog formats . . . 197
1.5.10 Archetypes . . . 198
1.5.11 Components . . . 204
1.5.12 Star rendering . . . 220
1.5.13 Defining an extrasolar system . . . 221
1.5.14 Cubemaps . . . 234
1.5.15 Virtual Textures . . . 235
1.5.16 Mesh warping . . . 239
1.6 Gaia Sky VR . . . 239
1.6.1 System requirements . . . 240
1.6.2 Set-up . . . 240
1.6.3 Downloading datasets . . . 241
1.6.4 Controls . . . 241
1.6.5 Caveats . . . 241
1.6.6 Common problems . . . 241
1.7 Additional resources . . . 241
1.7.1 Video tutorials . . . 243
1.7.2 Presentations . . . 243
1.7.3 Workshop notes . . . 243
1.8 Frequently Asked Questions . . . 260
1.8.1 Q: What is the base-data package? . . . 260
1.8.2 Q: Why do you have two different download pages? . . . 260
1.8.3 Q: Why so many Gaia-DR catalogs? . . . 260
1.8.4 Q: Gaia Sky crashes at start-up, what to do? . . . 260
1.8.5 Q: I'm running out of memory, what to do? . . . 261
1.8.6 Q: I can't see the elevation data on Earth or other planets! . . . 261
1.8.7 Q: What is the internal reference system used in Gaia Sky? . . . 261
1.8.8 Q: Can I contribute? . . . 261
1.8.9 Q: I like Gaia Sky so much, can I donate to contribute to the project? . . . 261

1.9 Changelog . . . . . 262
1.10 About . . . . . 262
1.10.1 Contact . . . . . 262
1.10.2 Author . . . . . 262
1.10.3 Acknowledgements . . . . . 262
1.10.4 Stats . . . . . 262

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

You can find a PDF version of this documentation here.

Gaia Sky is (partially) described in the paper Gaia Sky: Navigating the Gaia Catalog.

CHAPTER

# ONE

# CONTENTS

## 1.1 Installation and running

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

**Contents**


- *Installation and running*
  - *System requirements*
  - *Download*
  - *Installation procedure*
    - *Linux*
      - *Flatpak*
      - *AppImage*
      - *Unix installer*
      - *DEB package*
      - *RPM package*
      - *AUR package*
    - *Windows*
    - *macOS*
    - *TAR.GZ*
  - *Run from source*
    - *Requirements*
    - *Getting the catalog data*
    - *Compiling and running*
  - *CLI arguments*
  - *Packaging the software*

### 1.1.1 System requirements

Here are the minimum requirements to run this software:

| | |
|---|---|
| **Operating system** | Linux / Windows 7+ / macOS, x86_64 (ARM CPUs unsupported) |
| **CPU** | Intel Core i5 3rd Gen. 4+ cores recommended |
| **GPU** | Support for OpenGL 3.3 (4.x recommended), 1 GB VRAM |
| **Memory** | 2-6 GB RAM (depends on loaded datasets) |
| **Hard drive** | 1 GB of free disk space (depends on downloaded datasets) |

### 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 procedure

Depending on your system and your personal preferences the installation procedure may vary. This section describes the installation and running process for the different operating systems and packages.

#### Linux

We provide 4 distro-agnostic packages:

- *Flatpak*.
- *AppImage*.
- *Unix installer*.
- *TAR.GZ package*.

We also offer 3 distro-specific packages:

- *DEB* – **Debian** and derivatives.
- *RPM* – **RedHat** and derivatives.
- *AUR* – **Arch Linux** and derivatives.

#### Flatpak

Install the Flatpak package with the following:

```
flatpak install flathub de.uni_heidelberg.zah.GaiaSky
```

Then, run with:

```
flatpak run de.uni_heidelberg.zah.GaiaSky
```

### AppImage

The AppImage does not need installation. Download the package, give it execute permissions if necessary, and run it.

```
wget https://gaia.ari.uni-heidelberg.de/gaiasky/releases/latest/gaiasky_$VERSION_x86_64.
↪appimage
chmod +x gaiasky_$VERSION_x86_64.appimage
./gaiasky_$VERSION_x86_64.appimage
```

### Unix installer

Download the package, give it execute permissions and run it to start the installation process. Then follow the on-screen instructions:

```
chmod +x gaiasky_linux_$VERSION.sh
./gaiasky_linux_$VERSION.sh
```

Once installed, you can simply run the `gaiasky` command, or use your favourite launcher to find and run it.

### DEB package

This is the package for Debian-based distros (**Debian**, **Ubuntu**, **Mint**, etc.). Download the `gaiasky_$VERSION.deb` file and run the following command. You need root privileges to install a DEB package in your system.

```
dpkg -i gaiasky_$VERSION.deb
```

This installs the application in the `/opt/gaiasky/` folder and creates the necessary shortcuts and `.desktop` files.

Once installed, you can simply run the `gaiasky` command, or use your favourite launcher to find and run it.

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

```
apt remove gaiasky
```

### RPM package

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 need root privileges to install an RPM package in your system.

```
rpm --install gaiasky_linux_$VERSION.rpm
```

This installs the application in the `/opt/gaiasky/` folder and creates the necessary shortcuts.

Once installed, you can simply run the `gaiasky` command, or use your favourite launcher to find and run it.

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

```
yum remove gaiasky-x86
```

#### AUR package

We also offer an Arch User Repository (AUR) package for **Arch Linux** and derivatives. Install one of gaiasky, gaiasky-git or gaiasky-appimage. For example, if you use paru:

```
paru -S gaiasky
```

Once installed, you can simply run the gaiasky command, or use your favourite launcher to find and run it.

#### 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 need to choose the directory where the application is to be installed.

In windows, this means clicking on Start and then browsing the start menu folder Gaia Sky. You can run the executable(s) for Gaia Sky and Gaia Sky VR from there. You can also navigate to the installation folder and run the gaiasky.cmd file from a command prompt or PowerShell.

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 drag-and-drop the **Gaia Sky.app** application to your /Applications directory in Finder. Once copied, it is safe to unmount the dmg volume.

To run it, double click on the **Gaia Sky.app** launcher in your applications directory.

Our dmg package is not signed by Apple, so it will be detected as coming from an 'unidentified developer'. You can still install it by following the procedure described in this page.

#### TAR.GZ

Download the package, and extract it wherever. Then, use either the gaiasky or gaiasky.cmd script to start the program. On a Unix system, do:

```
tar -xzvf gaiasky-$VERSION.tar.gz -C target/directory/
cd target/directory/gaiasky-$VERSION
./gaiasky
```

### 1.1.4 Run from source

#### Requirements

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

- Java Development Kit (JDK, version 17 or above should suffice, we recommend using the latest LTS available).
- 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://codeberg.org/gaiasky/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* (key: `default-data`) 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 :guilabel:`Dataset manager` in the data tab of the preferences window. More information on the download manager can be found in *Dataset manager*.

You can also download the data packs manually here.

### Compiling and running

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

```
./gradlew core:run
```

If you want to pass CLI arguments via gradle, just use the gradle `--args` argument (**`gradlew core:run --args='-vr'`**).

**Tip:** Gaia Sky checks that your Java version is compatible with it when you run the build. Skip this check by setting the `GS_JAVA_VERSION_CHECK` environment variable to *false* in the context of gradle:

```
export GS_JAVA_VERSION_CHECK=false
```

In order to pull the latest changes from the remote git repository:

```
git pull
```

On Windows, you need to open the Command Prompt or PowerShell and run:

```
.\gradlew.bat core:run
```

### 1.1.5 CLI arguments

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

```
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
  -i, --asciiart
    Add nice ascii art to --version information.
    Default: false
  -s, --skip-welcome
    Skip the welcome screen if possible (base-data package must be present).
    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 to create a VR
    context through OpenVR.
    Default: false
  -e, --externalview
    Create a window with a view of the scene and no UI.
    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.
    Default: false
  -d, --debug
    Launch in debug mode. Prints out debug information from Gaia Sky to the
    logs.
    Default: false
  -g, --gpudebug
    Activate OpenGL debug mode. Prints out debug information from OpenGL to
    the standard output.
    Default: false
  -l, --headless
    Use headless (windowless) mode, for servers.
    Default: false
  --safemode
    Activate safe graphics mode. This forces the creation of an OpenGL 3.2
    context, and disables float buffers and tessellation.
    Default: false
  --nosafemode
    Force deactivation of safe graphics mode. Warning: this bypasses
    internal checks and may break things! Useful to get rid of safe graphics
```

(continues on next page)

(continued from previous page)

```
    mode in the settings.
    Default: false
  --hdpimode
    The HDPI mode to use. Defines how HiDPI monitors are handled. Operating
    systems may have a per-monitor HiDPI scale setting. The operating system
    may report window width/height and mouse coordinates in a logical
    coordinate system at a lower resolution than the actual physical
    resolution. This setting allows you to specify whether you want to work
    in logical or raw pixel units.
    Default: Pixels
    Possible Values: [Logical, Pixels]
```

### 1.1.6 Packaging the software

Gaia Sky can be exported to be run as a standalone app. Right now, doing so is only supported from Linux. You need the utility **help2man** in your path to generate the man pages. Remember to restart the gradle daemon after installing it. Then run:

```
gradlew core:dist
```

This creates a new directory `releases/gaiasky-$VERSION` 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. Additionally, you need a certificate for signing the Windows packages in `$GS/assets/cert/cert.pfx`. Then, just run:

```
gradlew core:pack -PwinKeystorePassword=$PASSWORD
```

Where $PASSWORD is the password of the certificate. This command produces the different OS packages (EXE, DMG, DEB, RPM, etc.) of Gaia Sky and stores them in the `releases/packages-$VERSION` directory.

## 1.2 System Directories

In this documentation we refer to a few different directories that Gaia Sky uses to store data and configuration settings: `$GS_DATA`, `$GS_CONFIG`, and `$GS_CACHE`.

- `$GS_DATA` — contains some essential files and directories for Gaia Sky to run properly. For example:
  - `$GS_DATA/camera` — storage point for *camera path* and *keyframe* files.
  - `$GS_DATA/frames` — default save location for the *frame output mode*.
  - `$GS_DATA/screenshots` — default save location for *screenshots*.
  - `$GS_DATA/crashreports` — whenever Gaia Sky crashes, a crash report is stored at this location.
  - `$GS_DATA/log` — contains the full Gaia Sky log of the last session. Only the last session's log is kept.
  - `$GS_DATA/data` — also referred to as simply `$data`, this is the default dataset save location. All datasets are stored in this location by default (can be changed from the dataset manager).

- `$GS_CONFIG` — contains the *configuration files*, the *bookmarks*, and the *keyboard mappings file*.
- `$GS_CACHE` — contains cached files, like Wikipedia images.

The locations of `$GS_DATA`, `$GS_CONFIG` and `$GS_CACHE` depend on the operating system:

- **Linux** — as of Gaia Sky 2.2.0, the Linux release of Gaia Sky uses the XDG base directory specification.
  - `$GS_DATA` = `~/.local/share/gaiasky/`
  - `$GS_CONFIG` = `~/.config/gaiasky/`
  - `$GS_CACHE` = `~/.cache/gaiasky/`
- **Windows and macOS** — the `.gaiasky` directory in the user home directory for both locations, so:
  - `$GS_DATA` = `$GS_CONFIG` = `[User.Home]/.gaiasky/`
  - `[User.Home]` on Windows is typically in `C:\Users\[username]`.
  - `[User.Home]` on macOS is typically in `/Users/[username]`.
  - `$GS_CONFIG` = `$GS_DATA`
  - `$GS_CACHE` = `$GS_DATA/cache`

### 1.2.1 Datasets location

By default, Gaia Sky stores the downloaded datasets in the `$GS_DATA/data` directory. The location where the datasets are saved is referred to as `$data`. The actual location of `$data` is stored in the configuration file (key `data::location`) and can be changed in the dataset manager window at startup.

- `$data` = `$GS_DATA/data`

### 1.2.2 Logs and crash reports

For every Gaia Sky session a system log is stored in the directory `$GS_DATA/log`. Logs are overwritten with each new session, so only the last log is effectively available at any given time.

- `$GS_DATA/log/gaiasky_log_lastsession.txt` — full log of the last Gaia Sky session.

Crash reports are stored in `$GS_DATA/crashreports` whenever Gaia Sky crashes. If that happens, please, create a new issue in https://codeberg.org/gaiasky/gaiasky/issues, and attach the crash report. Additionally, also attach the session log.

- `$GS_DATA/crashreports/gaiasky_crash_[$DATE].txt` — crash reports.

## 1.3 Quick start guide

> **Tip:** This guide is designed to be followed with the latest version of Gaia Sky!

The main aim of this quick start guide is to provide a concise on-ramp to the Gaia Sky platform by describing its operation and most common features.

**Gaia Sky crash course:** see this companion web presentation for a visual introduction to Gaia Sky.

The topics covered in this guide are the following:

- Gaia Sky introduction:

  - *Dataset manager*.
  - *Controls*, *movement*, *selection*.
  - *User interface*.
  - *Camera operation and modes*.
  - *Render modes* (3D, planetarium, 360, re-projection).
  - *Object and type visibility*.
  - *Visual settings*.
  - *External datasets* (loading, filters, SAMP).
- *Scripting*:
  - *Scripting basics*.
  - *The API*.
  - *Showcase scripts*.
  - *Hands-on session*.
- *Camera paths*:
  - *Recording and playback*.
  - *Keyframes system*.
- *Still frame output mode*.
  - *Video from still frames* with `ffmpeg`.

### 1.3.1 Before starting...

In order to follow this guide it is strongly recommended to have a local installation of Gaia Sky. To install Gaia Sky, follow the instructions for your operating system in *the installation section*.

### 1.3.2 Welcome window

When we start up Gaia Sky, we are greeted with this view:

From here, we have access to the global preferences ( , bottom-right), the help window ( , bottom-right), fire up the *Dataset manager*, or *Start Gaia Sky*.

### 1.3.3 Dataset manager

The *Dataset manager* is used to download, update, delete and enable/disable datasets. It consists of two tabs:

- *Available for download* – contains datasets that are available to be downloaded and installed.
- *Installed* – contains the datasets currently available locally.


Fig. 1: The welcome screen in Gaia Sky contains buttons to start the program, open the dataset manager, access the settings and more.

The datasets are downloaded directly from our servers over an encrypted HTTPS connection, and SHA256 checksums are used to verify the integrity of downloaded files.

The first time we start Gaia Sky we need to download at least the Base data pack (dataset key: `default-data`) to be able to start the program. This is a **REQUIRED** step. The base data pack contains essential data like most of the Solar System (planets, moons, orbits, asteroids, etc.), the Milky Way, grids, constellations and other important objects.

We may download any dataset in the *Available for download* tab by clicking on the icon .

Feel free to explore the available datasets.

The *Installed* tab shows the datasets that we have already downloaded and are available locally.

- **Enable/disable** a dataset using the **checkbox** in the dataset pane. Enabled datasets are loaded when Gaia Sky starts.
- **Remove** a dataset by right clicking with your mouse on it and selecting *Remove*.

Now, close the dataset manager and *Start Gaia Sky*.

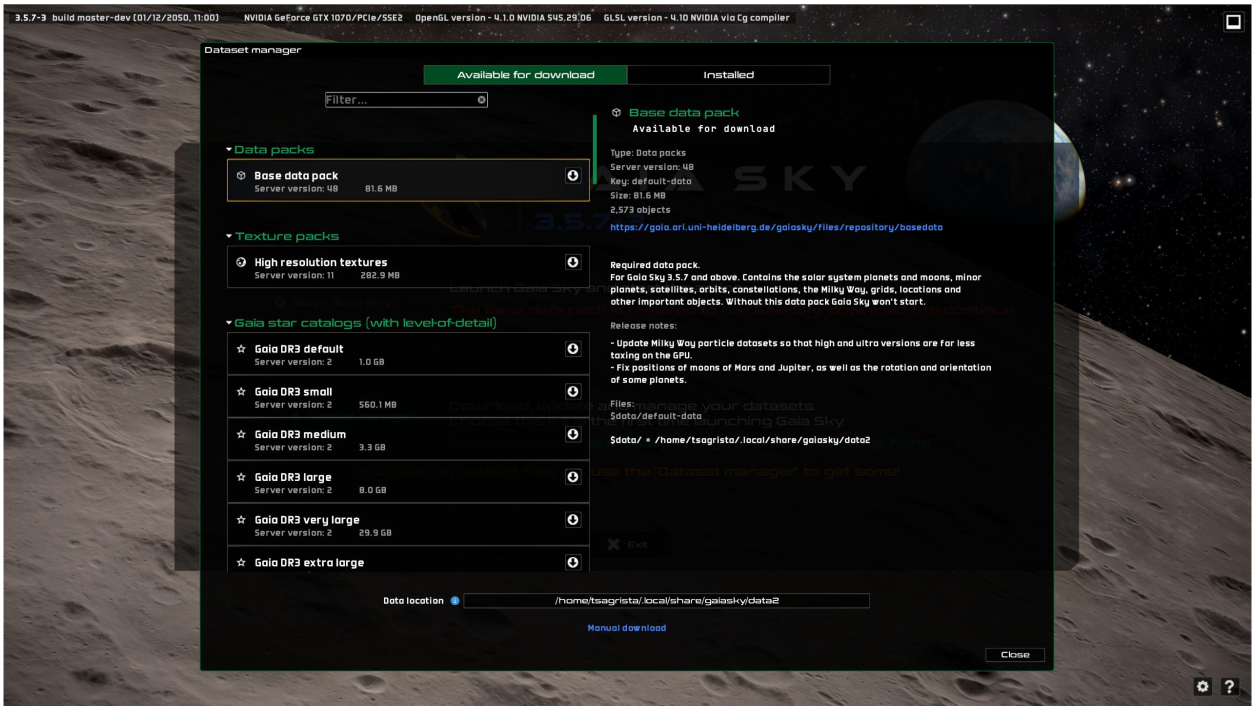



Fig. 2: The dataset manager with the base data pack selected.

### 1.3.4 Basic controls

When Gaia Sky is ready to go, we are presented with this screen:

In it, we can see a few things already:

- To the bottom-right, the *camera info panel* tells us that we are in focus mode, meaning that all our movement is relative to the focus object. The default focus of Gaia Sky is the Earth.
- To the top, the *quick info bar* tells us that our focus is the Earth, and that the closest object to our location is also the Earth. Additionally, we see that our home object is, again, the Earth.
- Anchored to the top-left, we see some buttons that give access to the *control panes*. If we click on one of these buttons, the respective pane opens. We will use them later.

#### Movement

But right now let's try some movement. In **focus mode** the camera orbits around the focus object, always pointing in the direction of the focus. **Try clicking and dragging with your left mouse button**. The camera should orbit around the Earth showing parts of the surface which were previously hidden. You may notice that the whole scene rotates. Now **try scrolling with your mouse wheel**. The camera moves either farther away from (scroll down) or closer to (scroll up) the Earth. We can always press and hold *z* to speed-up the camera considerably. This is useful to traverse long distances quickly.

Now, if we **click and drag with your right mouse button**, you can offset the focus object from the center, but your movement will still be relative to it.

You can also use your keyboard arrows ← ↑ → ↓ to orbit left or right around the focus object, or move closer to or away from it.

You can use *shift* with a **mouse drag** in order to roll the camera.

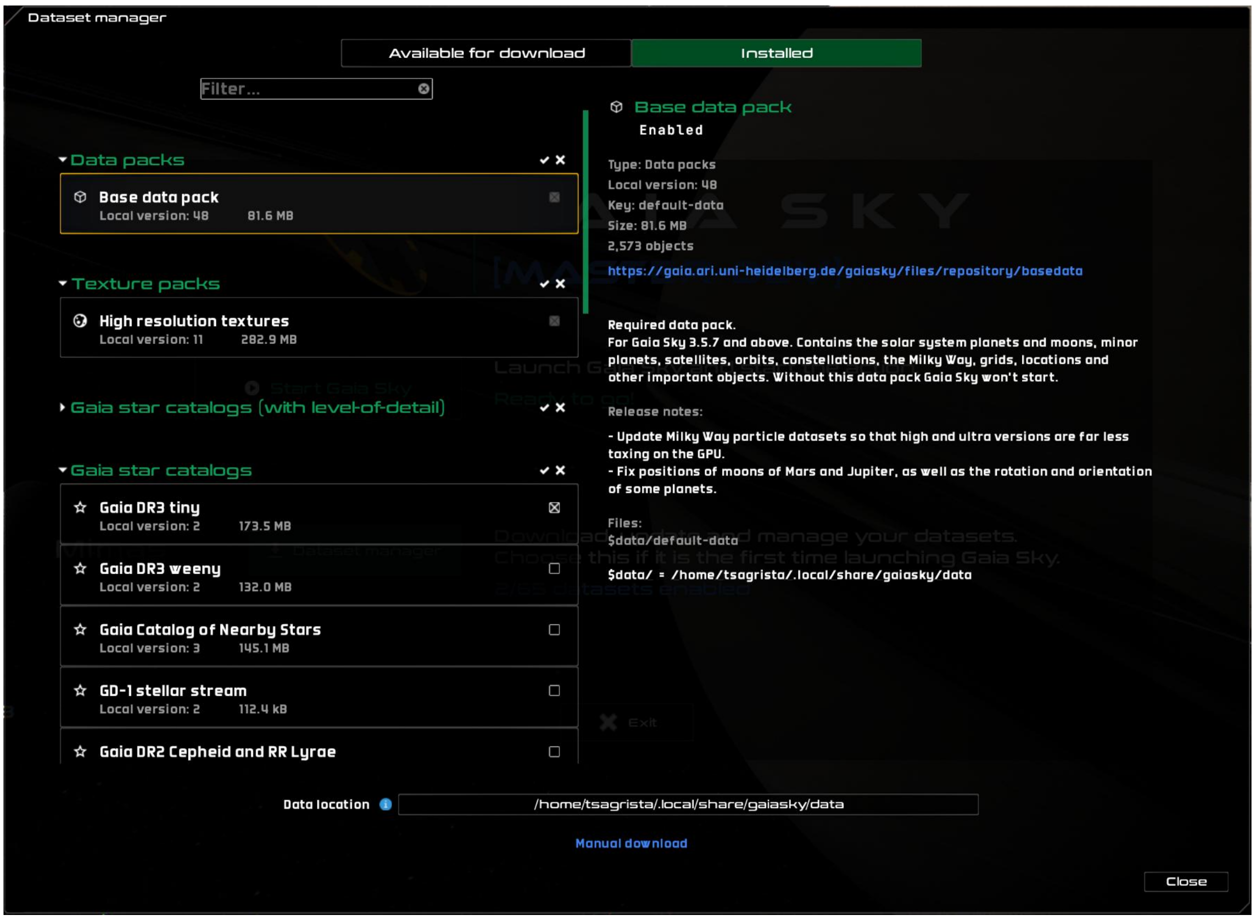



Fig. 3: Enable and disable datasets from the *Installed* tab.

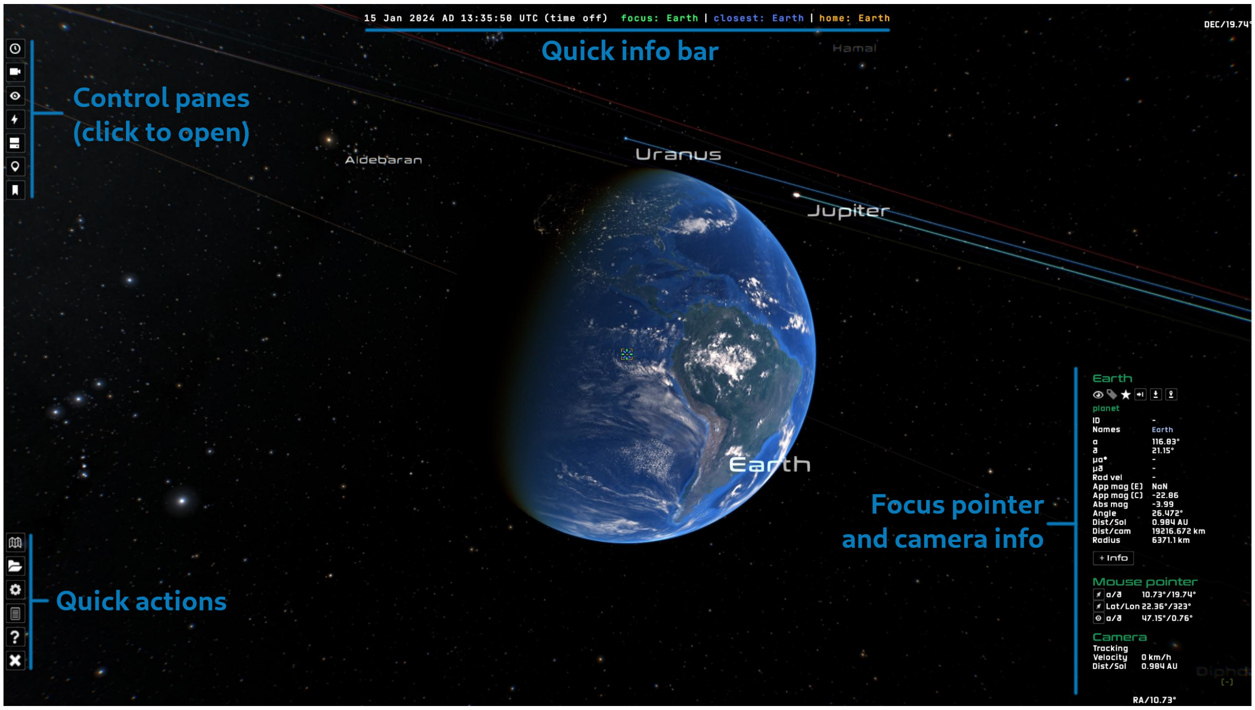



Fig. 4: Gaia Sky starts focussed on the Earth.

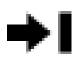
---

**Docs**

See the *controls section* of the user manual for more.

---

### Selection

You can change the focus by simply **double clicking** on any object on the scene. You can also press *f* to bring up the **search** dialog where you can look up objects by name. Try it now. Press *f* and type in "mars", without the quotes, and hit *esc*. You should see that the camera now points in the direction of Mars. To actually go to Mars simply scroll up until you reach it, or click on the  icon next to the name in the focus info panel. If you do so, Gaia Sky takes control of the camera and brings you to Mars.

If you want to move instantly to your current focus object, hit *ctrl* + *g*.

Any time, we can use the *Home* key to return back to Earth (in fact, we return to the home object, which is defined in the *configuration file*).

### 1.3.5 The user interface

The user interface of Gaia Sky consists of a few panes, buttons and windows. The most important of those are the **control panes**, accessible via a series of buttons anchored to the top-left.

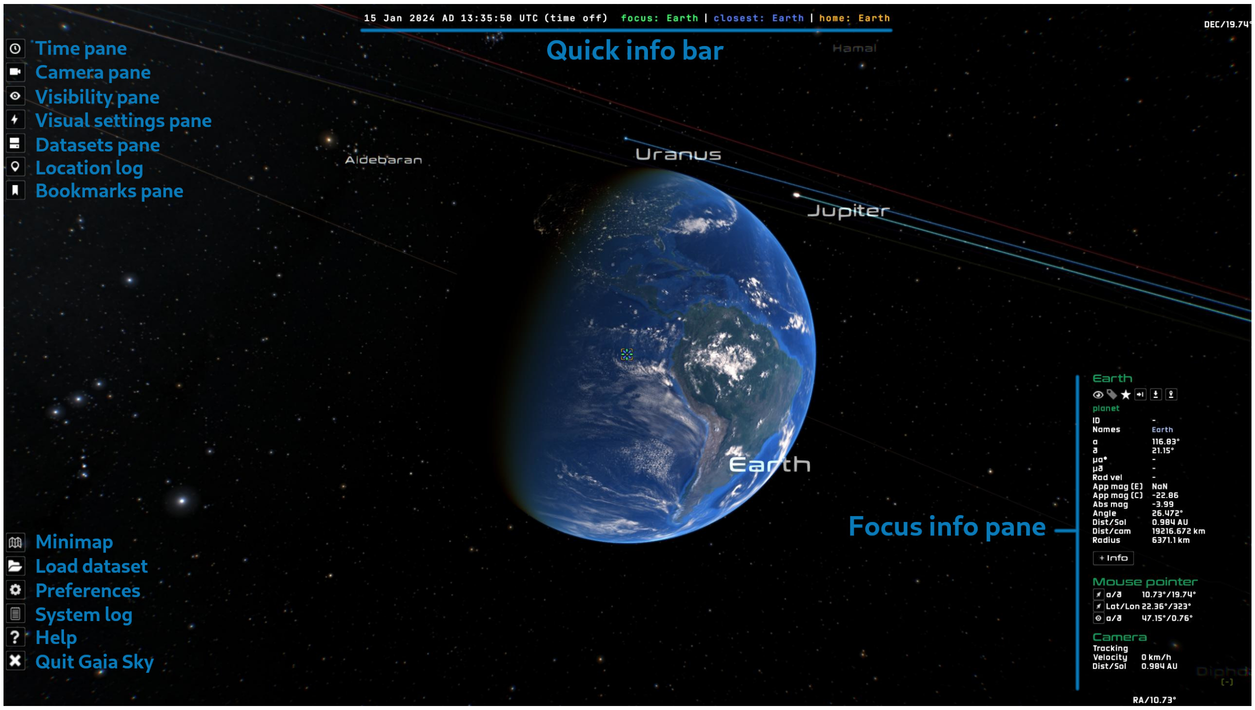



Fig. 5: Gaia Sky user interface with the most useful functions

**Docs**

See the *user interface section* of the user manual for more information.

#### Control panes

The control panes (previously called *control panel* in the old UI—it can still be used but is off by default) are made up of seven different panes:

- Time – shortcut: *t*.
- Camera – shortcut: *c*.
- Type visibility – shortcut: *v*.
- Visual settings – shortcut: *l*.
- Datasets – shortcut: *d*.
- Location log.
- Bookmarks – shortcut: *b*.

Each pane can be **expanded** and **collapsed** by clicking on the button or by using the respective keyboard shortcut (listed in the button tooltip).

Anchored to the bottom-left of the screen we can find six buttons to perform a few special actions:

- Toggle the mini-map – shortcut: *Tab*.
- Load a dataset – shortcut: *Ctrl* + *o*.
- Open the preferences window – shortcut: *p*.
- Show the session log – shortcut: *Alt* + *l*.
- Show the help dialog – shortcut: *h* or *F1*.
- Exit Gaia Sky – shortcut: *Esc*.

### Camera info panel

The **camera info panel**, also known as **focus info pane**, is anchored to the bottom-right of the main window.

**Docs**

See the *camera info panel section* of the user manual.

### Quick info bar

To the top of the screen, we can see the **quick info bar** which provides information on the current time, the current focus object (if any), the current closest object to our location and the current home object. The colors of these objects (green, blue, orange) correspond to the colors of the crosshairs. The crosshairs can be enabled or disabled from the interface tab in the preferences window (use *p* to bring it up).

**Docs**

See the *quick info bar section* for more information.

### System info panel

Gaia Sky has a built-in system information panel that provides system information and is hidden by default. We can bring it up with *ctrl* + *d*, or by ticking the *Show debug info*" check box in the *System* tab of the preferences window. By default, the debug panel is collapsed.

Expand the system info panel with the + symbol to get additional information.

The debug panel shows information on the current graphics device, system and graphics memory, the amount of objects loaded and on display, the octree (if a LOD dataset is in use) or the SAMP status.

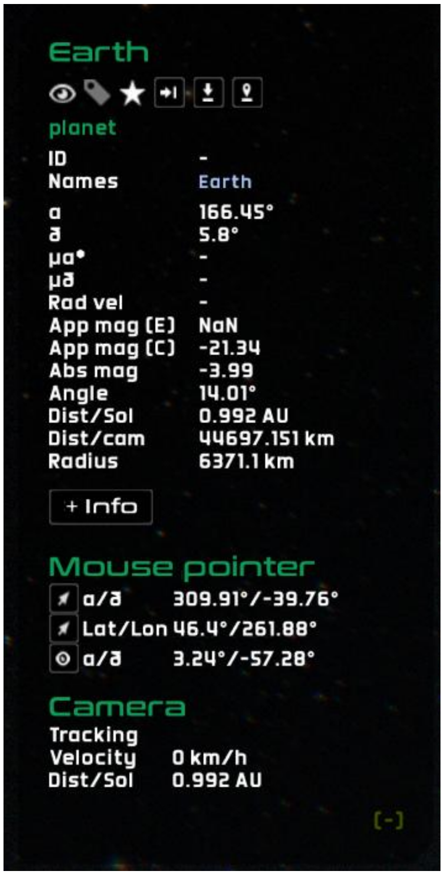



Fig. 6: The camera info pane when the camera is in focus mode. In this state, it is also referred to as focus info pane, and it displays information on the focus (top), the mouse pointer (middle), and the camera position and state (bottom).


Fig. 7: The quick info bar is anchored to the top of the window and displays useful information at a glance.


Fig. 8: Collapsed system info panel

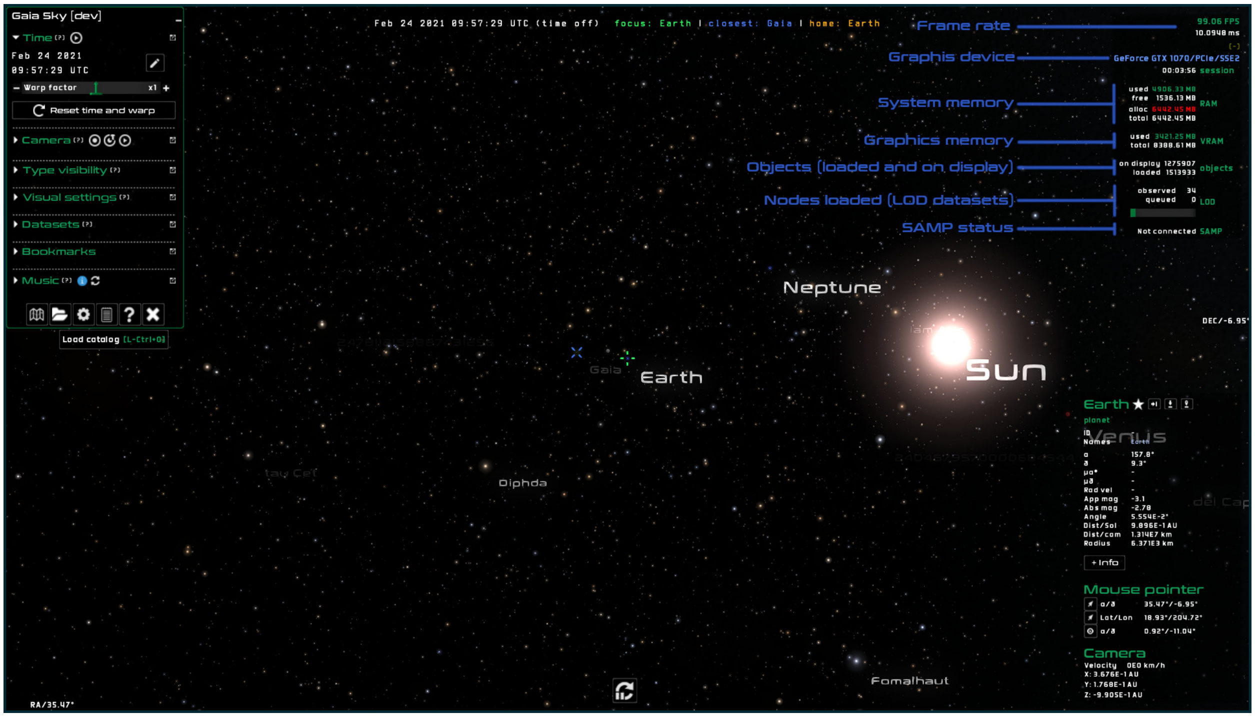



Fig. 9: Expanded system info panel

Additional debug information can be obtained in the system tab of the help dialog (*?* or *h*).

**Docs**

See the *system info panel section* for a full description.

## Time controls

**Tip:** Open the time pane by clicking on the clock button, or by pressing *t*.

Gaia Sky can simulate time. Play and pause the simulation using the / *Play/Pause* buttons in the time pane, or toggle using *Space*.

The *Time warp* slider lets us modify the speed at which the simulation time runs w.r.t. real time. Use *,* or and *.* or to divide by 2 and double the value of the time warp respectively. If we keep either *.* or *,* pressed, the warp factor will increase or decrease steadily.

Use the *Reset time and warp* button to reset the time warp to x1, and set the time to the current real world time (UTC).

Now, go ahead and press *Home*. This will bring us back to Earth. Now, start the time with or *Space* and **drag the slider slightly to the right** to increase its speed. We see that the Earth rotates faster and faster as we move the slider to the right. Now, **drag it to the left** until time is reversed and the Earth starts rotating in the opposite direction. Now time is going backwards!


Fig. 10: The time pane in the controls window of Gaia Sky.

If we set the time warp high enough we will notice that as the bodies in the Solar System start going crazy, the stars start to slightly move. That's right: Gaia Sky also simulates proper motions.

### 1.3.6 Camera modes

We have already talked about the **focus camera mode**, but Gaia Sky provides some more camera modes:

- *0* - **Free mode**: the camera is not locked to a focus object and can roam freely. The movement is achieved with the scroll wheel of the mouse, and the view is controlled by clicking and draggin the left and right mouse buttons
- *1* - **Focus mode**: the camera is locked to a focus object and its movement depends on it
- *2* - **Game mode**: similar to free mode but the camera is moved with *wasd* and the view (pitch and yaw) is controlled with the mouse. This control system is commonly found in FPS (First-Person Shooter) games on PC
- *3* - **Spacecraft mode**: take control of a spacecraft (outside the scope of this tutorial)

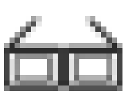
The most interesting mode is **free mode** which lets us roam freely. Go ahead and press *0* to try it out. The controls are a little different from those of **focus mode**, but they should not be to hard to get used too. Basically, use the **left mouse button** to yaw and pitch the view, use *shift* to roll, and use the **right mouse button** to pan.

**Docs**

See the *camera modes section* of the user manual.

### 1.3.7 Special render modes

There are three special render modes: **3D mode**, **planetarium mode**, **panorama mode** and **orthosphere view**. We can access these modes using the buttons at the bottom of the camera pane or the following shortcuts:

-  or *ctrl* + *s* - 3D mode
-  or *ctrl* + *p* - Planetarium mode

- or *ctrl* + *k* - Panorama mode
- or *ctrl* + *j* - Orthosphere view

**Docs**

See the *stereoscopic mode*, the *planetarium mode*, the *panorama mode*, and the *orthosphere view* sections of the user manual.

### 1.3.8 Type visibility

**Tip:** Expand and collapse the visibility pane by clicking on the eye button or with *v*.

The visibility pane offers controls to hide and show object types. Object types are groups of objects that are of the same category, like stars, planets, labels, galaxies, grids, etc. The pane also contains a button at the bottom that gives access to the *per-object visibility window*, which enables visibility control for individual objects.

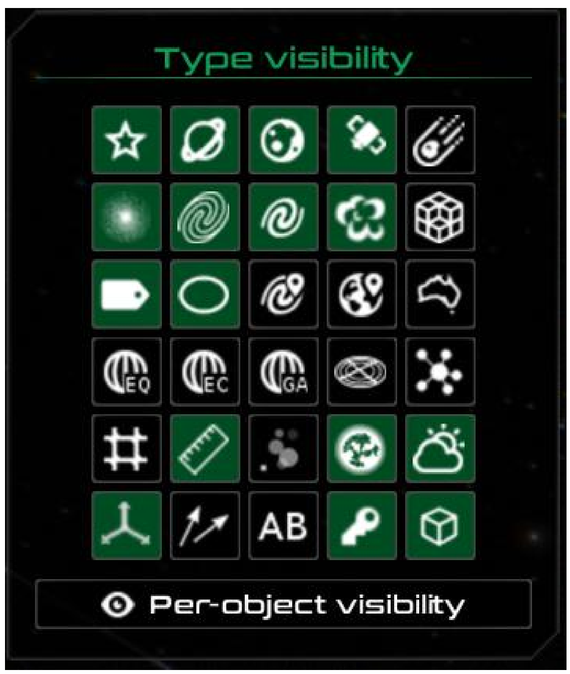



Fig. 11: The visibility pane contains controls to hide and show types of objects.

For example, we can hide the stars by clicking on the stars button. The object types available are the following:

- – Stars
- – Planets

- – Moons
- – Satellites
- – Asteroids
- – Star clusters
- – Milky Way
- – Galaxies
- – Nebulae
- – Meshes
- – Equatorial grid
- – Ecliptic grid
- – Galactic grid
- – Labels
- – Titles
- – Orbits
- – Locations
- – Cosmic locations
- – Countries
- – Constellations
- – Constellation boundaries
- – Rulers
- – Particle effects
- – Atmospheres

-  – Clouds
-  – Axes
-  – Velocity vectors
-  – Others

### Velocity vectors

One of the elements, the **velocity vectors**, enable a few properties when selected.

- **Number factor** – control how many velocity vectors are rendered. The stars are sorted by magnitude (ascending) so the brightest stars will get velocity vectors first
- **Length factor** – length factor to scale the velocity vectors
- **Color mode** – choose the color scheme for the velocity vectors
- **Show arrowheads** – Whether to show the vectors with arrow caps or not

**Tip:** Control the width of the velocity vectors with the **line width** slider in the **visual settings** pane.

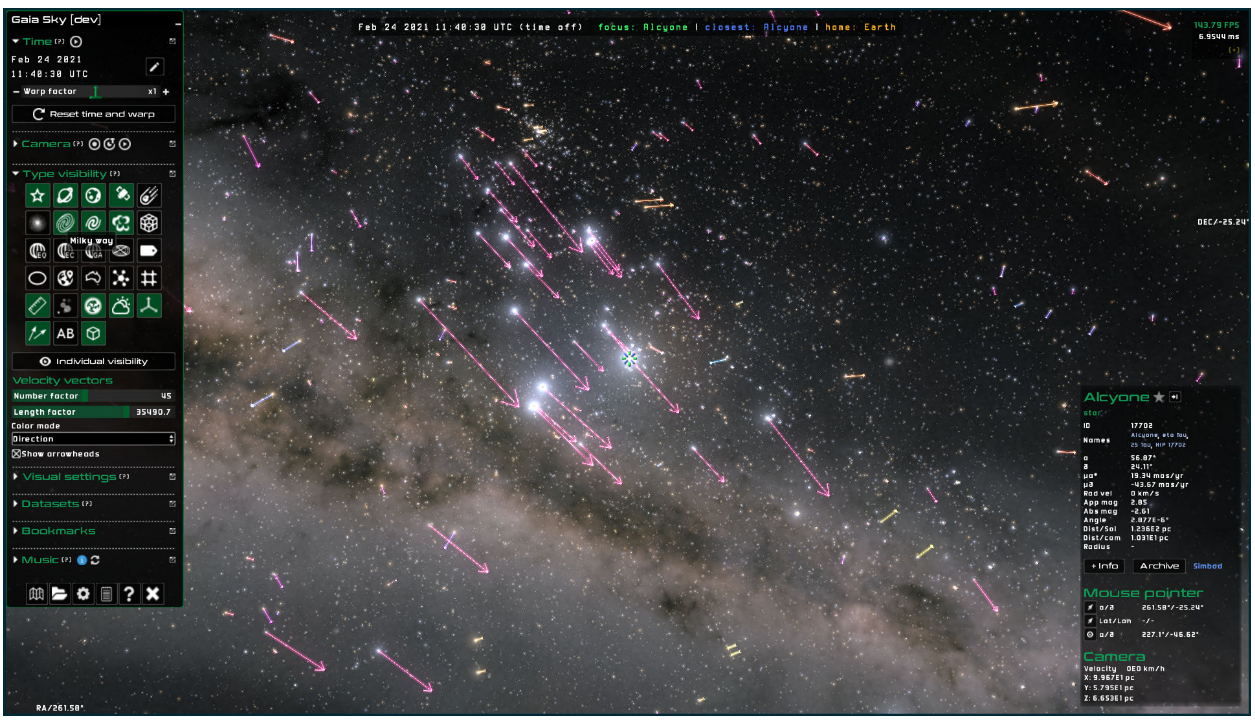



Fig. 12: Velocity vectors in Gaia Sky

**Docs**

See the *velocity vectors section* of the user manual.

### 1.3.9 Visual settings

**Tip:** Expand and collapse the visual settings pane by clicking on the bolt button or with *l*.

The **visual settings** pane contains a few options to control the shading of stars and other elements:

- **Star brightness** – control the brightness of stars.
- **Magnitude multiplier** – exponent of power function that controls the brightness of stars. Controls the brightness difference between bright and faint stars.
- **Star glow factor** – close-by star size.
- **Point size** – size of point-like stars and other objects.
- **Base star level** – the minimum brightness level for all stars.
- **Ambient light** – control the amount of ambient light. This only affects the models such as the planets or satellites.
- **Line width** – control the width of all lines in Gaia Sky (orbits, velocity vectors, etc.).
- **Label size** – control the size of the labels.
- **Elevation multiplier** – scale the height representation for planets with elevation maps.

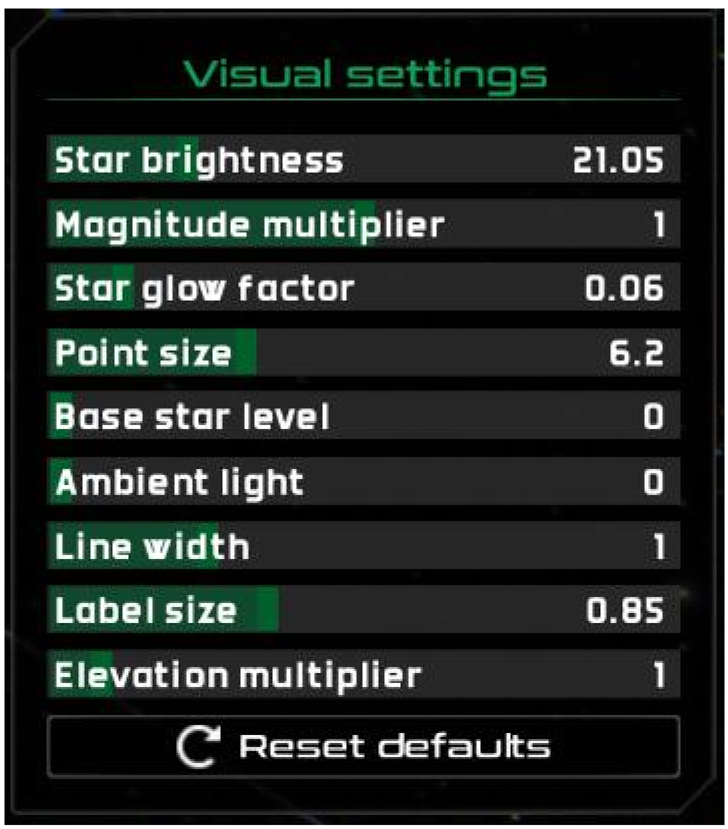



Fig. 13: The visual settings pane.

### 1.3.10 External datasets

We can also load datasets into Gaia Sky at run time. Right now, the VOTable, CSV and FITS formats are supported. Gaia Sky needs some metadata in the form of *UCDs* or column names in order to parse the dataset columns correctly.

**Docs**

See to the *STIL data loader* section of the Gaia Sky user manual for more information on how to prepare your datasets for Gaia Sky.

The datasets loaded in Gaia Sky at a certain moment can be found in the *datasets pane* of the control panel. Open it by clicking on the button or by pressing *d*.

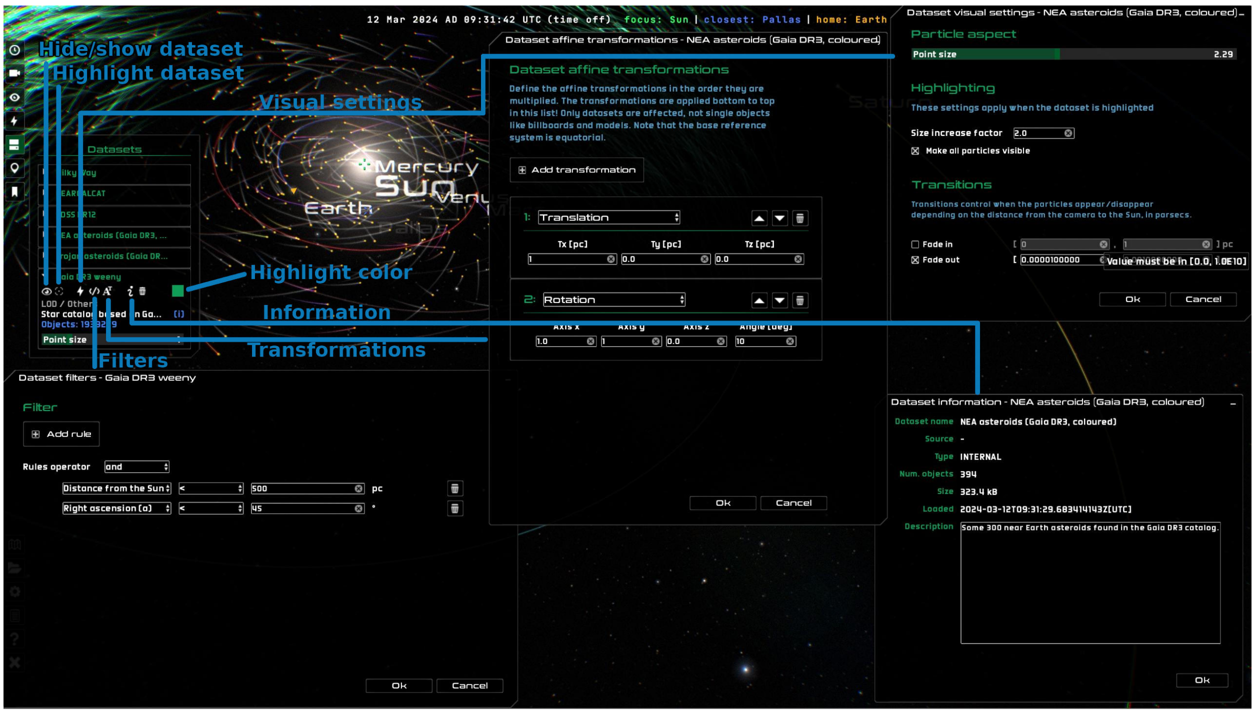



Fig. 14: Datasets pane of Gaia Sky.

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

- Directly from the UI, using the button (anchored to the bottom-left) or pressing *ctrl* + *o*.
- Through *SAMP*, via a connection to another astronomy software package such as Topcat or Aladin.
- Via a script, using one of the dataset loading API calls.
- Creating a dataset in the Gaia Sky format so that it appears in the dataset manager (see *here*.

**Docs**

See the *data format section* to know how to create a Gaia Sky dataset (advanced!).

**Loading a dataset from the UI** – Go ahead and remove the current star catalog by clicking on the icon in the datasets pane. Now, download a raw Hipparcos dataset VOTable, click on the icon (or press *ctrl* + *o*) and select the file. In the next dialog just click *Ok* to start loading the catalog. 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 on 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 (Odysee mirror, YouTube mirror):

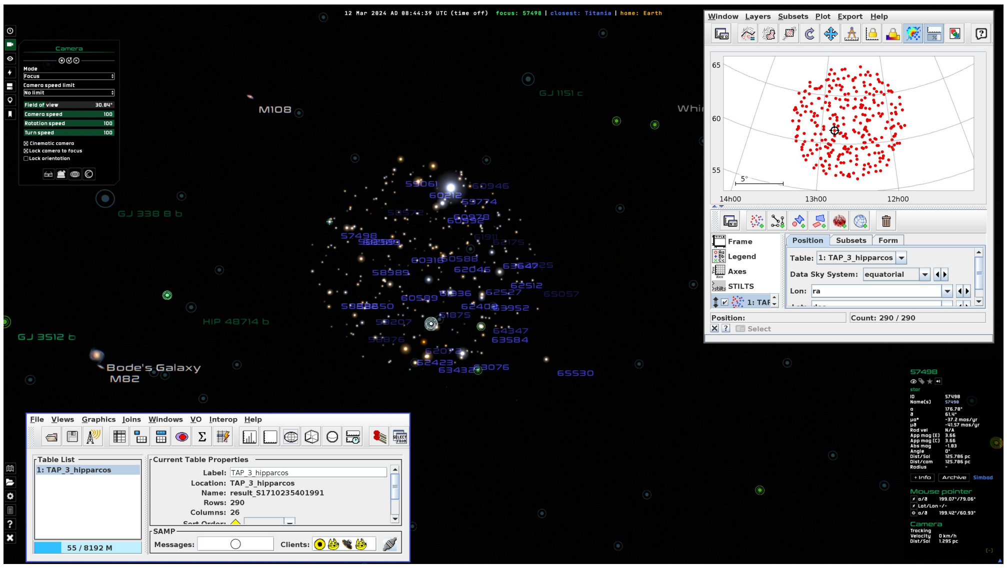



Fig. 15: Loading a dataset from Topcat through SAMP (click for video)

**Loading a dataset via scripting** – Wait for the scripting section of this course.

**Preparing a descriptor file** – Not addressed in this tutorial. See *the catalog formats section* for more information.

## Working with datasets

---

**Tip:** Expand and collapse the datasets pane by clicking on the hard disk button or with *d*.

---

All datasets loaded are displayed in the datasets pane in the control panel. A few useful tips for working with datasets:

- The visibility of individual datasets can be switched on and off by clicking on the button
- Remove datasets with the button
- We can **highlight a dataset** by clicking on the button. The highlight color is defined by the color selector right on top of it. Additionally, we can map an attribute to the highlight color using a color map. Let's try it out:
  1. Click on the color box in the Hipparcos dataset we have just loaded from Topcat via SAMP
  2. Select the radio button "Color map"
  3. Select the *rainbow* color map

4. Choose the attribute. In this case, we will use the number of transits, *ntr*
5. Click *Ok*
6. Click on the highlight dataset icon to apply the color map

- We can **define basic filters** on the objects of the dataset using their attributes from the dataset preferences window. For example, we can filter out all stars with $\delta > 50°$:

1. Click on the dataset preferences button
2. Click on *Add filter*
3. Select your attribute (declination $\delta$)
4. Select your comparator ($<$)
5. Enter your value, in this case 50
6. Click *Ok*
7. The stars with a declination greater than 50 degrees should be filtered out

Multiple filters can be combined with the **AND** and **OR** operators

### 1.3.11 External information

Gaia Sky offers three ways to display external information on the current focus object: **Wikipedia**, **Gaia archive** and **Simbad**.


Fig. 16: Wikipedia, Gaia archive and Simbad connections

- The *+Info* button opens a view that contains the local data on the object, and a preview of the Wikipedia article on this object, if it exists.

- When the *Archive* button appears in the focus info pane, it means that the full table information of selected star can be pulled from the Gaia archive.
- When the `Simbad` link appears in the focus info pane, it means that the objects has been found on Simbad, and we can click the link to open it in the web browser.

### 1.3.12 Scripting

Gaia Sky exposes an API that is accessible through Python (via Py4j) or through HTTP over a network (using the *REST API HTTP server*).

In this tutorial, we focus on the writing of Python scripts that tap into the Gaia Sky API. You will need Python 3 installed, along with the packages NumPy and Py4j.

To **install** the packages, run this in a terminal:

```
pip3 install --user numpy py4j
```

Once you have those installed, you can run a script with the system Python 3 interpreter. Of course, you need to launch Gaia Sky in the same computer for the connection to succeed. Right now, only local scripting is supported. If you need to operate Gaia Sky over the network, have a look at the *REST API section*.

To **run a script** named `my-gaiasky-script.py`, run this in a terminal:

```
python3 my-gaiasky-script.py
```

If everything works well, the connection should succeed and Gaia Sky should react accordingly.

But wait, we don't have a script to run yet! Do not fret, in the next section we learn the basics of writing a script for Gaia Sky.

**Docs**

See the *scripting section* in the user manual.

#### A basic script

Writing a basic script is quite simple. Essentially, you need a header that imports Py4j and creates the connection object. Then, you can start using the connection object to run calls.

The following script simply connects to Gaia Sky and prints *"Hello from a script!"* to both Python and the Gaia Sky log.

```
from py4j.clientserver import ClientServer, JavaParameters

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

# User code goes here.
# We use the 'gs' object to access the API.

# Let's print something.
message = "Hello from a script!"
# Print to Gaia Sky.
```

(continues on next page)

(continued from previous page)

```
gs.print(message)
# Print with Python.
print(message)

# Shutdown the gateway at the end.
gateway.shutdown()
```

Note that you need to **shutdown** the gateway at the end, this is important to clean things up and be able to run more scripts afterwards!

It is cool that we can print messages, but what other actions can we perform via scripting? Read on to know more about the API.

### Gaia Sky API

The Gaia Sky **API** (here) contains many more calls to interact with the platform in real time from Python scripts or a REST HTTP server. The API includes calls to:

- Add and remove messages and images to the interface,
- start and stop time, and change the time warp,
- add scene elements like shapes, lines, etc.,
- load full datasets in VOTable, CSV, FITS, or the internal JSON format,
- manage datasets (highlight, change settings, etc.),
- manipulate the camera position, orientation and mode,
- move the camera by simulating mouse actions (rotate around, forward, etc.),
- activate special modes like planetarium or panorama,
- create smooth camera transitions in position and orientation,
- change the various settings and preferences,
- back-up and restore the full configuration state,
- take screenshots, use the frame output mode.

The API specification is documented in the links below:

- Latest API version
- Older API versions (javadoc).

### Showcase scripts

The Gaia Sky repository contains many test and showcase scripts that may help with getting up to speed with Gaia Sky scripting. Many of them contain comments explaining what is going on:

- Interesting **showcase scripts** can be found here.
- Basic **testing scripts** can be found here.

#### Hands-on session

Here, we have a look at some real world scripts (full file listing), and write our own to later run them on Gaia Sky.

- Scripting presentation (dropbox link).

The proposed scripts are:

- Locating_the_Hyades_tidal_tails.py – a simple sequential script which exemplifies some of the most common API calls, and can be used to capture a video. The script requires the following data and subtitles files to run (save them in the same directory as the script):
  - Aldebaran.vot
  - Hyades_stars.csv
  - Hyades_subtitles.srt
  - distSDR3_N.csv
- line-objects-update.py – a script showcasing the feature to run scripting code within the Gaia Sky main loop, so that it runs synchronized with the main loop, every frame. This is used to run update operations every single frame. In our test script, we create a line between the Earth and the Moon, start the time simulation, and update the position of the line every frame so that it stays in sync with the scene.

### 1.3.13 Camera paths

Gaia Sky includes a feature to record and play back camera paths. This comes in handy if we want to showcase a certain itinerary through a dataset, for example.

**Recording a camera path** — The system will capture the camera state at every frame and save it into a `.gsc` (for Gaia Sky camera) file. We can start a recording by clicking on the ⦿ icon in the camera pane of the control panel. 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 the *folders* section in the Gaia Sky documentation).

**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.

**Tip: 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 camcorder 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, we can edit the “Maximum frame rate” input in the graphics settings of the preferences window.

**Docs**

See the *camera paths section* in the user manual.

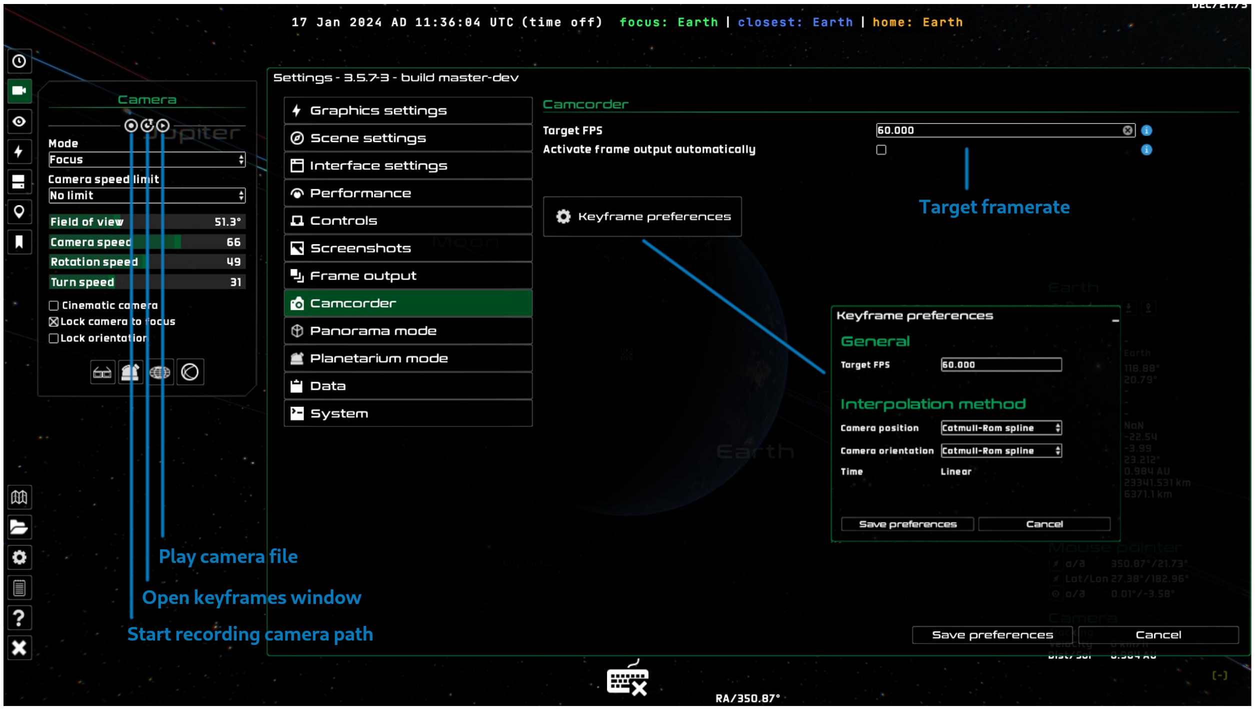



Fig. 17: Location of the controls of the camcorder in Gaia Sky

### Keyframe system

The camera path system 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 outside 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 .

**Docs**

See the *keyframes system section* in the user manual.

## 1.3.14 Frame output mode

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

We can start the frame output system by pressing *F6*. Once active, the system starts saving each still frame to disk (frame rate goes down, most probably). 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.

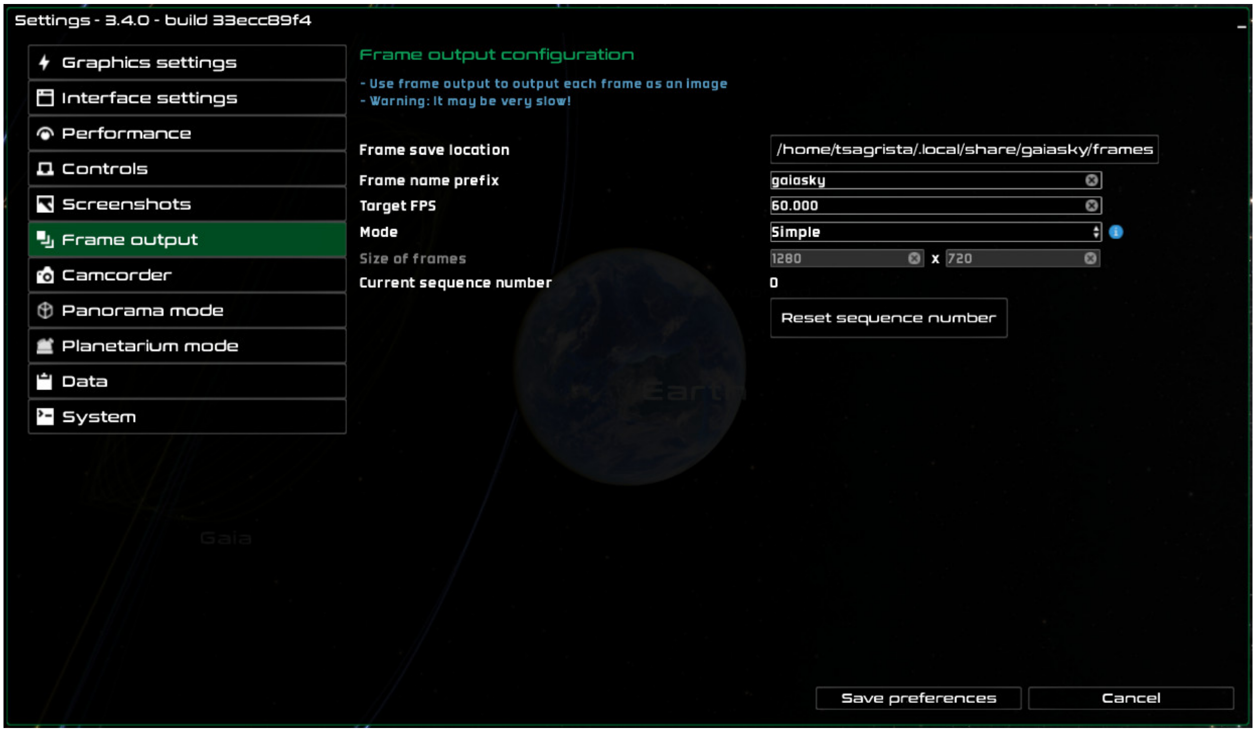



Fig. 18: The configuration screen for the frame output system

#### Create a video with `ffmpeg`

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* of the Gaia Sky user manual.

### 1.3.15 Conclusion

Congratulations! You have reached the end of the quick start guide. You are now a totally legit Gaia Sky master ;)

## 1.4 User manual

### 1.4.1 Dataset manager

When you start Gaia Sky, you are met with the welcome screen, which contains some information and buttons to start Gaia Sky, launch the dataset manager, open the preferences window, open the help window, and exit Gaia Sky.

The **dataset manager** provides an integrated way of downloading and enabling/disabling datasets. Enabled datasets are loaded when Gaia Sky starts up. All downloads are performed over a secure, encrypted HTTPS connection, and data consistency is checked once the download has finished with *sha256* checksums.

**Contents**

- *Dataset manager*

- Welcome screen
- Dataset manager
  - Data location
  - Available datasets
  - Installed datasets


### Welcome screen


Fig. 19: The welcome screen in Gaia Sky.

Gaia Sky greets the user with a welcome screen which lets her start Gaia Sky, manage the datasets or exit.

### Dataset manager

The dataset manager provides a hassle-free way of downloading, updating and enabling/disabling your datasets.

#### Data location

All datasets are installed in the data location. Check out *the directories section* for the defaults. The install location can be changed by clicking on the button next to *Data location*, at the bottom of the window. When changing the data location no data files are actually moved. If you want to migrate your data files to a different location, you must first do so by hand, and then point Gaia Sky to the new directory.

#### Available datasets

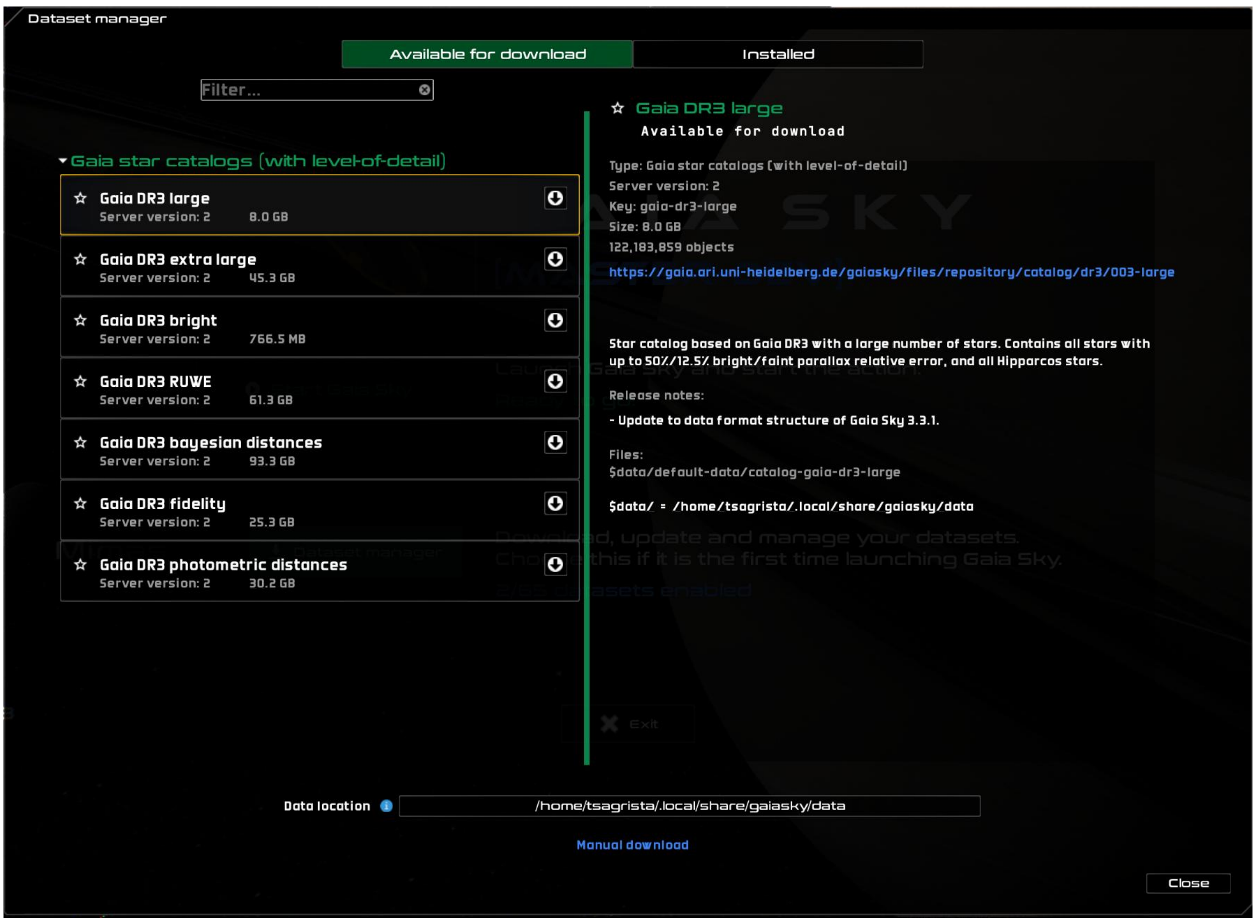



Fig. 20: The available datasets view in the dataset manager.

At the top of the window you can choose to view the datasets available for download, using the *Available for download* tab, and the installed datasets, using the *Installed* tab.

Each of these two views consists of a two-pane layout. The **left pane** displays a list of the installed or available datasets with some very basic information for each. Once the user clicks on one of these datasets, the **right pane** displays extensive information about the dataset and its files.

The *Available for download* tab lists all server datasets that can be downloaded and installed locally. In order to display a dataset in Gaia Sky, it must first be installed locally. To install a dataset, use the install button or right-click on the dataset entry in the left pane and select *Install* in the context menu.

Multiple datasets can be downloading at the same time without problems. The download process can be canceled at any time by clicking on the *Cancel download* button in the **right pane**. Canceled downloads can be resumed any time without losing progress, as the `.part` files are kept in the file system.

### Installed datasets

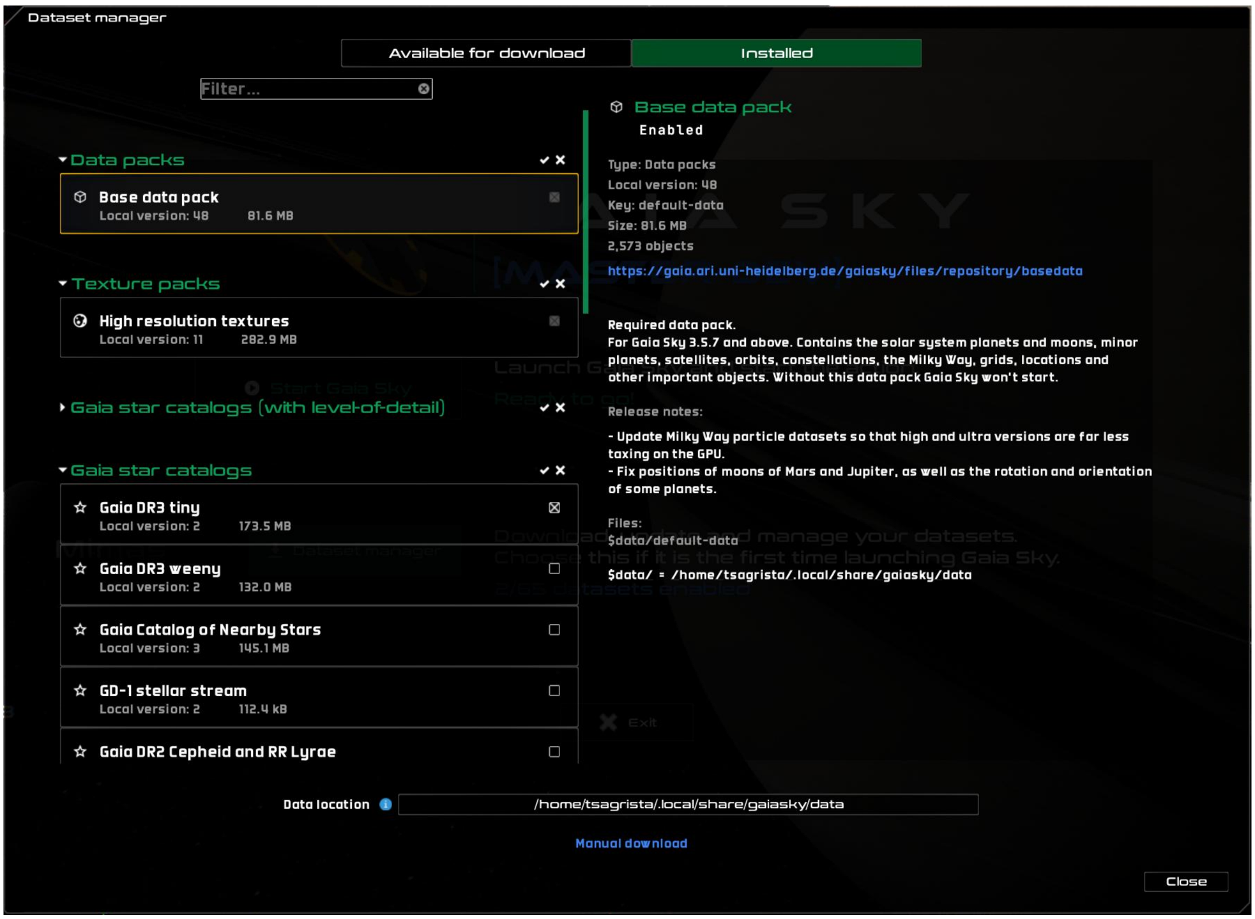



Fig. 21: The installed datasets view in the dataset manager.

The installed datasets view displays the datasets found in the currently selected data directory. From this view, you can **enable** and **disable** datasets by either using the checkbox next to the dataset name, or by right-clicking and selecting *Enable* or *Disable* in the context menu. Only datasets that are enabled are loaded into Gaia Sky.

Some datasets are always enabled. This is the case for all texture packs (whose usage depends on the graphics quality setting) and for the base data pack.

From this view you can also remove datasets. To do so, bring up the context menu by right-clicking on the dataset entry in the datasets list (left pane) and select *Remove*. Removing a dataset actually deletes all of its files on disk, so a confirmation dialog is displayed.

### 1.4.2 Controls

This section describes the controls of Gaia Sky.

**Contents**


- *Controls*
  - *Keyboard controls*
    - *Keyboard mappings*
    - *Free/focus mode controls*
    - *Spacecraft mode controls*
  - *Mouse controls*
    - *Focus mode*
    - *Free mode*
    - *Game mode*
  - *Gamepad controls*
    - *Default camera mappings*
    - *Spacecraft camera mappings*
    - *Gamepad UI*
  - *GUI navigation*


#### 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 it into `$GS_CONFIG/mappings/` (if it is not yet there) 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
```

(continues on next page)

(continued from previous page)

```
#action.playcamera                             = C
```

The available actions are the following:

- `action.toggle/element.stars` – toggle stars
- `action.toggle/element.planets` – toggle planets
- `action.toggle/element.moons` – toggle moons
- `action.toggle/element.satellites` – toggle satellites
- `action.toggle/element.orbits` – toggle orbits
- `action.toggle/element.labels` – toggle labels
- `action.toggle/element.equatorial` – toggle equatorial grid
- `action.toggle/element.ecliptic` – toggle ecliptic grid
- `action.toggle/element.galactic` – toggle galactic grid
- `action.toggle/element.clusters` – toggle star clusters
- `action.toggle/element.asteroids` – toggle asteroids
- `action.toggle/element.constellations` – toggle constellations
- `action.toggle/element.boundaries` – toggle constellation boundaries
- `action.toggle/element.meshes` – toggle meshes
- `action.toggle/element.keyframes` – toggle keyframes
- `action.toggle/element.recursivegrid` – toggle recursive grid
- `action.toggle/element.stereomode` – toggle stereoscopic mode
- `action.switchstereoprofile` – switch stereoscopic profile
- `action.toggle/element.planetarium` – toggle planetarium mode
- `action.toggle/element.planetarium.projection` – switch planetarium projection
- `action.toggle/element.360` – toggle cubemap mode
- `action.toggle/element.projection` – switch cubemap projection mode
- `action.toggle/element.orthosphere` – toggle orthosphere mode
- `action.toggle/element.orthosphere.profile` – switch orthosphere profile
- `action.toggle/element.octreeparticlefade` – toggle particle smooth transitions in LOD datasets
- `action.toggle/element.debugmode` – enable/disable debug information
- `action.toggle/element.cleanmode` – toggle clean UI mode (remove the user interface)
- `action.toggle/gui.minimap.title` – toggle minimap
- `action.toggle/gui.mousecapture` – toggle mouse capture
- `action.expandcollapse.pane/gui.time` – toggle time pane
- `action.expandcollapse.pane/gui.camera` – toggle camera pane
- `action.expandcollapse.pane/gui.visibility` – toggle visibility pane

- `action.expandcollapse.pane/gui.lighting` – toggle visual settings pane
- `action.expandcollapse.pane/gui.dataset.title` – toggle datasets pane
- `action.expandcollapse.pane/gui.bookmarks` – toggle bookmarks pane
- `action.screenshot` – capture and save screenshot
- `action.screenshot.cubemap` – save 6 current cubemap faces to image files (only in panorama, planetarium and orthosphere modes)
- `action.pauseresume` – start/stop time simulation
- `action.dividetime` – reduce time warp (x0.5)
- `action.doubletime` – increase time warp (x2)
- `action.time.warp.reset` – reset time warp
- `action.playcamera` – open a camera path file
- `action.decfov` – decrease field of view angle
- `action.incfov` – increase field of view angle
- `action.toggle/camera.mode` – switch camera modes
- `camera.full/camera.FREE_MODE` – enable free mode
- `camera.full/camera.FOCUS_MODE` – enable focus mode
- `camera.full/camera.GAME_MODE` – enable game mode
- `camera.full/camera.SPACECRAFT_MODE` – enable spacecraft mode
- `action.toggle/camera.cinematic` – toggle cinematic camera mode
- `action.camera.speedup` – keep pressed to speed the camera up
- `action.starpointsize.inc` – increase star point size
- `action.starpointsize.dec` – decrease star point size
- `action.starpointsize.reset` – reset star point size
- `action.gotoobject` – immediately move to focus object
- `action.home` – go to home object
- `action.search` – open search dialog
- `action.log` – show system log
- `action.preferences` – show preferences dialog
- `action.help` – open help dialog
- `action.slave.configure` – show slave configuration dialog
- `action.loadcatalog` – load a dataset
- `action.upscale` – debug upscale filter
- `action.keyframe` – add new keyframe at the end
- `action.controller.gui.in` – show/hide controller UI
- `action.toggle/element.controls` – expand/collapse UI controls
- `action.ui.reload` – reload user interface

- `action.resettime` – reset simulation time to current
- `action.toggle/element.frameoutput` – toggle frame output
- `action.exit` – quit Gaia Sky
- `action.togglefs` – toggle full screen mode

Find the current keyboard mappings associations in the controls tab of the preferences window within Gaia Sky.

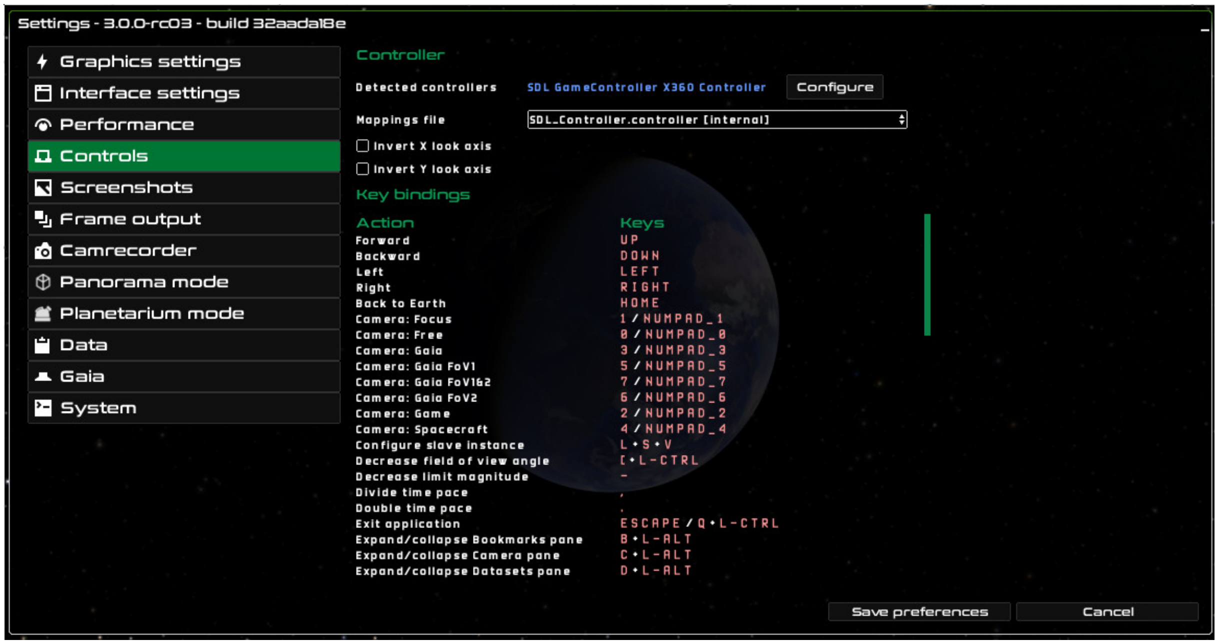



Fig. 22: The controls settings in Gaia Sky

## Free/focus mode controls

These are the default keyboard controls that apply to the focus, free and game camera modes.

| Key(s) | Action |
|---|---|
| $\uparrow$ | camera forward |
| $\downarrow$ | camera backward |
| $\rightarrow$ | rotate/yaw right |
| $\leftarrow$ | rotate/yaw left |
| *Ctrl* + *g* | instantly move to focus object |
| *Home* | back to Earth (or any other home object) |
| *Tab* | toggle minimap |
| *Ctrl* + *r* | reset time to current |
| *Num 0* or *0* | free camera |
| *Num 1* or *1* | focus camera |
| *Num 2* or *2* | game mode |
| *Num 3* or *3* | spacecraft mode |
| *Ctrl* + *o* | load new dataset |
| *Ctrl* + *m* | toggle camera mode |
| *Ctrl* + *c* | toggle cinematic camera behavior |

continues on next page

Table 1 – continued from previous page

| Key(s) | Action |
|---|---|
| `hold down` *z* | multiply camera speed |
| *Ctrl* + *w* | new keyframe |
| *Ctrl* + *k* | panorama mode |
| *Space* | pause/resume time |
| *F1* | help dialog |
| *F5* | take screenshot |
| *F6* | start/stop frame output mode |
| *F7* | save cubemap faces as image files |
| *F11* | toggle fullscreen/windowed mode |
| *Ctrl* + *f* or *f* | search dialog |
| *Esc* or *q* | quit application |
| *p* | open preferences dialog |
| *h* | open help dialog |
| *r* | run script dialog |
| *t* | toggle time pane |
| *c* | toggle camera pane |
| *v* | toggle visibility pane |
| *l* | toggle visual settings pane |
| *d* | toggle datasets pane |
| *b* | toggle bookmakrs pane |
| *Alt* + *c* | run camera path file dialog |
| *,* | halve time warp (hold for smooth decrease) |
| *.* | double time warp (hold for smooth increase) |
| *Ctrl* + *.* | reset time warp to 1 |
| *Shift* + *b* | toggle constellation boundaries |
| *Shift* + *c* | toggle constellation lines |
| *Shift* + *e* | toggle ecliptic grid |
| *Shift* + *g* | toggle galactic grid |
| *Shift* + *l* | toggle labels |
| *Shift* + *m* | toggle moons |
| *Shift* + *o* | toggle orbits |
| *Shift* + *p* | toggle planets |
| *Shift* + *q* | toggle equatorial grid |
| *Shift* + *s* | toggle stars |
| *Shift* + *t* | toggle satellites |
| *Shift* + *v* | toggle star clusters |
| *Shift* + *h* | toggle meshes |
| *Shift* + *r* | toggle recursive grid |
| *Shift* + *k* | toggle keyframes |
| *Shift* + *u* | expand/collapse control panel |
| *Ctrl* + *u* | toggle UI completely (hide/show user interface) |
| *Ctrl* + *d* | toggle debug info |
| *Ctrl* + *s* | toggle stereoscopic mode |
| *Ctrl* + *Shift* + *s* | switch between stereoscopic profiles |
| *Ctrl* + *k* | toggle 360 panorama mode |
| *Ctrl* + *Shift* + *k* | switch between 360 projections |
| *Ctrl* + *p* | toggle planetarium mode |
| *Ctrl* + *Shift* + *p* | switch planetarium projections |
| *Ctrl* + *j* | toggle orthosphere mode |

continues on next page

Table 1 – continued from previous page

| Key(s) | Action |
|---|---|
| *Ctrl + Shift + j* | switch between orthosphere profiles |

### Spacecraft mode controls

These controls apply only to the spacecraft mode.

| Key(s) | Action |
|---|---|
| *w* | apply forward thrust |
| *s* | apply backward thrust |
| *a* | roll left |
| *d* | roll right |
| *k* | stop spaceship automatically |
| *l* | stabilize spaceship automatically |
| ↑ | pitch up |
| ↓ | pitch down |
| ← | yaw left |
| → | yaw right |
| *PgUp* | increase engine power (x10) |
| *PgDown* | decrease engine power (x0.1) |

### 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

| Mouse + keys | Action |
|---|---|
| L-MOUSE DOUBLE-CLICK | select focus object |
| L-MOUSE CLICK | stop all rotation and translation movement |
| L-MOUSE + *DRAG* | apply rotation around focus |
| L-MOUSE + *Shift + DRAG* | camera roll |
| R-MOUSE + *DRAG* | pan view freely from focus |
| M-MOUSE + *DRAG* or WHEEL | move towards/away from focus |

### Free mode

| Mouse + keys | Action |
|---|---|
| L-MOUSE DOUBLE-CLICK | select object as focus (changes to focus mode) |
| L-MOUSE CLICK | stop all rotation and translation movement |
| L-MOUSE + *DRAG* | pan view |
| L-MOUSE + *Shift + DRAG* | camera roll |
| M-MOUSE + *DRAG* or WHEEL | forward/backward movement |

### Game mode

Use the mouse to look around and *wasd* to move.

### Gamepad controls

Gaia Sky supports Game controllers through SDL. This means that most controllers should *just work* out-of-the-box. The default controller mappings file, `SDL_Controller.controller`, should always be used initially. Should this file not work for your controller, you can create your custom mappings easily and interactively by going to the preferences window > controls and clicking on the "Configure" button next to your controller. Then, follow screen instructions.

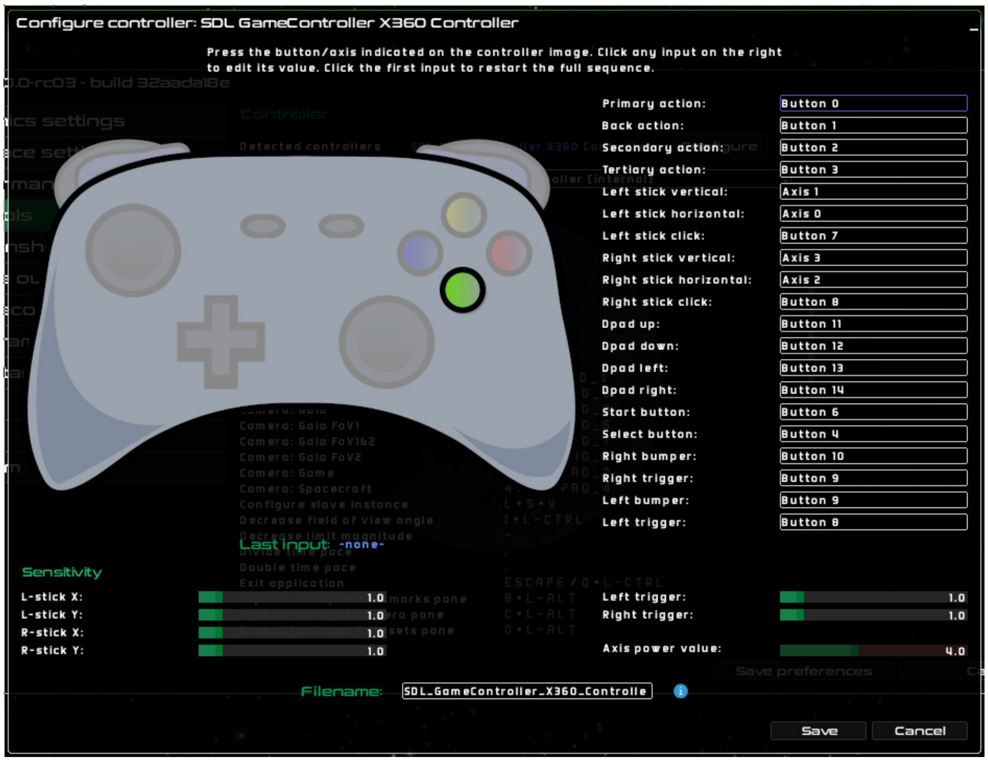



Fig. 23: Configuring gamepad controls in Gaia Sky

User mappings files (see here) can be added manually to `$GS_CONFIG/mappings` (see *folders*) folder, or set up automatically from within Gaia Sky. The controller mappings file contains the axis or button numbers for each input type. Below is an example of one such file.

```
#Controller mappings definition file for Wireless Steam Controller
axis.dpad.h=-1
axis.dpad.v=1
axis.lstick.h=0
axis.lstick.h.sensitivity=1.0
axis.lstick.v=1
axis.lstick.v.sensitivity=1.0
axis.lt=-1
axis.lt.sensitivity=1.0
axis.rstick.h=2
axis.rstick.h.sensitivity=1.0
```

(continues on next page)

(continued from previous page)

```
axis.rstick.v=3
axis.rstick.v.sensitivity=1.0
axis.rt=-1
axis.rt.sensitivity=-1.0
axis.value.pow=4.0
button.a=2
button.b=3
button.dpad.d=18
button.dpad.l=19
button.dpad.r=20
button.dpad.u=17
button.lb=6
button.lstick=13
button.lt=-1
button.rb=7
button.rstick=-1
button.rt=-1
button.select=10
button.start=11
button.x=4
button.y=5
```

### Default camera mappings

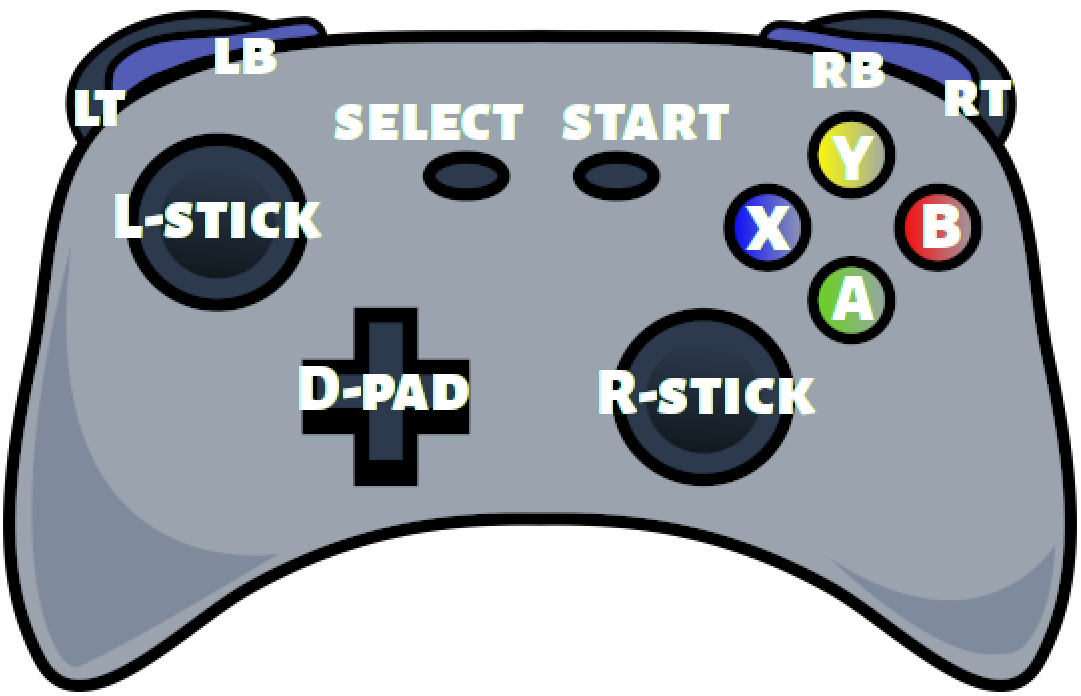



Fig. 24: Gamepad annotated with axes and buttons

The following table lists the actions assigned to each of the gamepad axes and buttons.

| Button/axis | Action |
|---|---|
|  /  | rotate around horizontally (focus mode), yaw (free mode) |
|  /  | rotate around vertycally (focus mode), pitch (free mode) |
|  /  | roll |
|  /  | forward/backward |
|  (right trigger) | roll right |
|  (left trigger) | roll left |
|  | preferences |
|  | toggle labels |
|  | toggle asteroids |
|  | toggle minimap |
|  | toggle orbits |
|  | hold to speed up time |
|  | hold to slow down time |
|  | start time |
|  | stop time |
|  (click) | stop time |

### Spacecraft camera mappings

In spacecraft mode, the actions mapped to the different gamepad axes and buttons are different. They are listed in the table below.

| Button/axis | Action |
|---|---|
| R / R | spacecraft yaw |
| R / R | spacecraft pitch |
| L / L | spacecraft roll |
| L / L | thrust forward/backward |
| RB (right bumper) | spacecraft roll right |
| LB (left bumper) | spacecraft roll left |
| RT (right trigger) | thrust forward |
| LT (left trigger) | thrust backward |
| A | toggle labels |
| B | toggle orbits |
| X | stop spacecraft |
| Y | level spacecraft |
| | increase engine power (x10) |
| | decrease engine power (x0.1) |

## Gamepad UI

The gamepad UI allows access to some basic actions and settings directly using a gamepad. To open it, press START .

There are seven tabs at the top that can be navigated with LB and RB . The tabs are the following:

- **Search** – provides a virtual keyboard to search for objects.
- **Bookmarks** – access the system bookmarks (limited to 4 nested folder levels).
- **Camera** – camera parameters like the mode or the field of view.
- **Time** – controls to start and stop time, as well as to set the time warp factor.
- **Types** – visibility of elements in Gaia Sky.
- **Controls** – gamepad settings and mappings.
- **Graphics** – graphics options like post-processing effect parameters.
- **System** – system-wide settings. Also a button to quit Gaia Sky.

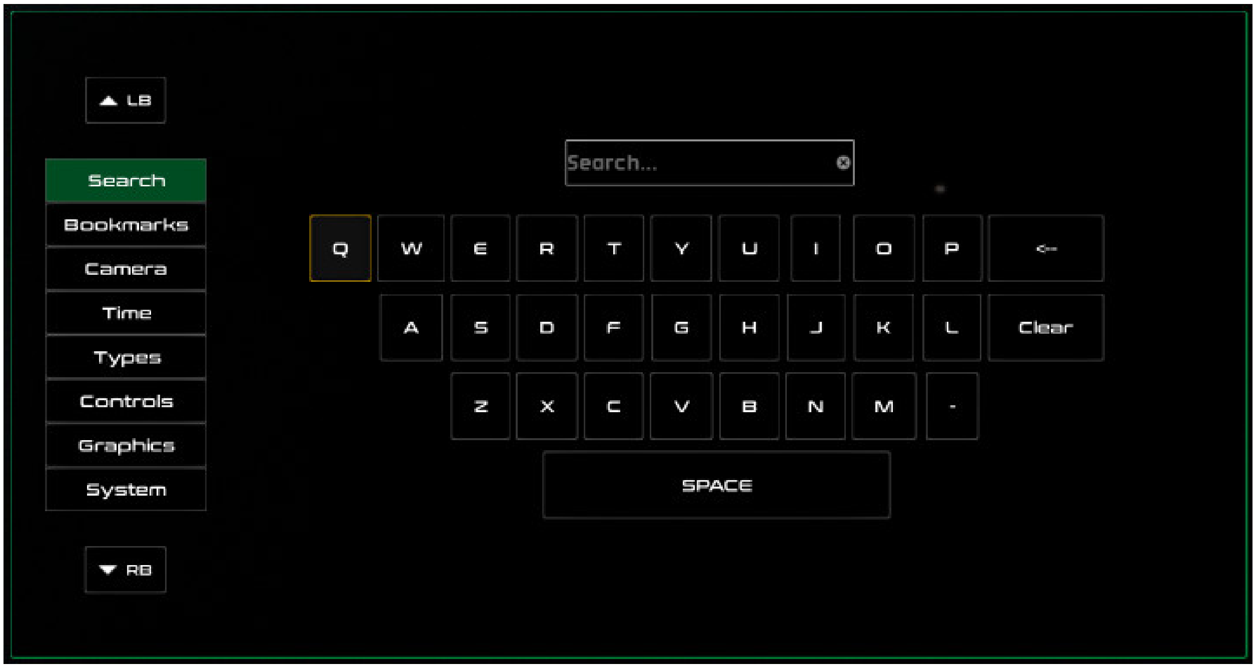
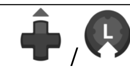
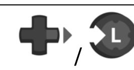
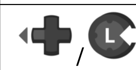
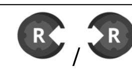
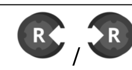
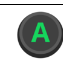



Fig. 25: The gamepad UI

Close the gamepad UI with  or  .

## GUI navigation

Gaia Sky supports the navigation of its GUI windows using the gamepad and keyboard mappings, additionally to the usual mouse clicks. Below are the most common actions and how to achieve them in a keyboard- or gamepad- centric workflow.

| Action | Keyboard | Gamepad |
|---|---|---|
| Action (click focused button) | *Enter* |  |
| Move focus up | ↑ |  |
| Move focus down | ↓ |  |
| Move focus right | → |  |
| Move focus left | ← |  |
| Move slider (when focused) | *←/→/home/end* |  |
| Move select box selection (when focused) | *←/→/home/end* |  |
| Check check box (when focused) | *Enter* |  |
| Cycle dialog bottom buttons | *Alt* |  |
| Close current dialog (with accept action) | / |  |
| Close current dialog (with cancel action) | *Esc* | / |
| Tab right | *Tab* |  |
| Tab left | *Shift + Tab* |  |

### 1.4.3 User interface

**Note:** Since Gaia Sky 3.5.5, Gaia Sky offers two UI modes: the **new UI** and the **old control panel**.

The main elements in the user interface are the control panes, the camera info panel, the quick info bar, the action buttons and the system info panel. They are all described in this section.

**Contents**

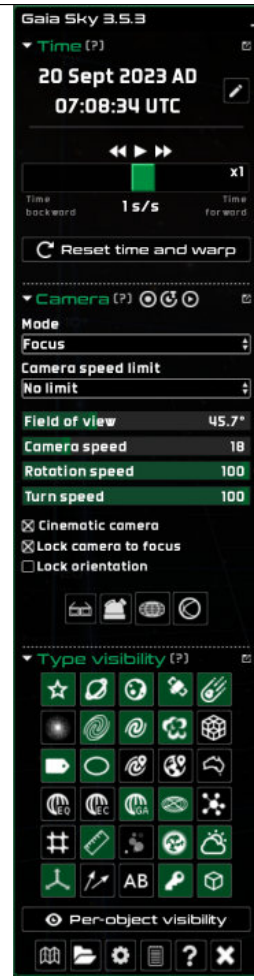

- *User interface*
  - *Control panes*
    - *Time pane*
    - *Camera pane*
    - *Visibility pane*
      - *Per-object visibility*
      - *Velocity vectors*

- * *Visual settings pane*
  * *Datasets pane*
  * *Location log pane*
  * *Bookmarks pane*
- *Camera info panel*
- *Quick info bar*
- *Action buttons*
  * *Minimap*
  * *Load dataset*
  * *Preferences window*
  * *System log*
  * *About/help*
  * *Exit*
- *System info panel*


### Control panes

The most important actions in Gaia Sky can be accessed via the control panes, anchored to the top-left of the screen. There are seven panes, *Time*, *Camera*, *Type visibility*, *Visual settings*, *Datasets*, *Objects*, and *Music*.

The panes are accessed via the control panel (in the old UI), or via buttons anchored to the left of the screen (new UI).

| | | |
|---|---|---|
|   |   |   |
| The new UI, based on anchored buttons to the left of the screen. | Collapsed control panel (old UI) | Expanded control panel (old UI) |

The seven panes, except for the Time pane in the old UI, are hidden at startup. To expand them and reveal its controls just click on the little arrow bottom icon at the right of the pane title (in the old UI), or click on the corresponding button (new UI). Use the arrow right icon, or the corresponding button to collapse them again. In the old UI, panes can also be detached to their own window. To do so, use the detach icon.

The seven panes are:

- *Time pane*.
- *Camera pane*.
- *Visibility pane*.
- *Visual settings pane*.
- *Datasets pane*.
- *Location log pane*.

- *Bookmarks pane*.

### Time pane

---

**Hint:** Expand and collapse the time pane by clicking on the clock button or with *t*.

---

Play and pause the simulation using the / `Play/Pause` buttons in the time pane, or toggle using *Space*. You can also change time warp, which is expressed as a scaling factor, using the provided **Warp factor** slider. Use , or and . or to divide by 2 and double the value of the time warp respectively. If you keep either of those pressed, the warp factor will increase or decrease steadily.

Use the *Reset time and warp* button to reset the time warp to x1, and set the time to the current real world time (UTC).


Fig. 26: The time pane displays the simulation date and time, along with controls to start and pause the time, and a slider to control the time warp (speed).

### Camera pane

---

**Hint:** Expand and collapse the camera pane by clicking on the camera button or with *c*.

---

In the camera options pane on the left you can select the type of camera. This can also be done by using the *Numpad 0-3* keys.

There are four 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.

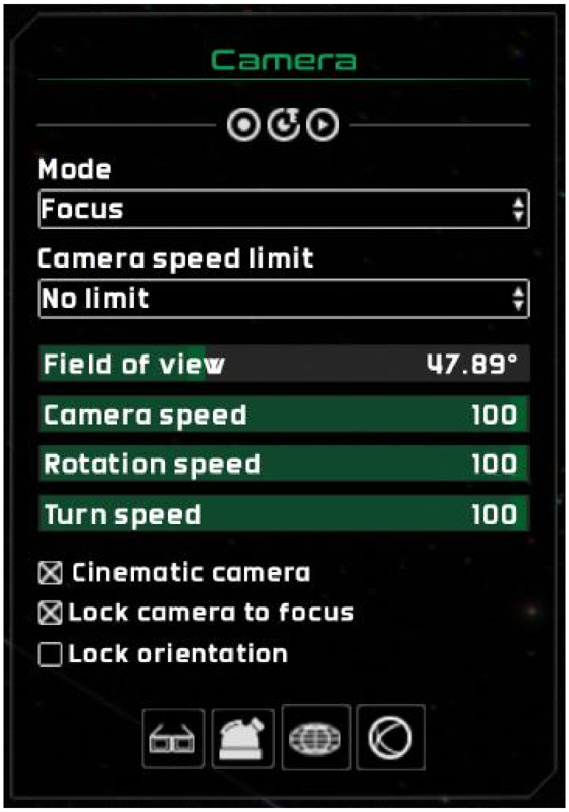



Fig. 27: The camera pane contains controls related to the camera setup and operation.

- **Spacecraft**– take control of a spacecraft and navigate around at will.

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, i.e. how fast it goes forward and backward.
- **Rotation speed** – control the transversal speed of the camera, i.e. how fast it rotates around an object.
- **Turn speed** – control the turning speed of the camera, i.e. how fast it changes its orientation (yaw, pitch and roll).

The checkbox **Cinematic camera** enables the cinematic behavior, described in the *camera behaviors section*.

The checkbox **Lock the camera to focus** links the reference system of the camera to that of the focus object and thus it moves with it. When focus lock is checked, the camera stays at the same relative position to the focus object.

The checkbox **Lock orientation** applies the rotation transformation of the focus to the camera, so that the camera rotates when the focus does.

### Visibility pane

**Hint:** Expand and collapse the visibility pane by clicking on the eye button or with *v*.

The visibility pane offers controls to hide and show object types. Object types are groups of objects that are of the same category, like stars, planets, labels, galaxies, grids, etc. The pane also contains a button at the bottom that gives access to the *per-object visibility window*, which enables visibility control for individual objects.

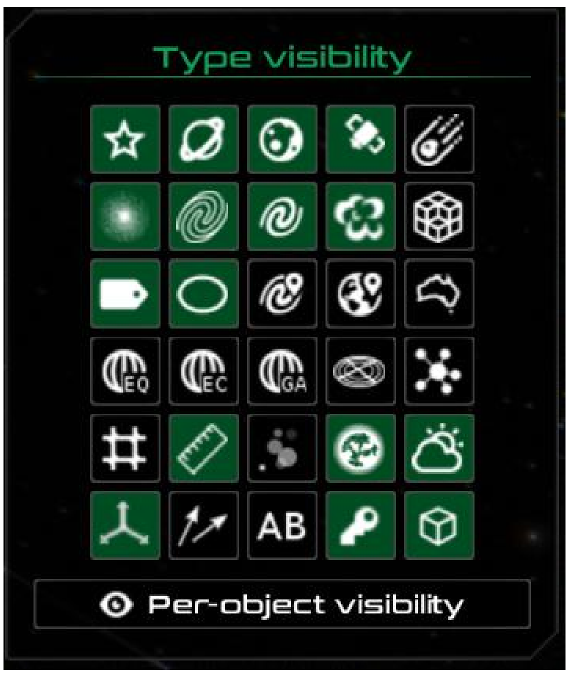



Fig. 28: The visibility pane contains controls to hide and show types of objects.

For example you can hide the stars by clicking on the `stars` toggle. The object types available are the following:

- – Stars
- – Planets
- – Moons
- – Satellites
- – Asteroids
- – Star clusters
- – Milky Way
- – Galaxies
- – Nebulae
- – Meshes
- – Equatorial grid
- – Ecliptic grid
- – Galactic grid
- – Recursive grid
- – Labels
- – Titles
- – Orbits
- – Locations
- – Cosmic locations
- – Countries
- – Constellations

- – Constellation boundaries
- – Rulers
- – Particle effects
- – Atmospheres
- – Clouds
- – Axes
- – Velocity vectors
- – Keyframes
- – Others

## Per-object visibility

This button provides access to controls to manipulate the individual visibility of objects.

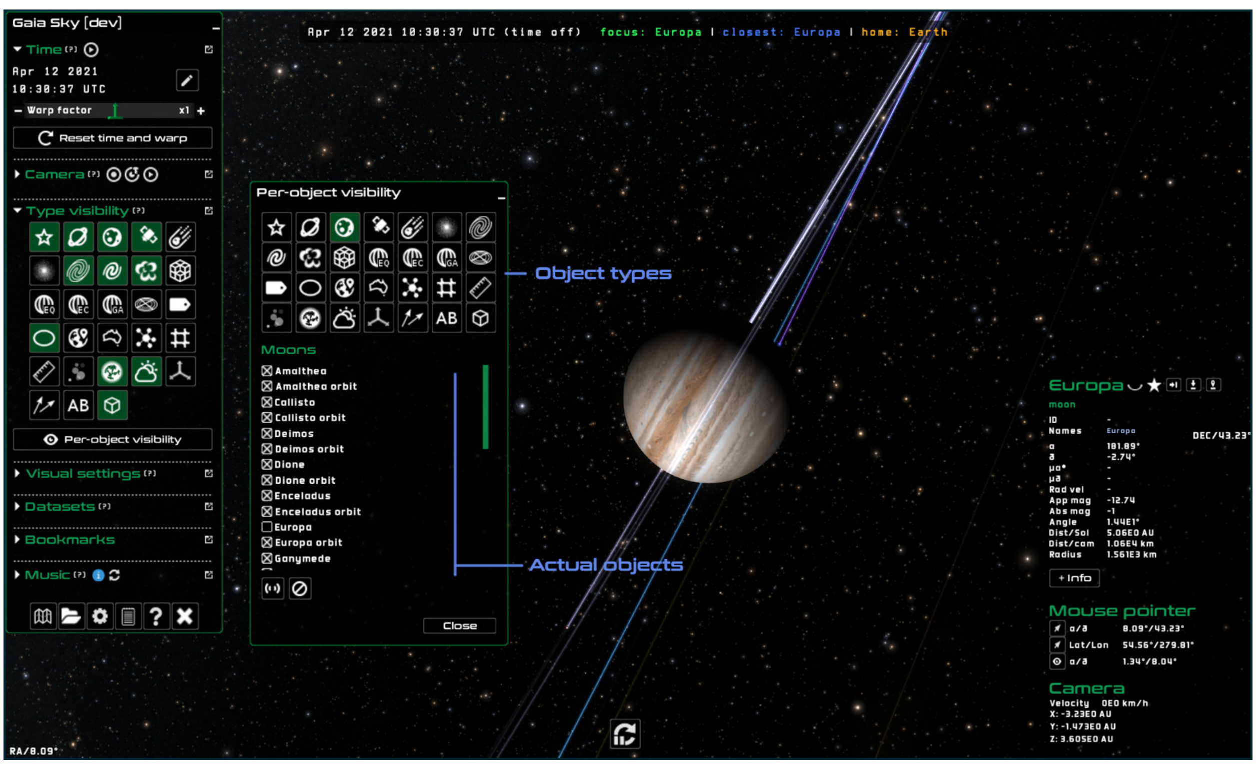



Fig. 29: Individual object visibility button and dialog

As shown in the image above, when clicking the *Per-object visibility* button, a new dialog appears, from which individual objects can be toggled on and off. They are organized per object type (top of the dialog). Once the object type is selected, the list of object appears in the bottom part.

**Hint: Stars do not appear** in the per-object visibility panel!

Since there are so many stars, they are not in the per-object visibility panel as single objects. Instead, they show up in groups. A single standalone catalog is a single star group. In the case of LOD catalogs like the ones based on Gaia data releases, each octree node contains a star group. However, individual star visibility can still be manipulated using the eye icon in the focus information pane when the star is focused.

## Velocity vectors

Enabling **velocity vectors** activates the representation of star velocities, if the currently loaded catalog provides them. Once velocity vectors are activated, a few extra controls become available to tweak their length and color.

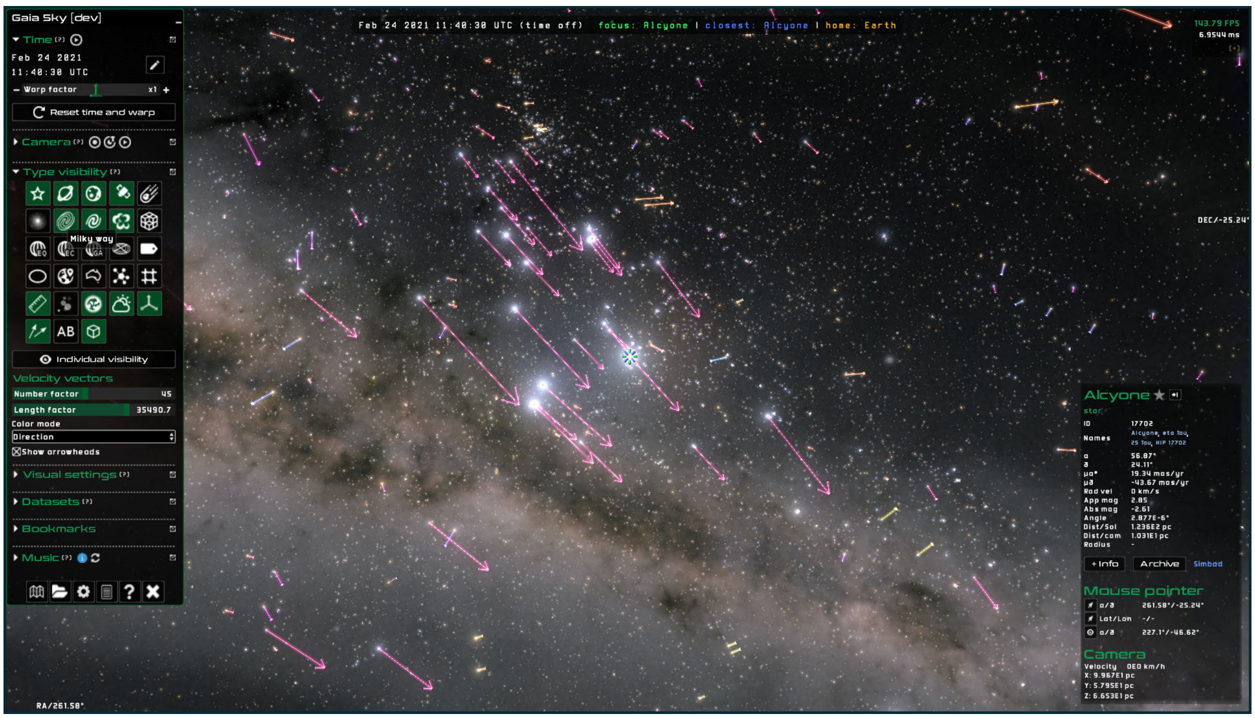



Fig. 30: Velocity vectors in Gaia Sky

- **Number factor** – control how many velocity vectors are rendered. The stars are sorted by magnitude (ascending) so the brightest stars will get velocity vectors first.
- **Length factor** – length factor to scale the velocity vectors.
- **Color mode** – choose the color scheme for the velocity vectors:
  - **Direction** – color-code the vectors by their direction. The vectors $\vec{v}$ are pre-processed ($\vec{v}' = \frac{|\vec{v}|+1}{2}$) and then the $xyz$ components are mapped to the colors $rgb$.
  - **Speed** – the speed is normalized in from $[0, 100] Km/h$ to $[0, 1]$ and then mapped to colors using a long rainbow colormap (see here).

  - **Has radial velocity** – stars in blue have radial velocity, stars in red have no radial velocity.
  - **Redshift from the Sun** – map the redshift (radial velocity) from the sun using a red-to-blue colormap.
  - **Redshift from the camera** – map the redshift (radial velocity) from the current camera position using a red-to-blue colormap.
  - **Solid color** – use a solid color for all arrows.
- **Show arrowheads** – Whether to show the vectors with arrow caps or not.

**Hint:** Control the width of the velocity vectors with the **line width** slider in the **visual settings** pane.

### Visual settings pane

**Hint:** Expand and collapse the visual settings pane by clicking on the bolt button or with *l*.

The **visual settings** pane contains a few options to control the shading of stars and other elements:

- **Brightness power** – exponent of power function that controls the brightness of stars. Makes bright stars brighter and faint stars fainter.
- **Star brightness** – control the brightness of stars.
- **Star size (px)** – 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.
- **Line width** – control the width of all lines in Gaia Sky (orbits, velocity vectors, etc.).
- **Label size** – control the size of the labels.
- **Elevation multiplier** – scale the height representation.

### Datasets pane

**Hint:** Expand and collapse the datasets pane by clicking on the hard disk button or with *d*.

The **datasets pane** contains all the datasets currently loaded. For each dataset, a highlight color can be defined. The dataset visual settings window can be used to modify the particle aspect, highlighting properties or the transition limits.

It is also possible to define arbitrary filters on any of the properties of the elements of the dataset, and to add arbitrary affine transformations. Datasets can be highlighted by clicking on the little crosshair below the name.

Please see the *datasets pane section* for more information on this.

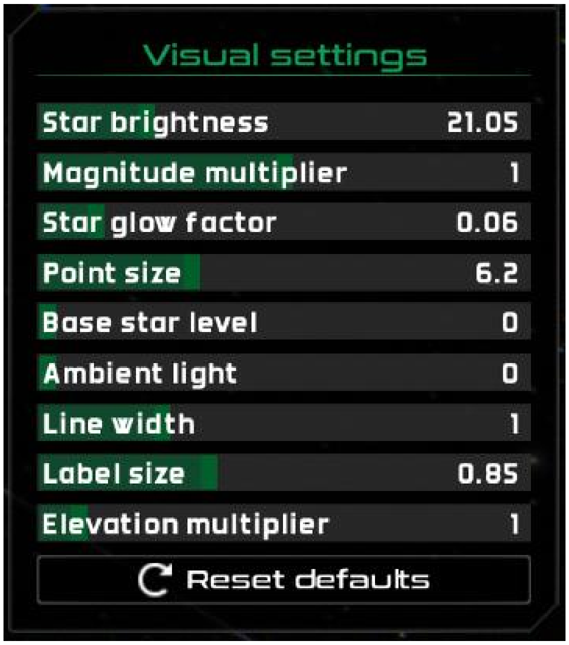



Fig. 31: The visual settings pane with all its sliders.


Fig. 32: The datasets pane with three datasets (Milky Way, Gaia DR3 tiny, BH2 system).

### Location log pane

**Hint:** Expand and collapse the location log pane by clicking on the map marker button.

Gaia Sky keeps track of the visited locations during a session, up to 200 entries. More information on the location log can be found in the *location log section*.


Fig. 33: The location log pane keeps track of the objects you have visited.

### Bookmarks pane

**Hint:** Expand and collapse the bookmarks pane by clicking on the bookmark button or with *b*.

Gaia Sky offers a bookmark system to keep your favorite objects organized and at hand. This panel centralizes the operation of bookmarks. You can find more information on this in the *bookmarks section*.

### Camera info panel

The **camera info panel**, also known as **focus info panel**, is anchored to the bottom-right of the main window. See the *camera info panel section* for more information.

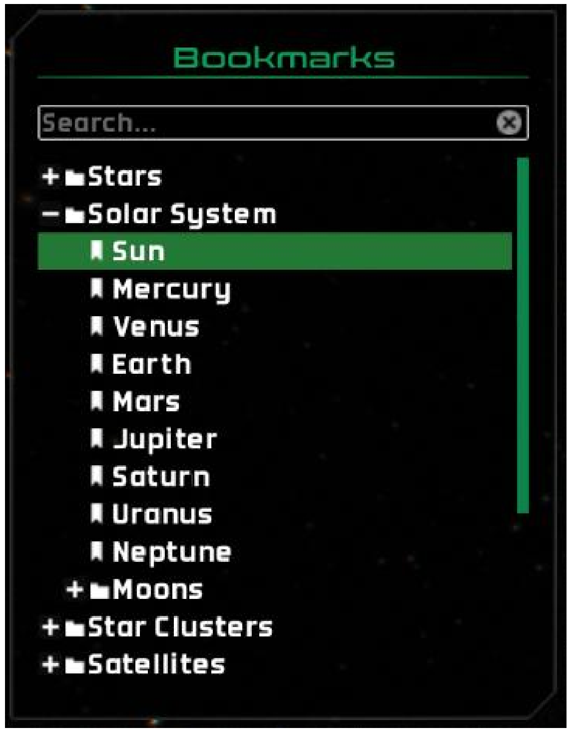



Fig. 34: The bookmarks pane shows the user-defined bookmarks.

### Quick info bar

Anchored to the top of the screen is the quick info bar, which provides the following information at a glance:

- **Simulation date and time** – click it to open the date/time picker window to edit the time.
- **Time warp** – the current speed of time, or "time off" if time is paused.
- Current focus object – the current camera focus object, if any.
- Current closest object – the current object closest to our location.
- Current home object – the home object. This is typically the Earth, but can be changed by editing the *configuration file*.

The colors of the focus, closest and home objects correspond to the colors of the *cross-hairs*. The cross-hairs can be enabled or disabled from the *Interface settings* tab in the preferences window (use *p* to bring it up).

### Action buttons

Anchored to the bottom-left are some buttons to perform some special actions. They are described in the following sub-sections:

- *Minimap*.
- *Load dataset*.
- *Preferences window*.
- *System log*.
- *About/help*.

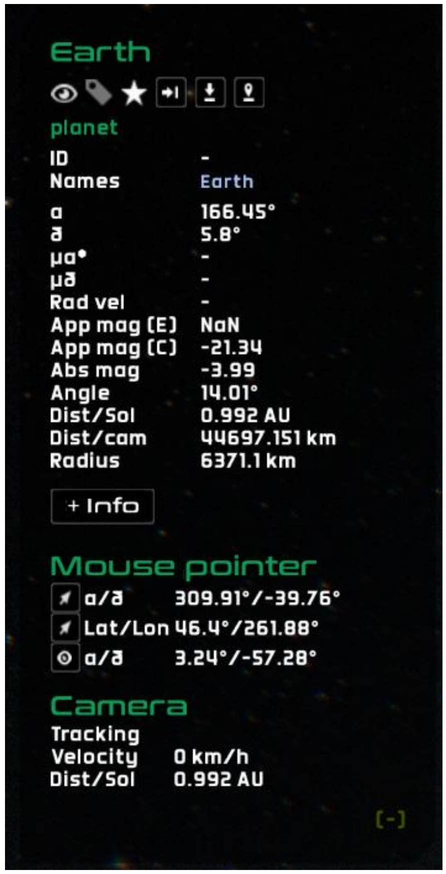



Fig. 35: The camera info pane when the camera is in focus mode. In this state, it is also referred to as focus info pane, and it displays information on the focus (top), the mouse pointer (middle), and the camera position and state (bottom).

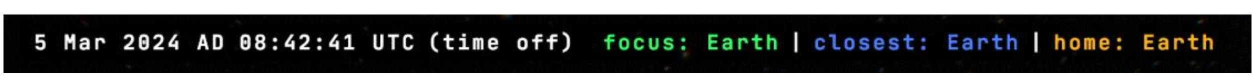



Fig. 36: The quick info bar is anchored to the top of the window and displays useful information at a glance.

- *Exit*.

### Minimap

Use the mini-map button or *Tab* to toggle the mini-map on an off. The mini-map offers a contextual view of your position as a top and side projection, relative to the closest objects and the distance to the Sun.

### Load dataset

Use the open folder button or *Ctrl + o* to load a new VOTable file (`.vot`) into Gaia Sky. The *dataset loading section* contains more information on dataset loading. Also, check out the *STIL data loader* section for more information on the metadata needed for Gaia Sky to parse the dataset correctly.

### Preferences window

Use the preferences button, or *p*, to bring up the preferences window, from which the settings and configuration can be modified. For a detailed description of the configuration options refer to the *Configuration Instructions*.

### System log

Use the log button, or *Alt + l*, to bring up the system 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

Use the help button, or *h*, to bring 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*.

### System info panel

Bring up the system info panel by hitting *Ctrl + d*, or by using the *Show debug info* checkbox in the *System* tab in the preferences window. The *system information panel section* contains more information on this topic.

## 1.4.4 Camera settings

The camera settings are accessed via the *camera pane*. This section describes the two main settings that affect how the camera behaves: *camera modes* and *camera behaviors*.

**Contents**


- *Camera settings*
  - *Camera modes*
    - *Focus mode*
      - *Object tracking*
    - *Free mode*
    - *Game mode*
    - *Spacecraft mode*
  - *Camera behaviors*
    - *Cinematic behavior*
    - *Non-cinematic behavior*


### Camera modes

Gaia Sky offers four basic camera modes.

**Hint:** The 'Gaia scene' camera mode has been removed in Gaia Sky 3.2.2. The three 'Gaia FOV' modes have been removed after Gaia Sky 3.5.4-1.

#### 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.

### Object tracking

Usually, in focus mode, the direction of the camera points to the focus object. It is possible, however, to track a different object while still having our position linked to the focus object. To do so, **right-click** on the object to track and select '*Track object: Object name*' in the **context menu** that pops up. This will cause the camera direction to automatically follow the tracking object at all times. To disable tracking, right-click anywhere and select '*Remove tracking object*'

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

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

**Hint:** *Numpad 1* or *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*

**Hint:** *Numpad 0* or *0* – enter free mode

### Game mode

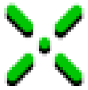
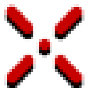
This mode maps the standard control system for most games (*wasd* + Mouse look) in Gaia Sky. Additionally, it is possible to add gravity to objects, so that when the camera is closer to a planet than a certain threshold, gravity will pull it to the ground.Quit

**Hint:** *Numpad 2* or *2* – enter game 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** – shown as a ball with the horizon and other marks. It represents the current orientation of the spacecraft with respect to the equatorial system.
-  – indicate the direction the spacecraft is currently headed to.

-  – indicate direction of the current velocity vector, if any.
-  – indicate 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 .
-  – stabilise the yaw, pitch and roll angles. If rotation is applied during the stabilisation, the stabilisation is canceled.
-  – stop the spacecraft until its velocity with respect to the Sun is 0. If thrust is applied during the stopping, the stopping is canceled.
-  – return to the focus mode.

Additionally, it is possible to adjust three more parameters:

- **Responsiveness** – control how fast the spacecraft reacts to the user's yaw/pitch/roll commands. It could be seen as the power of the thrusters.
- **Drag** – control 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** – make 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*

**Hint:** `NUMPAD_3` – enter spacecraft mode

### Camera behaviors

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

**Hint:** enable and disable the cinematic camera behavior with *ctrl* + *c*.

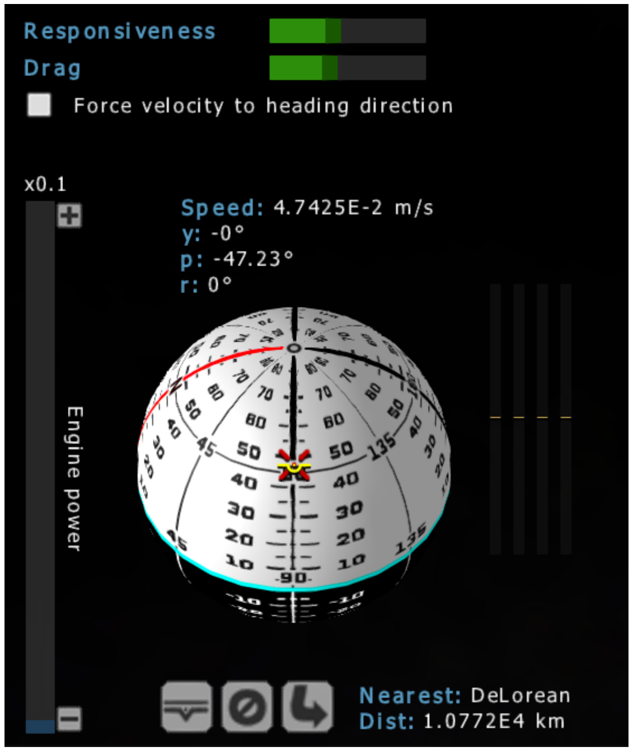



Fig. 37: Spacecraft mode controls view, with the attitude indicator ball at the center, the control buttons at the bottom and the engine power to the left.

#### Cinematic behavior

This behavior 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 behavior

In this behavior 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** behavior as of version `1.5.0` and probably better meets the expectations of new users.

### 1.4.5 Search objects

**Hint:** You can search objects by pressing *f*, */* or *Shift* + *f* at any time.

You can look up any object by name by pressing the search key binding (see info box above). This brings up the search dialog and focuses the search input field. Just enter the name of the object in that input field and Gaia Sky will focus it immediately if there is an exact match. Otherwise, search suggestions are shown as you type, with the most relevant results at the top. Use *Tab* to cycle between them, and *Enter* to focus on the current selection.

A successful search puts the camera in *focus mode*.

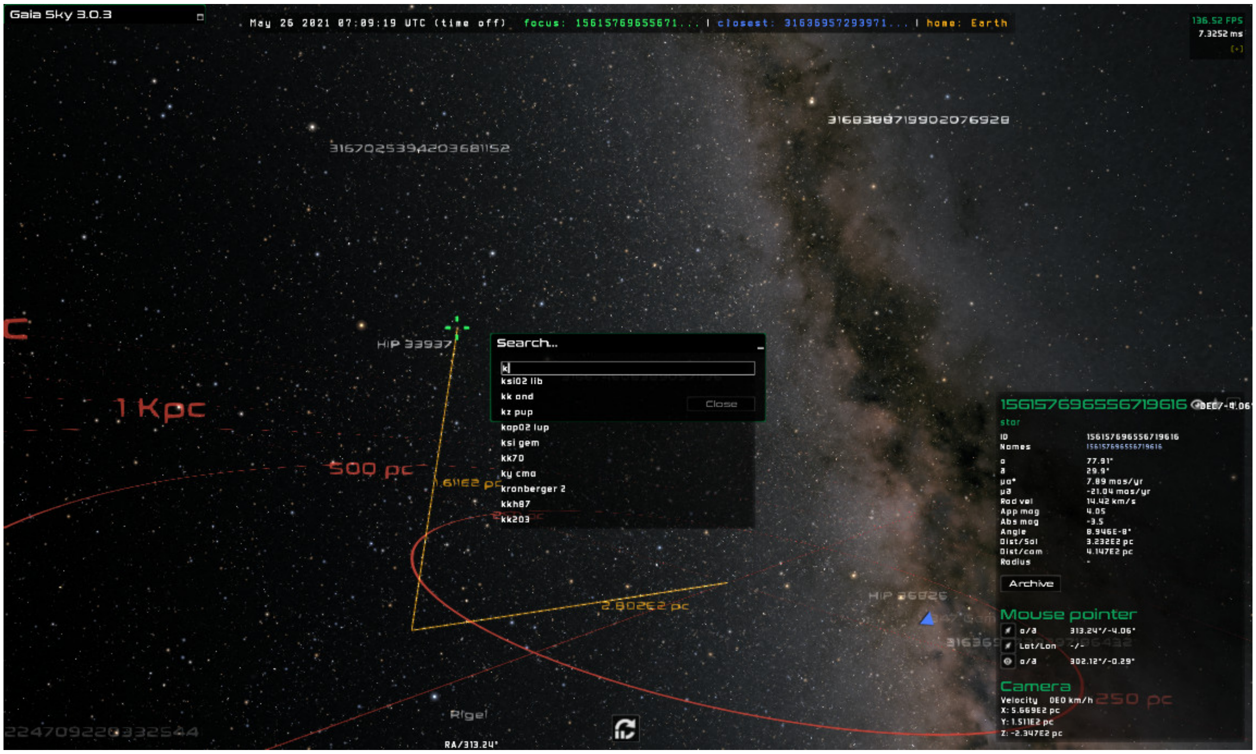



Fig. 38: The search dialog in Gaia Sky

### 1.4.6 Camera info panel

The **camera info panel**, also known as **focus panel**, is anchored to the bottom-right of the main window.

**Contents**


- *Camera info panel*
  - *Focus pane*
  - *Mouse pointer*
  - *Camera*


Whenever the camera is in *focus mode*, information about the current focus is displayed here. Additionally, the current location of the mouse pointer and the speed and coordinates of the camera in the *internal reference system* are also shown at the bottom.

The camera info panel contains three blocks when the camera is in focus mode. When it is in free mode, only the two bottom blocks are available:

- Focus information, at the top. Only shown when the camera is in focus mode.
- Mouse pointer information, in the middle.
- Camera information, to the bottom.

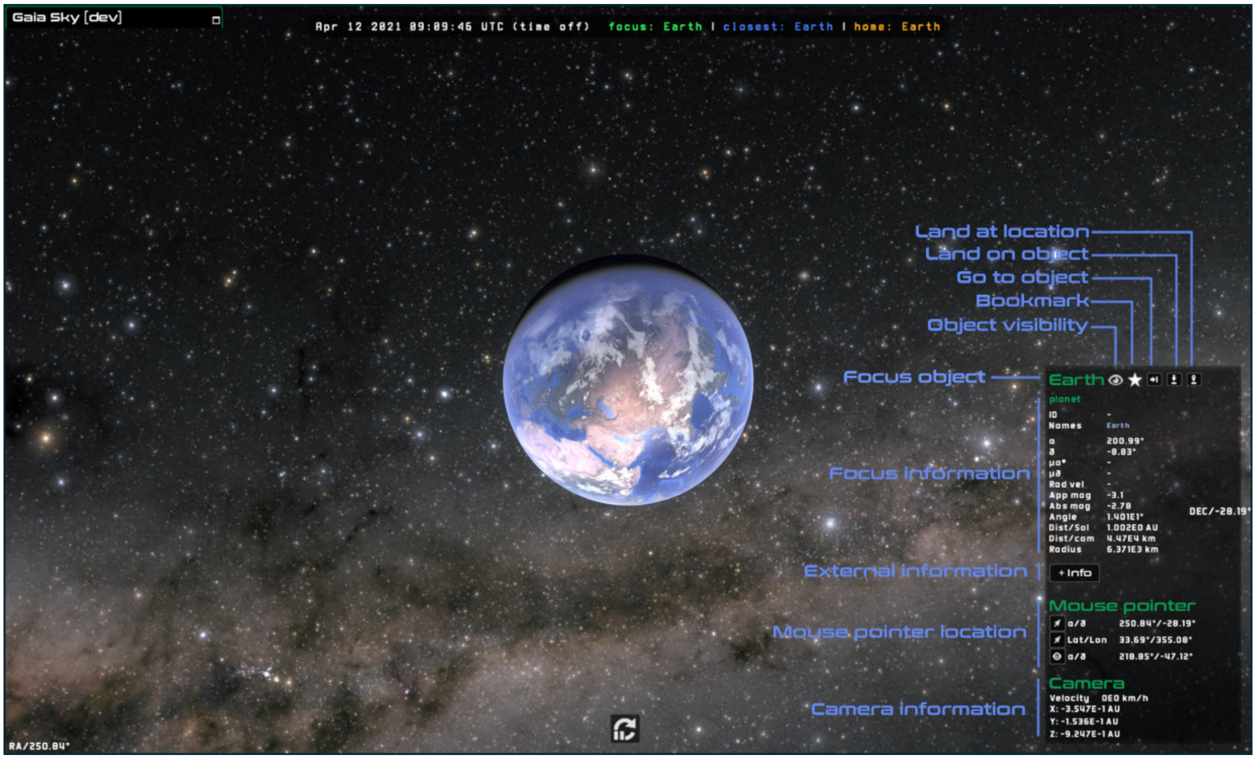



Fig. 39: The camera info panel in Gaia Sky

## Focus pane

The top line of the focus pane contains the name and type of the current focus object (in your theme accent color, green in the screenshot above) plus some icons:

- toggle the visibility of this object on and off.
- toggle the 'always show label' flag for this object, so that its label is always shown regardless of the object's solid angle.
- add this object to the *bookmarks*.
- moves the camera to the object with a smooth transition.
- land on the object.
- open the window to choose a location to land on, and execute the landing.

The information items contained in the focus pane are updated in real time, and are the following:

- Object type.
- **ID** – object ID.
- **Names** – object names, as a list.

- $\alpha$ – right ascension ($\alpha$), in degrees.
- $\delta$ – declination ($\delta$), in degrees.
- $\mu_{\alpha\star}$ – proper motion in alpha ($\mu_{\alpha\star}$), in mas/s.
- $\mu_{\delta}$ – proper motion in delta ($\mu_{\delta}$), in mas/s.
- **Rad vel** – radial velocity, in km/s.
- **App mag (E)** – apparent magnitude as seen from Earth.
- **App mag (C)** – apparent magnitude as seen from the current camera position.
- **Abs mag** – absolute magnitude.
- **Angle** – current solid angle. For stars, this involves a lot of guess-work. See the *star rendering section* for more information.
- **Dist/sol** – distance from the focus object to the Sun.
- **Dist/cam** – distance from the focus object to the camera.
- **Radius** – radius of the object, in km.
- + *Info* button – lists all the local data on the object, and offers a preview of the Wikipedia article for this object, if it exists. If the object belongs to a VOTable catalog and has additional columns, those are displayed here as well.
- *Archive* button – provides the archive data for the given star. Only works for Gaia and Hipparcos stars.
- Simbad link, opens in a browser with the object information in the Simbad database.

### Mouse pointer

This section contains the current location of the mouse pointer in the equatorial reference system, as sky-projected coordinates. Additionally, when the pointer is over a planet or moon, we already get the longitude and latitude values.

- $\alpha$ / $\delta$ (pointer) – the current location of the mouse pointer in the equatorial reference system (sky coordinates).
- **LatLon** – the latitude and longitude of the mouse pointer on the surface of a planet or moon. Only updated when the mouse pointer is on a planet or moon.
- $\alpha$ / $\delta$ (view) – the current location of the center of the view in the equatorial reference system (sky coordinates).

### Camera

This section contains the current speed of the camera in Km/h, plus the distance from the camera to the Sun and the current location of the camera in the *internal reference system* (equatorial cartesian coordinates).

- **Tracking** – the name of the object the camera is currently tracking, if any.
- **Velocity** – current camera velocity.
- **Dist/Sol** – distance from the camera to the Sun.

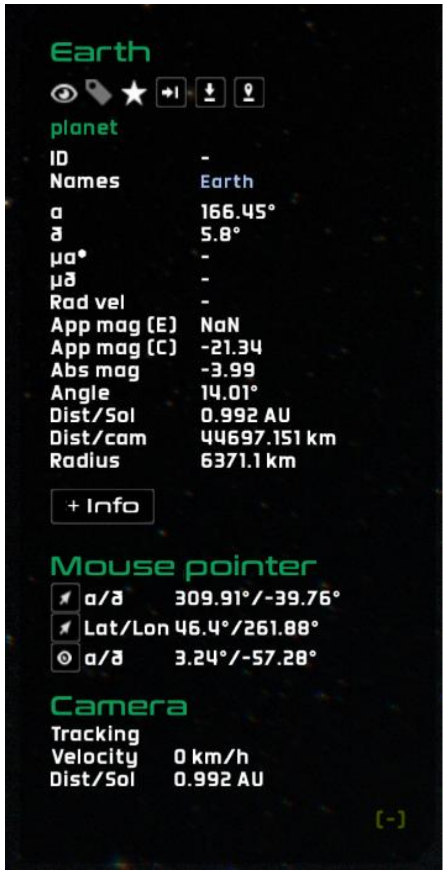



Fig. 40: The camera info panel when the camera is in focus mode. In this state, it is also referred to as focus info pane, and it displays information on the focus (top), the mouse pointer (middle), and the camera position and state (bottom).


Fig. 41: The camera info panel when the camera is in free mode only provides information on the status, velocity and location of the camera, as well as the mouse pointer.

### 1.4.7 Object visiblity

Gaia Sky offers two different ways to control object visiblity: *type visibility* and *per-object visibility*. The main difference is that, while type visibility acts on all objects of a given component type (think stars, labels, grids, planets, etc.), per-object visibility is capable of hiding and showing individual objects.

You can toggle **object types** on and off by clicking on their icon in the `Type visibility` pane in the control panel, as described *here*.

You can also hide and show **individual objects** by using the eye icon in the focus panel (when in focus mode) or by using the dedicated window, as described in the *individual visibility section*.

### 1.4.8 Datasets

Gaia Sky supports the loading and visualization of datasets and catalogs (used interchangeably in this document) of different nature. Catalogs are groups of similar objects that are loaded and displayed at once.

**Contents**


- *Datasets*
  - *Preparing datasets*
  - *Loading datasets*
    - *Star catalogs*
    - *Particle datasets*
    - *Star cluster catalogs*
    - *Variable star catalogs*
  - *Datasets pane*
    - *Dataset highlighting*
    - *Dataset visual settings*
    - *Dataset filters*
    - *Dataset transformations*
    - *Dataset inforamtion*

### Preparing datasets

Please see the *STIL data loader section* for information about how to prepare the datasets for Gaia Sky.

### Loading datasets

Catalogs and 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 catalog, click on the folder icon in the controls window or press *ctrl* + *o* and choose the file you want to load. Supported formats are `.csv` (Comma-Separated Values), `.vot` (VOTable) and FITS (as of `3.0.2`). Once the dataset has been chosen, a new window is presented to the user asking the type of the dataset and some extra options associated with it. This window is also presented when loading a dataset via SAMP.

**Hint:** As of version `3.0.2`, Gaia Sky supports interactive loading of FITS files.

Datasets can be **star** catalogs, **particle** datasets, **star cluster** datasets, or **variable star** catalogs, depending on whether the new dataset contains stars (with magnitudes, colors, proper motions and whatnot), just particles (only 2D or 3D positions and extra attributes), clusters (with properties like the visual radius) or variable stars (with light curves and periods).

Please, see the *Preparing catalogs* for more information on how to prepare the datasets for Gaia Sky.

### Star catalogs

Star catalogs are expected to contain some attributes of stars, like magnitudes, color indices, proper motions, etc., and use the regular star shaders to render the stars. When selecting star datasets, there are a couple of settings available:

- **Dataset name** – the name of the dataset.
- **Magnitude scale factor** – subtractive scaling factor to apply to the magnitude of all stars (`appmag = appmag - factor`).
- **Label color** – the color of the labels of the stars in the dataset.
- **Fade in** – these are two distances from the Sun, in parsecs, that will be used as interpolation limits to fade in the whole dataset. The dataset will not be visible if the camera distance from the Sun is smaller than the lower limit, and it will be fully visible if the camera distance from the Sun is larger than the upper limit. The opacity is interpolated between 0 and 1 if the camera distance from the Sun is larger than the lower limit and smaller than the upper limit.
- **Fade out** – these are two distances from the Sun, in parsecs, that will be used as interpolation limits to fade out the whole dataset. The dataset will not be visible if the camera distance from the Sun is larger than the upper limit, and it will be fully visible if the camera distance from the Sun is smaller than the lower limit. The opacity is interpolated between 1 and 0 if the camera distance from the Sun is larger than the lower limit and smaller than the upper limit.

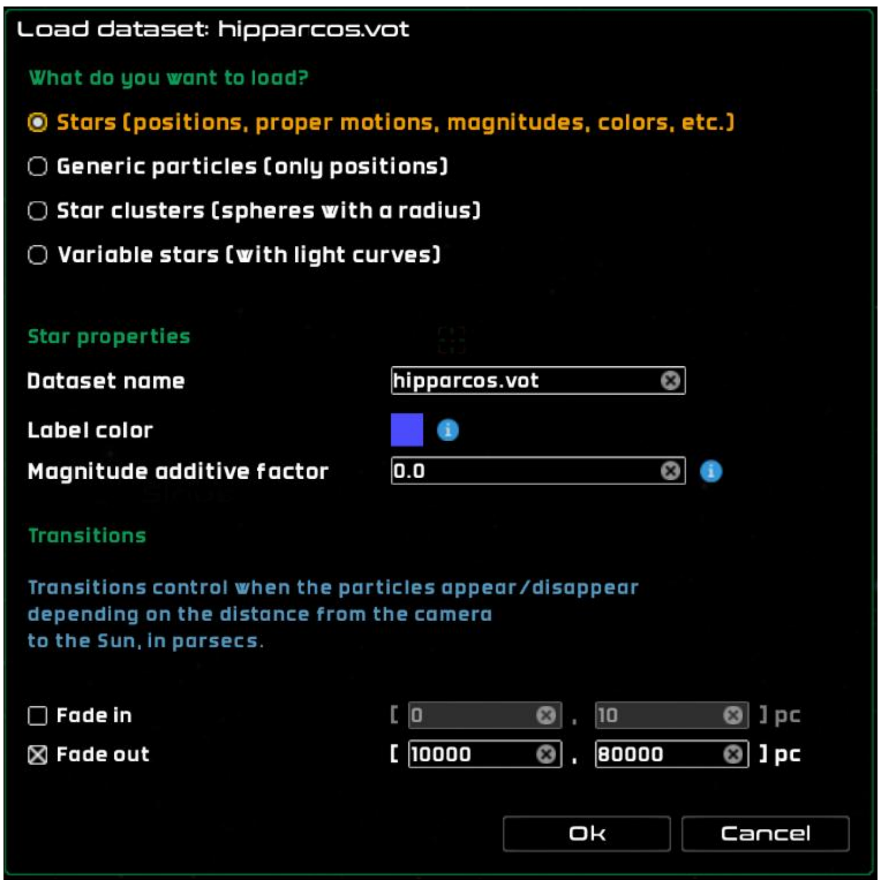



Fig. 42: Loading a star catalog

### Particle datasets

Particle datasets only require positions to be present, and use generic shaders to render the particles. Some parameters can be tweaked at load time to control the appearance and visibility of the particles:

- **Dataset name** – the name of the dataset.
- **Particle color** – the color of the particles. Can be modified with the particle color noise.
- **Particle color noise** – a value in [0,1] that controls the amount of noise to apply to the particle colors in order to get slightly different colors for each particle in the dataset.
- **Label color** – color of the label of this dataset. Particles themselves do not have individual labels.
- **Particle size** – the size of the particles, in pixels.
- **Minimum solid angle** – the minimum solid angle (in radians) used to represent this particle. This is a minimum bound on the size of the particles.
- **Maximum solid angle** – the maximum solid angle (in radians) used to represent this particle. This is a maximum bound on the size of the particles.
- **Number of labels** – the number of labels to render for this dataset. Set to 0 to render no labels.
- **Profile decay** – a power that controls the radial profile of the actual particles, as in `(1-d)^pow`, where `d` is the distance from the center to the edge of the particle, in [0,1].
- **Component type** – the component type to apply to the particles to control their visibility. Make sure that the chosen component type is enabled in the Visibility pane.
- **Fade in** – these are two distances from the Sun, in parsecs, that will be used as interpolation limits to fade in the whole dataset. The dataset will not be visible if the camera distance from the Sun is smaller than the lower limit, and it will be fully visible if the camera distance from the Sun is larger than the upper limit. The opacity is interpolated between 0 and 1 if the camera distance from the Sun is larger than the lower limit and smaller than the upper limit.
- **Fade out** – these are two distances from the Sun, in parsecs, that will be used as interpolation limits to fade out the whole dataset. The dataset will not be visible if the camera distance from the Sun is larger than the upper limit, and it will be fully visible if the camera distance from the Sun is smaller than the lower limit. The opacity is interpolated between 1 and 0 if the camera distance from the Sun is larger than the lower limit and smaller than the upper limit.

### Star cluster catalogs

Star cluster catalogs can also be loaded directly from the UI as of Gaia Sky 2.2.6. The loader also uses STIL to load CSV or VOTable files. In CSV mode the units are fixed, otherwise they are read from the VOTable, if it has them. The order of the columns is not important. The required columns are the following:

- `name`, `proper`, `proper_name`, `common_name`, `designation` – one or more name strings, separated by '|'.
- `ra`, `alpha`, `right_ascension` – right ascension in degrees.
- `dec`, `delta`, `de`, `declination` – declination in degrees.
- `dist`, `distance` – distance to the cluster in parsecs, or
- `pllx`, `parallax` – parallax in mas, if distance is not provided.
- `rcluster`, `radius` – the radius of the cluster in degrees.

Optional columns, which default to zero, include:

- `pmra`, `mualpha`, `pm_ra` – proper motion in right ascension, in mas/yr.

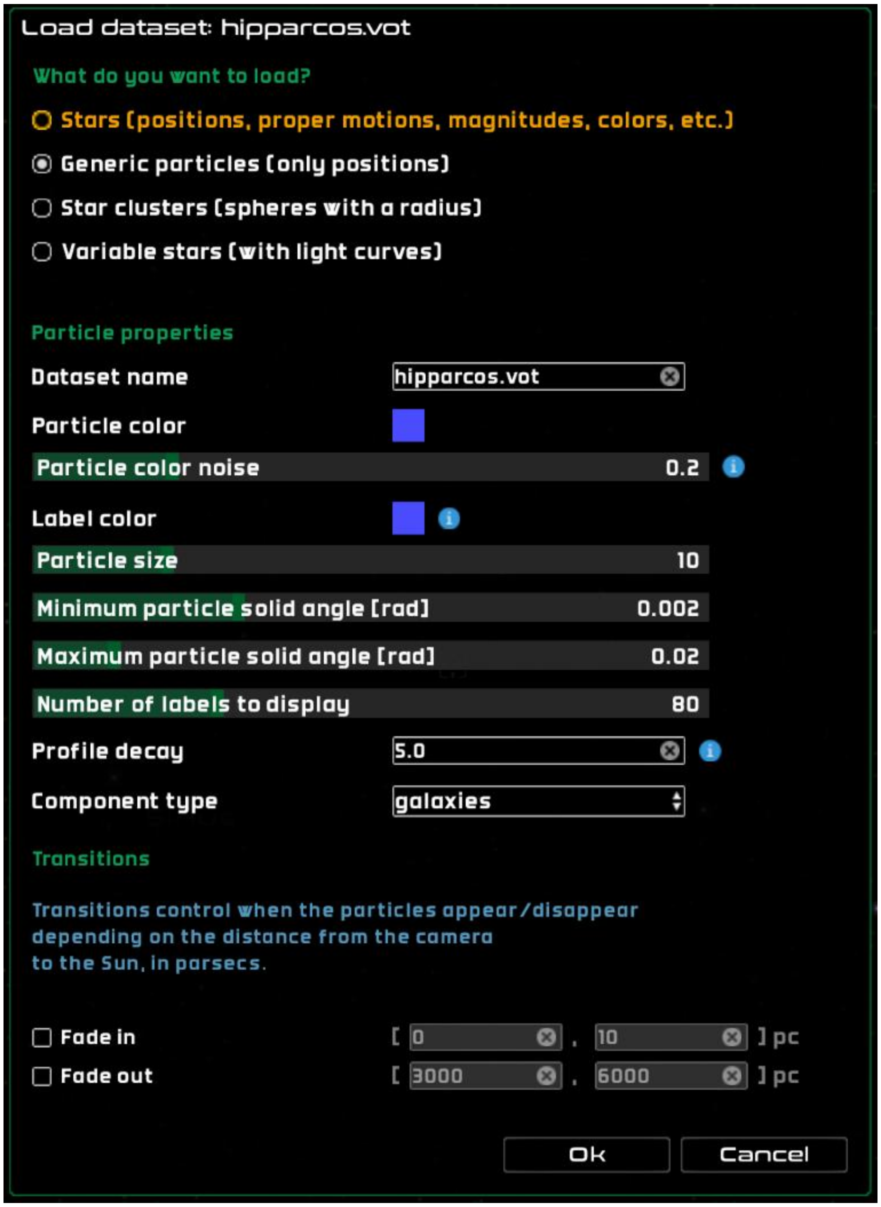



Fig. 43: Loading a point cloud dataset

- `pmdec`, `mudelta`, `pm_dec` – proper motion in declination, in mas/yr.
- `rv`, `radvel`, `radial_velocity` – radial velocity in km/s.

Star cluster datasets require positions and radii to be present, and use wireframe spheres to render the clusters. The parameters that can be tweaked at load time are:

- **Dataset name** – the name of the dataset.
- **Particle color** – the color of the clusters and their labels.
- **Label color** – color of the label of this dataset. Particles themselves do not have individual labels.
- **Component type** – the component type to apply to the particles to control their visibility. Make sure that the chosen component type is enabled in the Visibility pane.
- **Fade in** – these are two distances from the Sun, in parsecs, that will be used as interpolation limits to fade in the whole dataset. The dataset will not be visible if the camera distance from the Sun is smaller than the lower limit, and it will be fully visible if the camera distance from the Sun is larger than the upper limit. The opacity is interpolated between 0 and 1 if the camera distance from the Sun is larger than the lower limit and smaller than the upper limit.
- **Fade out** – these are two distances from the Sun, in parsecs, that will be used as interpolation limits to fade out the whole dataset. The dataset will not be visible if the camera distance from the Sun is larger than the upper limit, and it will be fully visible if the camera distance from the Sun is smaller than the lower limit. The opacity is interpolated between 1 and 0 if the camera distance from the Sun is larger than the lower limit and smaller than the upper limit.

### Variable star catalogs

Variable stars are represented in Gaia Sky by displaying the changing magnitude visually in the scene when time is on. These datasets are expected to contain a time series (magnitudes vs times) and a period. Only variable stars with a period are loaded, the rest are discarded.

See the *STIL data provider section* for more information on how to prepare variable star datasets for Gaia Sky.

- **Dataset name** – the name of the dataset.
- **Magnitude scale factor** – subtractive scaling factor to apply to the magnitude of all stars (`appmag = appmag - factor`).
- **Label color** – the color of the labels of the stars in the dataset.
- **Fade in** – these are two distances from the Sun, in parsecs, that will be used as interpolation limits to fade in the whole dataset. The dataset will not be visible if the camera distance from the Sun is smaller than the lower limit, and it will be fully visible if the camera distance from the Sun is larger than the upper limit. The opacity is interpolated between 0 and 1 if the camera distance from the Sun is larger than the lower limit and smaller than the upper limit.
- **Fade out** – these are two distances from the Sun, in parsecs, that will be used as interpolation limits to fade out the whole dataset. The dataset will not be visible if the camera distance from the Sun is larger than the upper limit, and it will be fully visible if the camera distance from the Sun is smaller than the lower limit. The opacity is interpolated between 1 and 0 if the camera distance from the Sun is larger than the lower limit and smaller than the upper limit.

The process by which light curves are loaded and used in Gaia Sky is a bit involved and outlined below:

1. First, we check that time series (magnitudes v times) and periods are actually present in the file.
2. Then, *NaN* values are removed from the light curve data points.

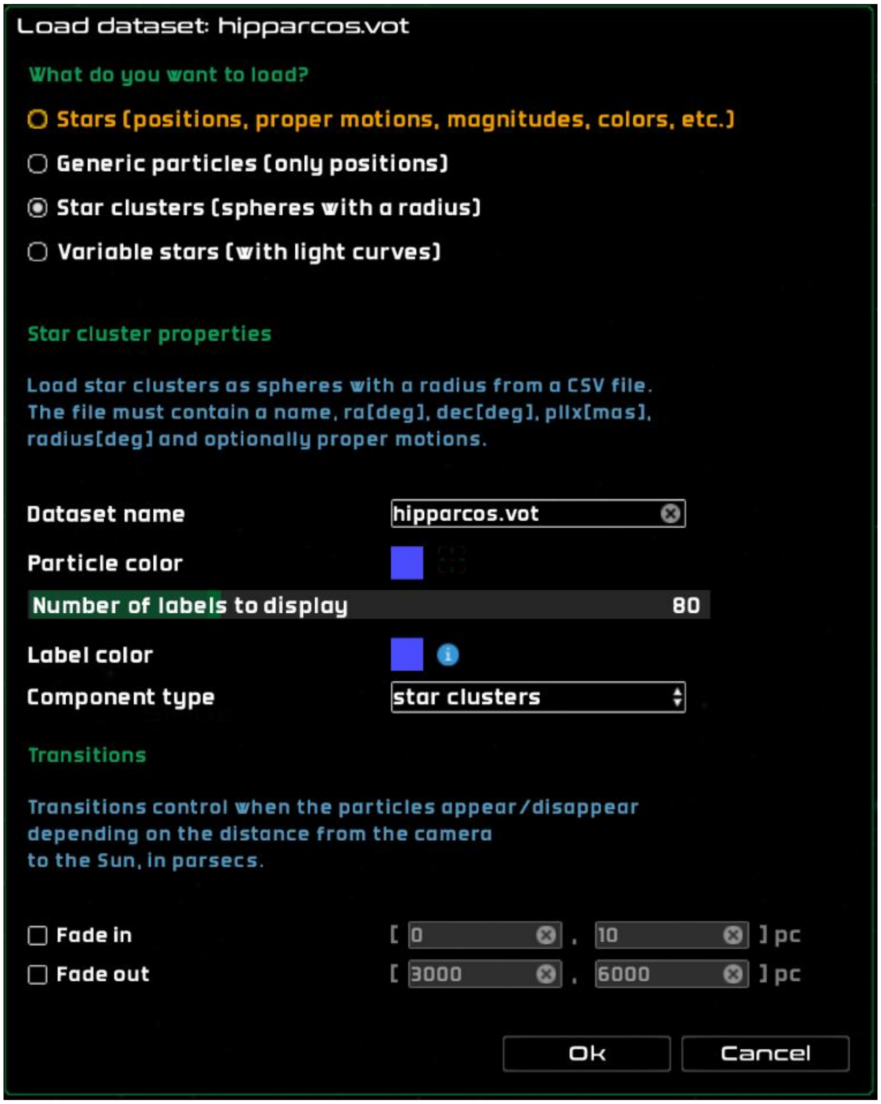



Fig. 44: Loading a star cluster catalog

3. We fold the time series into a phase diagram using the period and sort the result accordingly with the phase for each data point.
4. Due to a GPU memory trade-off (the time series data must be sent to the GPU for each star, and all stars must have the same in-memory size in the GPU), we have a limitation of 20 data points per star. If the number of incoming data points is larger than 20, we re-sample the phase diagram.
5. Finally, the magnitudes are converted to pseudo-sizes for easier representation, and passed on to the model.

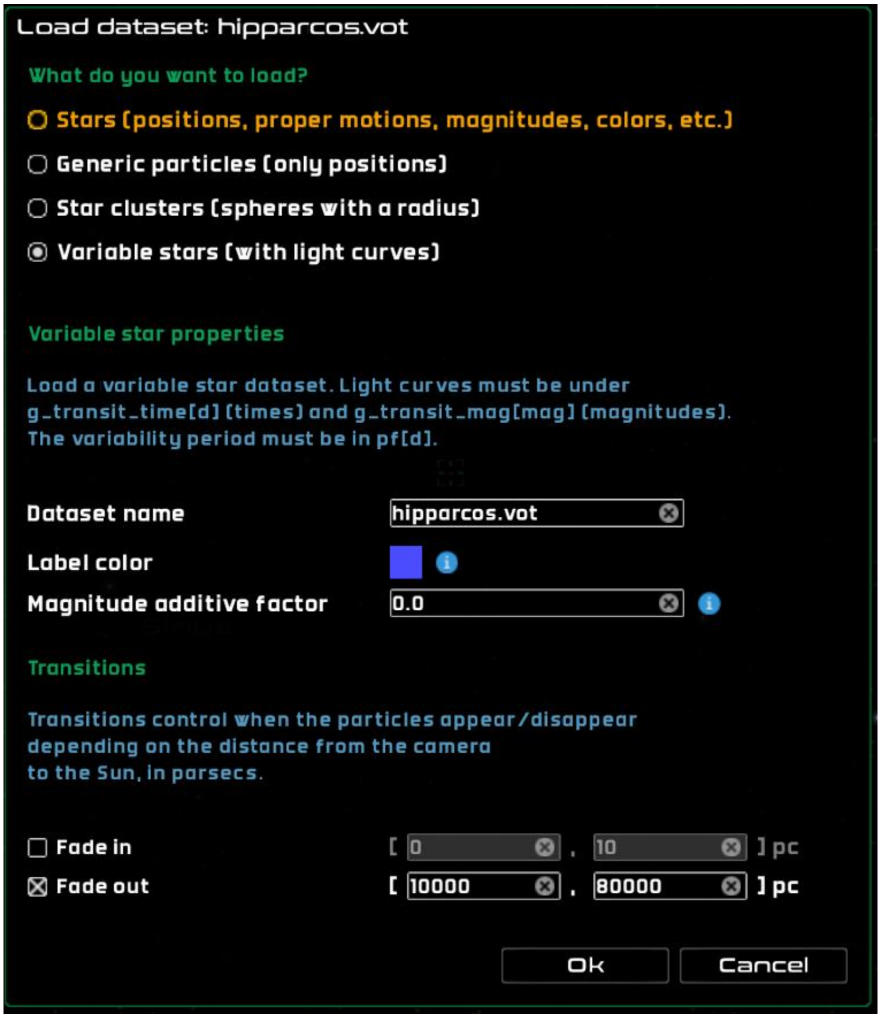



Fig. 45: Loading a variable star catalog

### Datasets pane

You can find a list of all datasets currently loaded in the *Datasets* pane, anchored to the top-left of the screen. You can bring it up automatically by pressing *d*.

Each dataset has a panel that can be expanded by clicking on the ▶ icon by the dataset name. Once expanded, a dataset panel can be collapsed with ▼.

The dataset panel, once expanded, contains a few controls that depend on the type of dataset, and that allow the user to modify some settings about how the dataset is displayed. These controls are in the topmost line in the dataset pane.

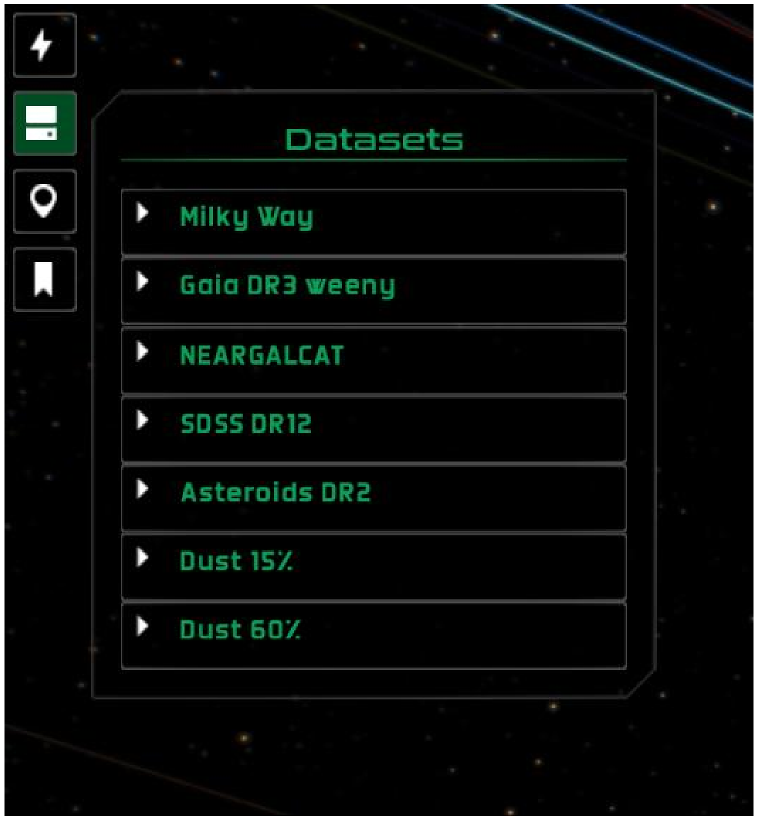



Fig. 46: Datasets pane in Gaia Sky


Fig. 47: Dataset panel in the datasets pane for the 'Gaia DR3 weeny' catalog

From left to right, the controls are the following:

- – toggle the visibility of the dataset. This makes the whole dataset appear and disappear.
- – highlight the dataset using the current color and particle size. The **color** *can be changed* by clicking on the rightmost button (blue square in the image above), and the particle size factor can be adjusted from the dataset visual settings window. Datasets can also be color-mapped. Only star, particle, LOD and orbital elements datasets can be highlighted.
- – open the *dataset visual settings* window.
- – open the *dataset filters* window.
- – open the *dataset affine transformations* window.
- – open the *dataset information* window.
- – delete the dataset.

After the controls, we can find some information:

- The type of dataset, in gray.
- The dataset description, if any. Move your mouse to the small (i) symbol to get the full description in a tooltip.
- The number of objects in the dataset, in blue.

### Dataset highlighting

Datasets can be highlighted by clicking on the target icon . When highlighted, the colors of the particles change according to the highlighting color or color map selected (see below), and the particles may also become larger or smaller depending on the settings in the highlight section of the *visual settings dialog*.

To the right of the dataset pane 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.

A **color map** can be selected by clicking on the "Color map" radio button, and 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

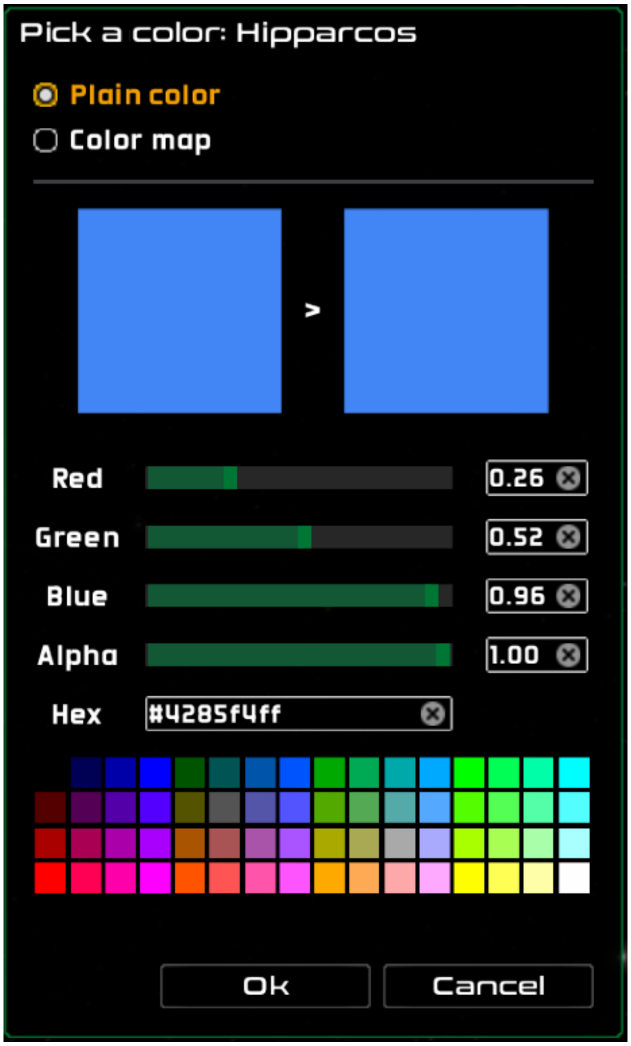



Fig. 48: The highlighting plain color picker dialog

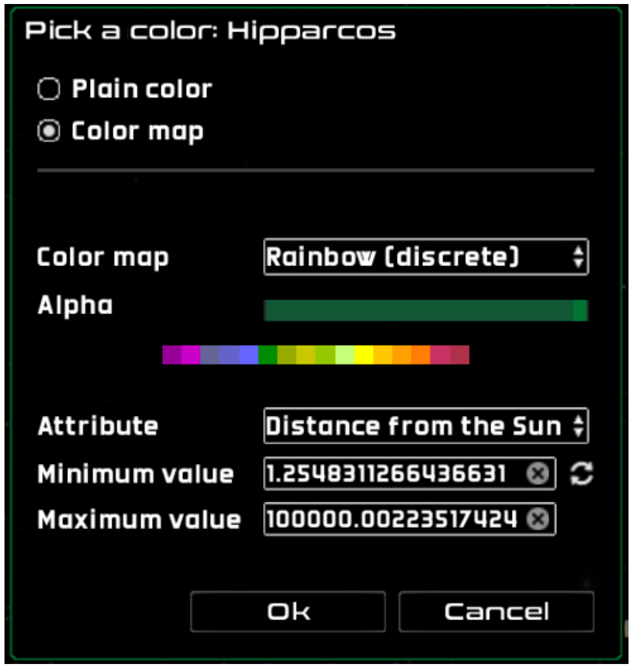



Fig. 49: The highlighting color map dialog

### Dataset visual settings

Open the dataset visual settings window by clicking on the bolt icon . There are three sections, named **particle aspect**, **highlighting** and **transitions**.

In the **particle aspect** section we can find the following controls:

- **Point size** – this slider controls the dataset point size. This acts as a factor on the actual size of the particles of the dataset.
- **Minimum particle solid angle [rad]** – only present in particle datasets, this slider controls the minimum visual solid angle of each particle.
- **Maximum particle solid angle [rad]** – only present in particle datasets, this slider controls the maximum visual solid angle of each particle.

In the **highlighting** section, we can find the following properties:

- **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.

In the **Transitions** section, we can define fade-in and fade-out rules depending on the distance from the camera to the center of the dataset, or to the center of the reference system.

- **Fade in** – this check box enables the fade-in transitions, where the dataset opacity goes from 0 (invisible) to 1 (fully visible), mapped to the user given-distances in parsecs.
- **Fade out** – this check box enables the fade-out transitions, where the dataset opacity goes from 1 (fully visible) to 0 (invisible), mapped to the user given-distances in parsecs.

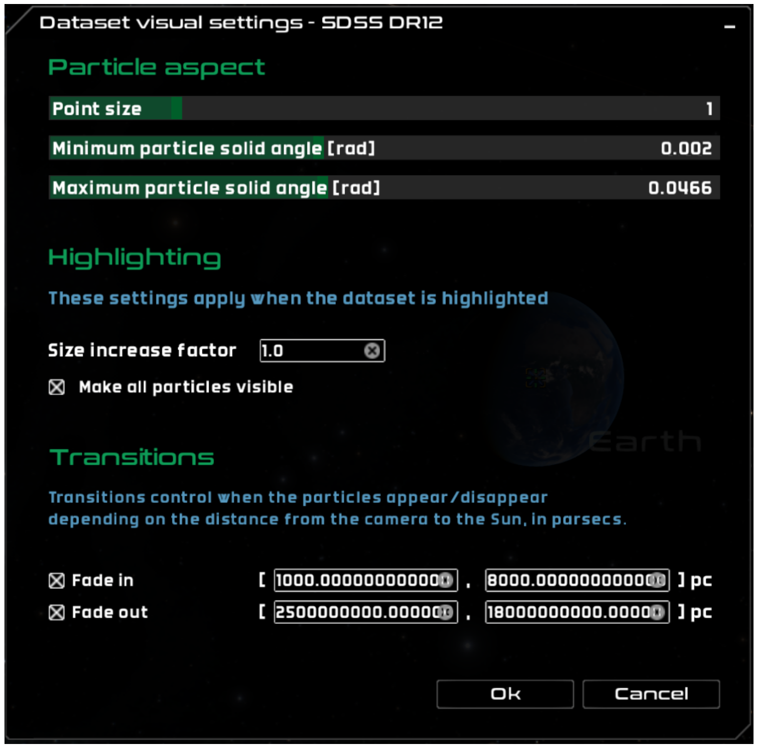



Fig. 50: The dataset visual settings dialog

### Dataset filters

Open the dataset filters window by clicking on the code icon . Filters are **only available to particle, stars and LOD datasets**.

This dialog allows for the creation of arbitrary selection filters by setting conditions (rules) on 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 type and loading method.

Click on the *Add filter* button to add a filter, and use *Add rule* to add new rules to the current filter. The *Rules operator* select box enables the selection of the logical operator. Then, each rule contains the attribute, the comparator operation (<, <=, >, >=, ==, !=) and a value. Use the bin icon to delete a rule.

### Dataset transformations

The dataset transformations window (open it by clicking on the matrix icon ) enables the definition of arbitrary affine transformations (only translation, rotation and scaling available, plus reference system transforms) and application to the datasets in real time. Transformations are available to all datasets, but **only particles in groups will be affected**. Single objects (models, single stars, planets, moons, etc.) that are part of a dataset are not applied the transformations.

Transformations are defined in a sequence. Each transformation is represented by a matrix. The matrices are multiplied in the defined order. This means that the **transformations are actually applied last-to-first**. If you want to rotate a dataset, and then translate it, you need to first define a translation and then a rotation.

Add a new transformation by clicking on the *Add transformation* button. Once the transformation appears, there are a few settings you can change:

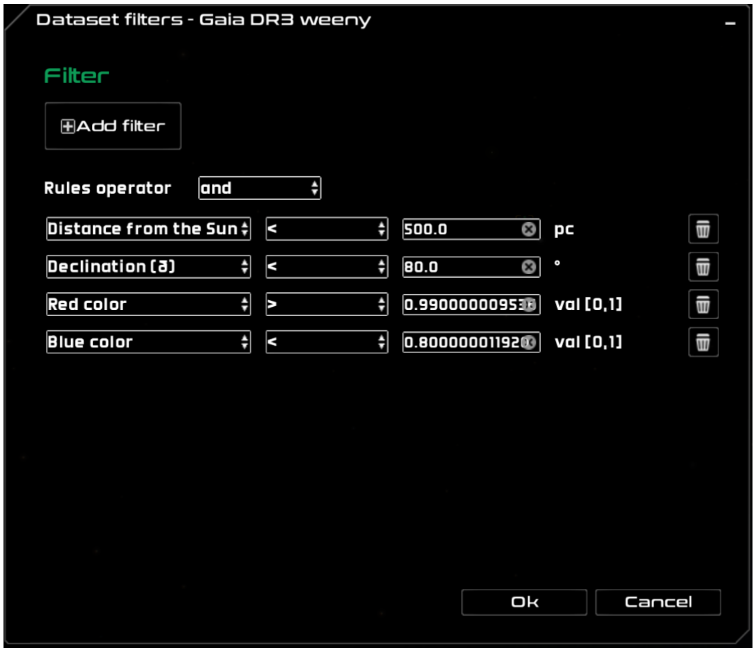



Fig. 51: The dataset filters dialog

- **Type** – select the transformation type: **translation**, **rotation**, **scaling** or **reference system**.
-  – move the transformation up in the chain.
-  – move the transformation down in the chain.
-  – remove the transformation.

For each transformation type we have different inputs:

- **Translation** – choose the X, Y and Z of your translation vector, in parsecs.
- **Rotation** – choose the rotation axis X, Y and Z components, plus the rotation angle, in degrees.
- **Scaling** – choose the scaling factor in X, Y and Z. No units here.
- **Reference system** – select the reference system transformation you want to apply from the select box. The possible transformations are:
- Galactic to equatorial
- Equatorial to galactic
- Ecliptic to equatorial
- Equatorial to ecliptic
- Galactic to ecliptic
- Ecliptic to galactic

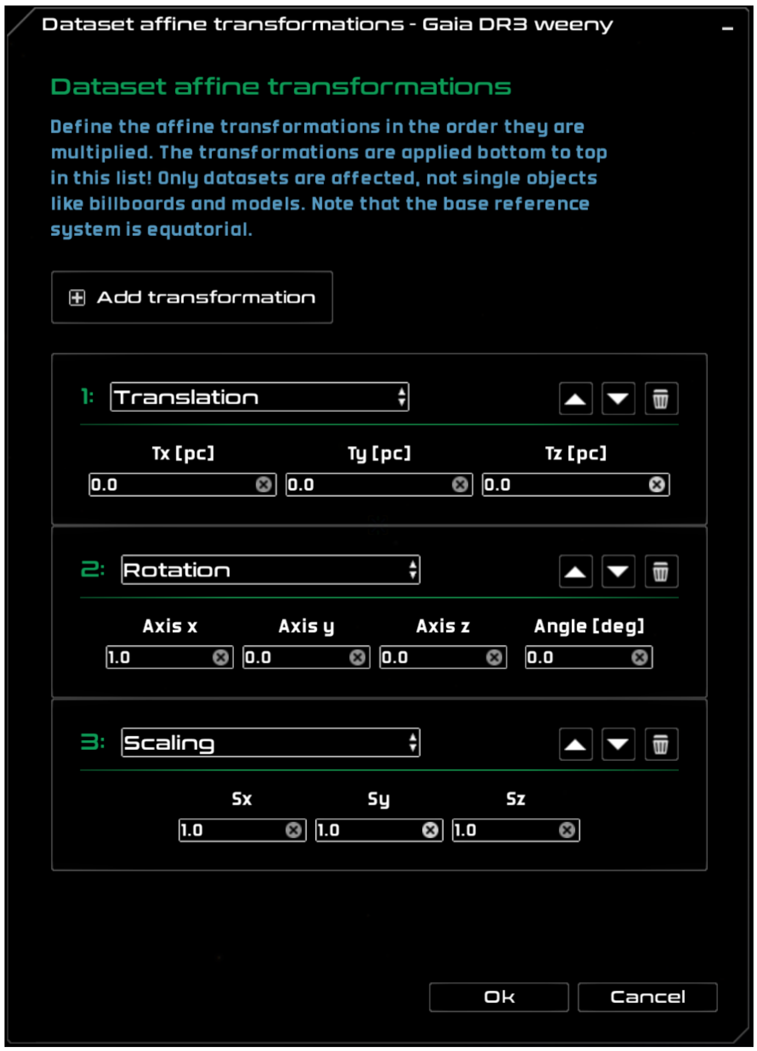



Fig. 52: The dataset transformations dialog

### Dataset inforamtion

Get some additional information on a dataset by clicking on the 'i' icon .

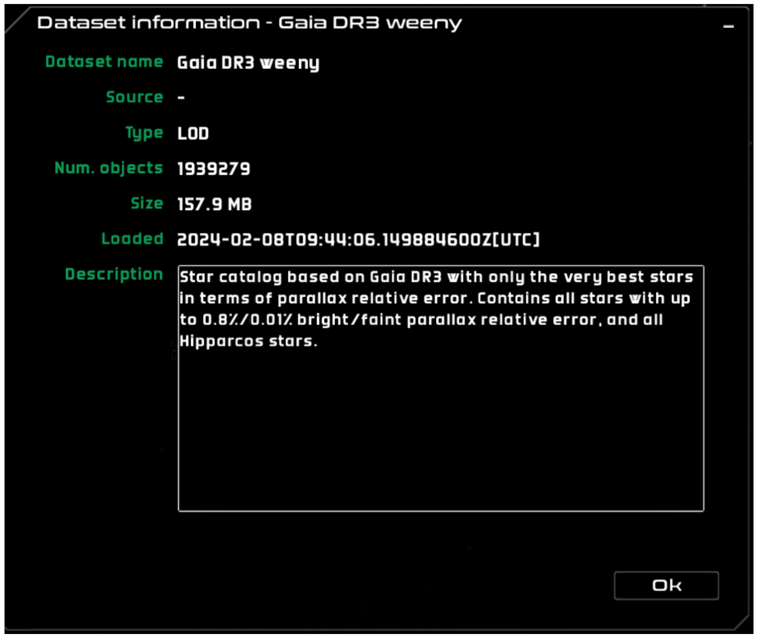



Fig. 53: The dataset information dialog

For each dataset you get:

- **Dataset name** – the name of the dataset.
- **Source** – the source. Only populated if the dataset is loaded from the UI or via SAMP.
- **Type** – the type of dataset.
- **Num. objects** – the number of objects in the dataset.
- **Size** – the size in disk.
- **Loaded** – exact time when the dataset was loaded.
- **Description** – dataset description.

## 1.4.9 Bookmarks

Gaia Sky offers a bookmarks manager to keep your favorite objects and locations organized, in the form of the **bookmarks pane**. Open the bookmarks pane by clicking on the bookmark button (to the top-left of the main window), or by pressing *b*.

Bookmarks are laid out in a folder tree. Bookmarks can either be in the root level or in any of the folders, which can also be nested.

There are two types of bookmarks:

- **Object bookmarks** – the bookmark contains an object, addressed by its name or identifier. When an object bookmark is activated, the camera is put in focus mode and the object becomes the current active focus. If the object does not exist in the current scene, nothing happens. If the object exists but is not visible, a small text appears below the bookmarks tree notifying the user.

- **Location bookmarks** – the bookmark contains a camera state (position, direction and up vectors) plus time. When the bookmark is activated, the camera is put in free mode and in the state defined by the bookmark. The time is also changed to the time defined in the bookmark.

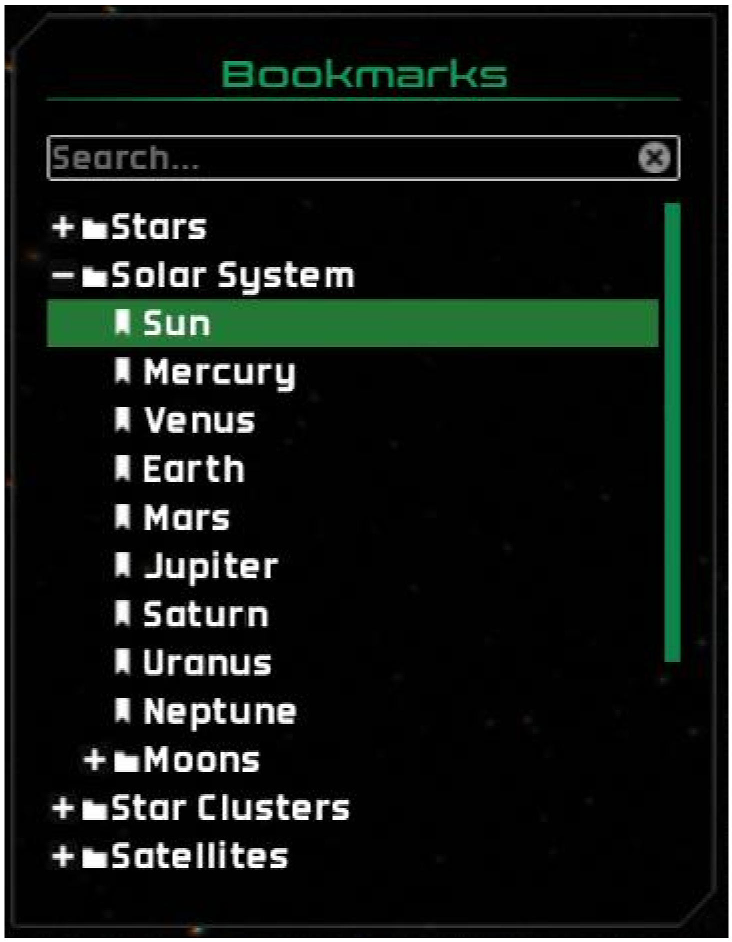



Fig. 54: The bookmarks pane in Gaia Sky.

New bookmarks are added at the end of the root level (top of the folder structure). Move bookmarks around with the context menu that pops up when right clicking on them. This context menu also provides controls to create new folders and to delete bookmarks. Bookmarks can also be deleted by clicking on the star next to the name. Once the bookmark is removed, the star's color changes to gray.

Bookmarks are saved to the file `$GS_CONFIG/bookmarks/bookmarks.txt` (see *the folders section*). The format of the file is straightforward: each non-blank and non-commented (preceded by `#`) line contains a bookmark. The form of the bookmark is `folder1/folder2/[...]/$OBJECT`, where `$OBJECT` depends on the type of bookmark.

- For **object bookmarks**, `$OBJECT` is just the name or identifier:

```
Solar System/Moons/Phobos
```

- For **location bookmarks** `$OBJECT` takes the form `{[x,y,z]|[dx,dy,dz]|[ux,uy,uz]|time_instant|name}` where:
  - `[x,y,z]` is the position in the internal reference system and internal units.

- `[dx,dy,dz]` is the camera direction vector, normalized.
- `[ux,uy,uz]` is the camera up vector, normalized.
- `time_instant` is the time, with year, month, day, hour, minute, second and millisecond, in the format 1970-01-01T00:00:00Z.
- `name` is a user-given name to identify the bookmark. Names do not need to be unique, but it is recommended.

You can edit this file directly or share it with others.

This is a valid bookmarks file, containing both object and location bookmarks:

```
# Bookmarks file for Gaia Sky, one bookmark per line, folder separator: '/', comments: '#
↪'
Stars/Sirius
Stars/Betelgeuse
Star Clusters/Pleiades
Star Clusters/Hyades
Satellites/Gaia
Solar System/Sun
Solar System/Earth
Solar System/Mercury
Solar System/Venus
Solar System/Mars
Solar System/Phobos
Solar System/Deimos
Solar System/Jupiter
Solar System/Saturn
Solar System/Uranus
Solar System/Neptune
Solar System/Moons/Moon
Solar System/Moons/Phobos
Solar System/Moons/Deimos
Solar System/Moons/Amalthea
Solar System/Moons/Io
Solar System/Moons/Europa
Solar System/Moons/Ganymede
Solar System/Moons/Callisto
Solar System/Moons/Prometheus
Solar System/Moons/Titan
Solar System/Moons/Rhea
Solar System/Moons/Dione
Solar System/Moons/Tethys
Solar System/Moons/Enceladus
Solar System/Moons/Mimas
Solar System/Moons/Janus
Eclipses/{[-1.3818553459726281232945836836106e2,-5.9917425700177573571529058068255e1,2.
↪1303961097244123780058304559795e1]|[-0.9548201218738775,0.050259057590566286,-0.
↪29290367357694286]|[0.20409057609035922,0.8273195986777884,-0.5233443592843308]|1601-
↪06-30T02:22:39Z|1601 June 30}
Eclipses/{[1.1368509657421360252389426851098e2,4.9302412843139140040634986507950e1,8.
↪0423454187100112838538598275 4e1]|[0.6572659958889423,-0.556806002482852 6,0.
↪5079059817005352]|[0.452255658439425,0.8304891688337087,0.3251961867233694]|1816-11-
↪19T09:48:15.369Z|1816 November 19}
```

(continues on next page)

(continued from previous page)

```
Eclipses/{[2.712929921336811249592957850 23e1,1.1775967148962571594413864754 48e1,-1.
↪4555234726955511211277605134986e2]|[0.4097340656192956,-0.6784083087277446,-0.
↪6098197783937811]|[-0.35381119523816085,0.497988712246363,-0.7917227296215221]|1997-03-
↪09T01:13:10.032Z|1997 March 9}
```

### Creating bookmarks

You can create **object bookmarks** by simply clicking on the star next to the object's name when in focus. Once the object is in the bookmarks, the star will brighten up with a clear white color (depending on the UI theme). Object bookmarks can also be created by right-clicking on the object and selecting *Bookmark: [object name]* in the context menu that pops up.

You can create **location bookmarks** by positioning the camera in the location, orientation and time of your desired bookmark, right clicking anywhere on the scene and selecting *Bookmark current position/time*.

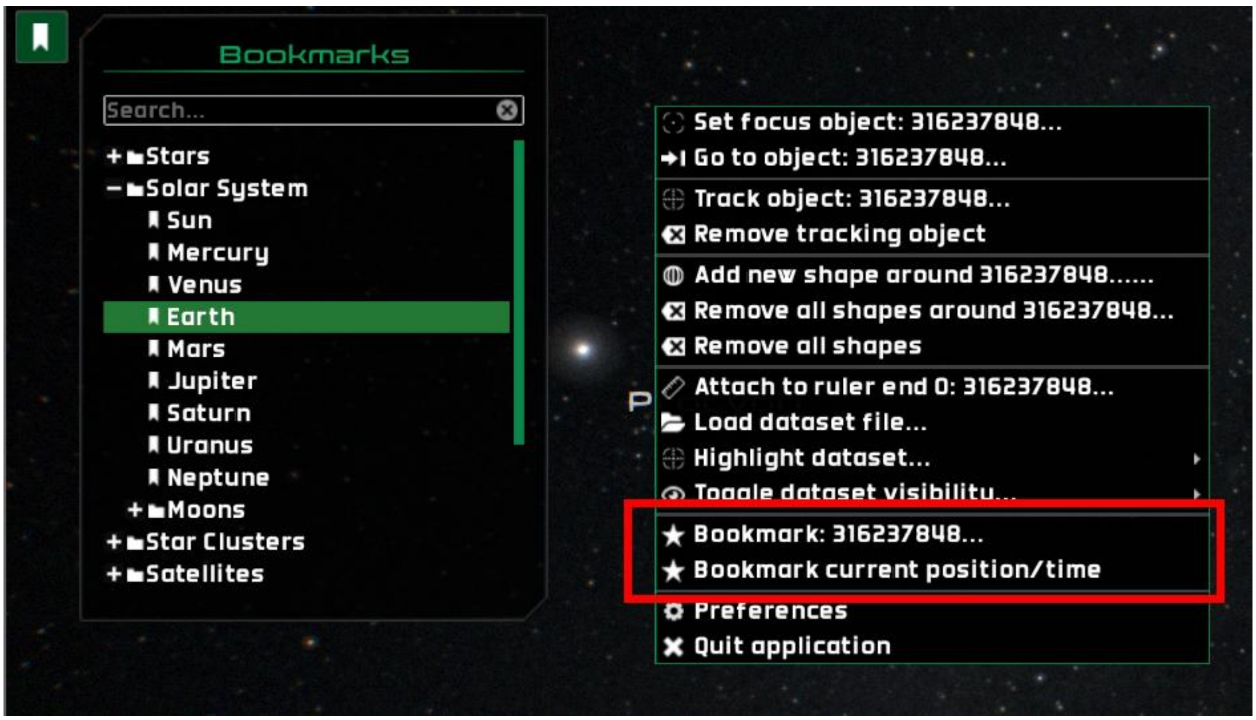



Fig. 55: The bookmarks entries in the context menu to create an object and a location bookmark.

### 1.4.10 Location log

Gaia Sky provides a small location log feature that keeps track of the visited locations and objects during a session. Currently, the location log is limited to 200 entries. Old entries are deleted as new ones come in.


Fig. 56: The location log pane keeps track of the objects you have visited.

The **location log pane** can be found anchored to the right in the main window. Expand and collapse it by clicking on the map marker button.

Every entry in the location log displays the **time since the visit** to the object (in orange, hover over it to get the absolute time), and has two actions available:

- re-visits the location with the same camera and time setup as when it was first added: this sets the camera position, direction and up vectors to match exactly the ones at the time of the visit, and set the simulation time as well
- instantly go to the object of this location

### 1.4.11 System information

Gaia Sky has a couple of built-in methods to get information on the system and graphics memory, the frame rate, the graphics device, the LOD status and much more. First, the *system information panel* offers a quick and easy way to access all sorts of system information while running Gaia Sky, in the main user interface. Second, the *debug mode* enables the logging of additional information to the *system log*, which can be helpful to analyze crashes or bugs.

**Contents**


- *System information*
  - *System information panel*

– *Debug mode*

### System information panel

Gaia Sky has a built-in debug information panel that provides a lot of good information on the current system and is hidden by default. You can bring it up with *ctrl + d*, or by ticking the *Show debug info* check box in the *System* tab of the preferences dialog.

By default, the system information panel is collapsed.


Fig. 57: Collapsed system information panel, showing the current frame rate (green) and the frame time (white). The small *[+]* icon to the bottom expands the panel.

You can expand it with the *[+]* symbol to get additional information.

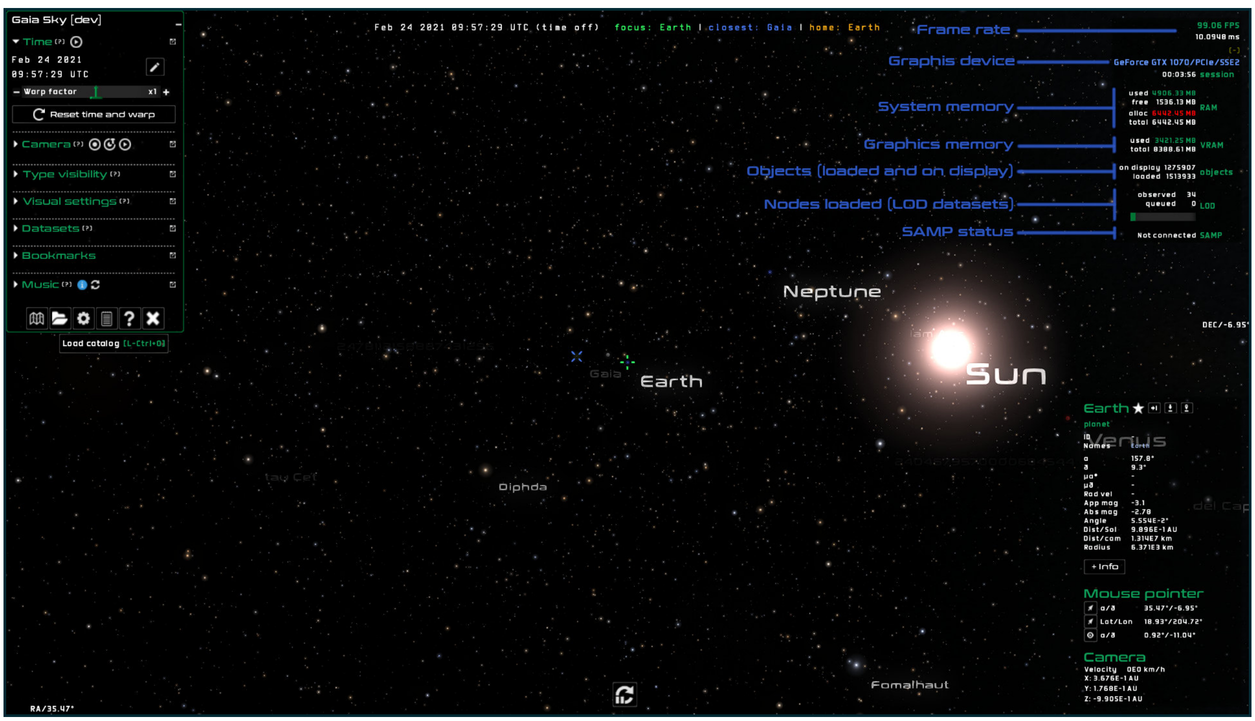



Fig. 58: Expanded system information panel, displaying the graphics device, system memory, graphics memory, loaded objects, LOD nodes and SAMP status, additionally to the frame rate and time.

As you can see, it contains information on the current graphics device, system and graphics memory, the amount of objects loaded and on display, the octree (if a LOD dataset is in use) or the SAMP status.

Additional debug information can be obtained in the system tab of the help dialog (*?* or *h*).

### Debug mode

Gaia Sky includes a mode where more information is printed in the standard output (and the log files) to help locate and identify possible problems. This is called **debug mode**.

In order to run Gaia Sky in debug mode, you need to launch it from the command line (your terminal application of choice in Linux or macOS, PowerShell or cmd in Windows) using the `-d` or `--debug` flags.

On **Linux** or **macOS**, fire up your terminal, navigate to your Gaia Sky installation directory, and run:

```
./gaiasky --debug
```

On **Windows**, open PowerShell, navigate to your Gaia Sky installation directory, and run:

```
.\gaiasky.exe --debug
```

You will be able to see the log printed out in the terminal window. You can also recover the log files if you need to. More info in the *logs section*.

## 1.4.12 Camera paths

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

**Contents**


- *Camera paths*
  - *Camera path file format*
  - *Camcorder*
  - *Recording camera paths*
    - *Frame rate*
  - *Keyframes editor*
    - *Keyframes file format*
    - *Creating and editing keyframes*
    - *Adding keyframes*
    - *Keyframes list*
    - *Playback controls*
    - *Export keyframes to camera paths*
    - *Keyframes preferences*
    - *Export keyframes with OptFlowCam*
  - *Playing camera paths*

### Camera path file format

The format of the `.gsc` files is pretty straightforward. It is a comma- or white space-separated text file (both are supported), 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. Lines starting with the character '#' are ignored (with the exception of the frame rate annotation).

The **first line** in a camera path file may contain the target frame rate. If so, it should look like this:

```
#fps 60.0
#time,pos_x,pos_y,pos_z,dir_x,dir_y,dir_z,up_x,up_y,up_z
2021-12-03T10:15:30Z,17.663479293630523,7.669932047249439,-147.39800210363168,0.8908469033686479,0.15523508799939112,-0.42695885306703063,-0.15546522466942284,0.9872364428222099,0.03456544346939234
2021-12-03T12:15:30Z,17.66336523250385,7.669924111844091,-147.39824473603434,0.8930812247239476,0.15539061859390402,-0.4222081023601746,-0.15543885339145308,0.9872410377214266,0.03455280444582746
[...]
```

In that case, when the camera file is played back, the frame rate is automatically detected and used.

As for the content, the reference system used for positions and directions is explained in the *Internal reference system* section. The units are $1 * 10^{-9} m$.

The format of each row is as follows:

- time, which may be in one of these formats:
  - `long integer`, defined as the number of milliseconds since epoch, 1970-01-01T00:00:00Z (UTC).
  - `ISO-8601 date string` that contains an instant in UTC, like '2011-12-03T10:15:30Z'.
- 3x `double-precision float` – [px, py, pz] position of the camera.
- 3x `double-precision float` – [dx, dy, dz] direction vector of the camera.
- 3x `double-precision float` – [ux, uy, yz] up vector of the camera.

Gaia Sky provides two ways to **record camera paths**: real time recording and keyframes. *Keyframes* are dealt with in the next sub-section. Here we look at the real time recording of camera paths using the integrated **camcorder**.

### Camcorder

The camcorder is located at the top of the *camera pane*. It offers buttons to do the following actions:

- / – start and stop recording a camera path. When the *REC button* is gray, the camcorder is not recording. Press it to start recording a new camera path. When the camcorder is recording, the *REC button* is red. Press it to stop the current recording. When a recording is stopped, it is automatically saved to a file in the `$GS_DATA/camera` *directory*. The file name is auto-generated.
- – open the *keyframes editor*.
- – *play a camera path file*.

### Recording camera paths

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* turns red, , which indicates the camera is being recorded.

**Hint:** You can't record camera paths while the camcorder is in *playing* mode.

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

### Frame rate

**Mind the FPS!** The camcorder stores the time, position and orientation of the camera every frame. It is important that recording and playing back are done with the same (stable) frame rate.

To set the target recording frame rate, edit the *Target FPS* field in the *camcorder settings* of the preferences window. This makes sure the camera path is recorded using the target frame rate.

*Camera path files are annotated with the target frame rate* In order to play back the camera file at the right frame rate. When the frame rate is in the camera file, the playback system automatically uses it.

### Keyframes editor

The keyframe editor 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 of the control panel (marked with a red arrow in the screenshot below). A new window will pop up from which you'll be able to create and manage the keyframes.

### Keyframes file format

Keyframes can be saved and loaded to and from `.gkf` files using the **keyframes file format**. 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*).

The keyframes file format is a text format with comma-separated values. Lines starting with '#' are ignored.. It contains a line for each keyframe. Each line has the following columns:

- `double-precision float` – keyframe duration since the last keyframe, in seconds. The duration of the first keyframe must be 0.0.
- simulation time of the keyframe, which may be in one of these formats:
  - `integer`, defined as the number of milliseconds since epoch, 1970-01-01T00:00:00Z (UTC).
  - `ISO-8601 date string` that contains an instant in UTC, like '2011-12-03T10:15:30Z'.
- 3x `double-precision float` – [px, py, pz] position of the camera in this keyframe.
- 3x `double-precision float` – [vx, vy, vz] direction of the camera in this keyframe.

- 3x `double-precision float` – [ux, uy, uz] up vector of the camera in this keyframe.
- OPTIONAL: 3x `double-precision float` – [tx, ty, tz] position of the camera target, or point of interest, for this keyframe. This vector is only present if the camera was in focus mode when the keyframe was created or modified.
- `integer` – seam mark. This is 1 if the keyframe is a seam, and 0 otherwise.

### Creating and editing keyframes

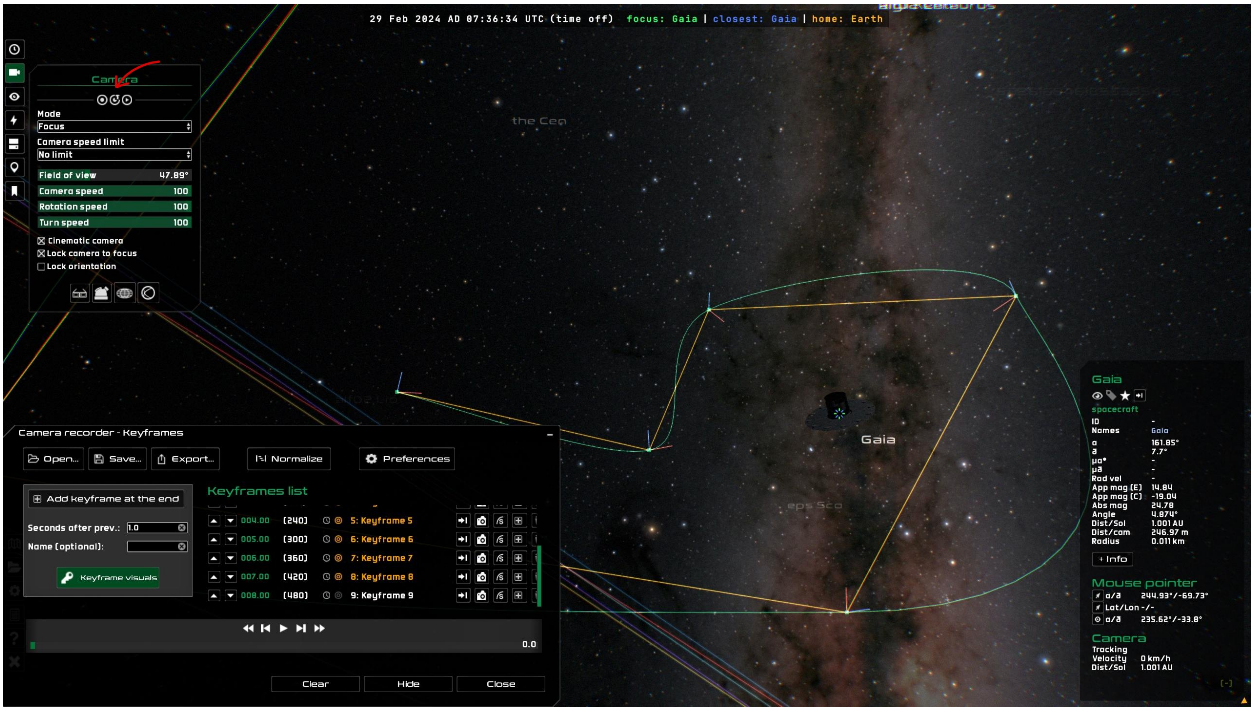



Fig. 59: Creating a keyframed camera path around Gaia. To the bottom-left, we see the keyframes editor window with some keyframes. The red arrow points to the button used to launch the keyframes editor.

A visual representation of keyframes is displayed in the 3D world (see screenshot above) as lines (splines, orientations, etc.) and points (keyframe locations). The colors are as following:

- Yellow lines – linear interpolation paths between keyframes.
- Green lines – spline paths (either B-Splines or Catmull-Rom splines) between keyframes. They represent the true camera position. If the position interpolation is set to *linear* in the keyframes settings, the green lines coincide with the yellow lines. If the position interpolation is set to *Catmull-Rom splines*, the green line should hit every keyframe position perfectly. If the position interpolation is set to *B-splines*, the green line should only hit perfectly the path start and end points.
- Red lines – for each keyframe, the red line represents the camera direction vector.
- Blue lines – for each keyframe, the blue line represents the camera up vector.
- Green points – represent keyframe locations for keyframes which are not currently selected.
- Magenta points – represent the keyframe that is currently selected and acts as a camera focus, if any.

The keyframe visual representation is only displayed when the visibility of keyframes in enabled, so make sure that the visibility of `Keyframes` is on (you can use the *Keframe visuals* button in the keyframes editor directly).

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).
- *Alt* + Right mouse – drag to rotate the keyframe orientation around the perpendicular to the up and the direction vector (not represented in the scene).

### Adding keyframes

In order to add a new keyframe, click on the *Add keyframe at the end* button. The new keyframe will be created after the current one (if any), and with the current camera state (position, orientation). If the camera is in focus mode, the keyframe name will appear in yellow, and the keyframe will have a target, or point of interest. See the *OptFlowCam export section* for more information.

The keyframe is also created with as many seconds after the previous keyframe as stated in the *Seconds after prev.* text field, and with the name given in the *Name (optional)* text field. If no name is entered, the default name of "Keyframe x" is used, where x is a monotonically increasing integer.

### Keyframes list

To the right of the keyframes editor is the keyframes list. It is a list of keyframes with their properties, along with some buttons to perform some actions. The list is sorted top-to-bottom in the same order as they are in the path. Each entry in the list has the following elements, in order, left to right:

- – move the keyframe up in the list. This moves the keyframe one position to the left in the path.
- – move the keyframe down in the list. This moves the keyframe one position to the right in the path.
- Keyframe time – the time of the keyframe relative to the first one, in seconds. The first keyframe in the list always has the time 000.00. By default, keyframes are created 1.0 seconds after the previous one. **Left click** on the green keyframe time label to **edit** it on the fly. Once the edition is done, press *Enter* to persist.
- (Frame number) – the frame number relative to the first keyframe. This is just the keyframe time times the target FPS (defined in the *keyframe preferences*).
- – hover over this icon to see the simulation time attached to this keyframe.
- – when this icon and the keyframe name are in yellow color, the keyframe has a **target** position (also referred to as 'point of interest'). A keyframe has a target only when it was created when the camera was in focus mode. In that case, the keyframe target is the position of the camera focus object at the time. Targets are used only by the *OptFlowCam export function*. Refer to that section for more information.
- Keyframe name – the name of the keyframe. By default, this will be "keyframe x", where x is a monotonically increasing integer number in the keyframes list, starting at 1. **Left click** on the keyframe name label to **edit** it on the fly. As with the target icon, the keyframe name appears in yellow if the keyframe has a target position

(point of interest). Targets are used only by the *OptFlowCam export function*. Refer to that section for more information.

- – go to the keyframe. Puts the camera in the state specified by the keyframe.
- – set keyframe to the current camera state. This allows to modify the given keyframe by setting it to the current state of the camera (including position, orientation and target, if any).
- – mark the keyframe as seam. In case the spline interpolation method is chosen, this will break the spline path.
- – add a new keyframe after this one, interpolating position, orientation and time with the next one. If the two keyframes have a target, the target position is also interpolated.
- – remove the keyframe.

### Playback controls

Below the keyframes list is a series of playback controls and a timeline slider. The slider is annotated with the current frame, and can be used to position the camera at any location in the path in real time. No need to export the camera path.

- **Beginning** – Move to the beginning of the timeline.
- **Step back** – Step one frame backward.
- **Play/pause** – Play or pause the camera path.
- **Step forward** – Step one frame forward.
- **End** – Move to the end of the timeline.

### Export keyframes to camera paths

To the top of the keyframes window there are a few buttons to load, export and save keyframes projects.

- **Open...** – load a new `.gkf` keyframes file. The button opens a file picker in the default camera directory (`$GS_DATA/camera`, see *system directories*).
- **Save...** – save the current keyframes to a keyframes file `.gkf` in `$GS_DATA/camera`. You can choose the file name, but not the directory. If another file with the same name exists, a unique file name is generated from the given one.
- **Export...** – export the current project to a camera path file `.gsc`. Optionally, the *OptFlowCam method* can be used to export. The export process uses the settings defined in the *kefyrame preferences*. You can choose the file name, but not the directory. If another file with the same name exists, a unique file name is generated from the given one.
- **Normalize** – recompute all keyframe times so that speed is constant. The total length and distance are unaltered.
- **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** – method used to interpolate between positions (orientations are always interpolated using quaternion interpolation). 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. Seam keyframes break up the path into two sections, so that two splines will be used ending and beginning at the keyframe.
  - **B-splines** – produce smoothed paths which **do not** hit every keyframe. In this mode, keyframes can be seams. Seam keyframes break up the path into two sections, so that two splines will be used ending and beginning at the keyframe.
  - **Linear interpolation** – keyframe positions are joined by straight lines. In this mode, the yellow and green lines above are the same.

### Export keyframes with OptFlowCam

Gaia Sky includes the option to use the OptFlowCam method[1] to export the keyframes to camera path files. This method works very well in smoothing paths which span over long distance ranges and extremely varying scales.

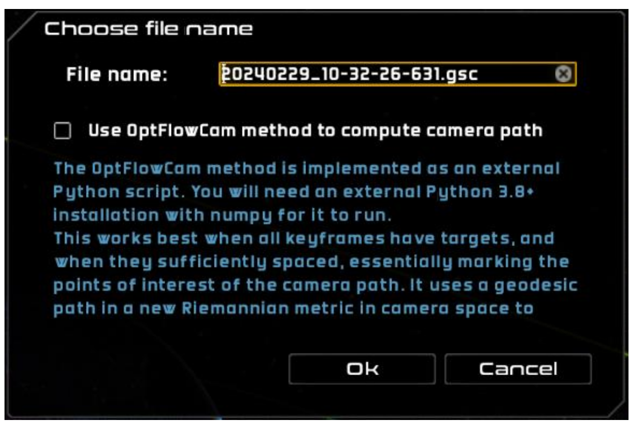



Fig. 60: The keyframes export window contains a check box to activate the OptFlowCam post-processing.

A few caveats need to be taken into account to use this functionality:

- The **OptFlowCam** processing is implemented as an external **Python script**, so a local installation of Python 3.x must be in place and accessible via the operating system. NumPy is also required. Typically, once Python is installed, you can install NumPy by running `pip3 install numpy` in a terminal. In some cases, if the Python environment is externally managed, you may need to install it via your package manager (for instance, in Arch Linux you would do `pacman -S python-numpy`). The Flatpak version of Gaia Sky already includes Python and NumPy.
  - Install Python on Windows.
  - Install Python on macOS.

[1] *Piotrowski, Motejat, Roessl, Theisel. "OptFlowCam: A 3D-Image-Flow-Based Metric in Camera Space for Camera Paths in Scenes with Extreme Scale Variations", Computer Graphics Forum, 2024 (10.1111/cgf.15056),* link.

  - Install Python on Linux: use the package manager provided by your distribution.

- The technique works best if **every keyframe has a target** or point of interest (see sections above for more information), so make sure that all your keyframes are created when the camera is in focus mode. Otherwise, a default distance-to-target of 500.000 Km is assumed.
- Note that the green line visually indicating the camera path **is not respected** in this mode. Only the keyframes themselves are guaranteed to be hit, but not so the interpolated positions between them.
- Following up on the previous point, the method is **not interactive**, and only kicks in at export time. Do not use the visual path lines for guidance, as they are not updated with this method in real time.

### Playing camera paths

In order to play a camera file, click on the *PLAY button* at the top of the *camera pane*. A file picker dialog opens, where you can select the camera path file to play. The file picker opens by default in the `$GS_DATA/camera` folder (see *folders*).

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

**Hint:** You can't play camera paths while the camcorder is in *recording* mode.

Camera paths are **recorded at a fixed frame rate**. Starting in version 3.6.1, these files are annotated with the target frame rate, so that the player automatically uses it.

In previous versions of Gaia Sky, however it was necessary to manually cap the frame rate to the target frame rate. To do so, see the *graphics configuration section*.

## 1.4.13 Settings and configuration

Gaia Sky can be configured using the on-screen UI and the preferences window. Bring up the preferences window by clicking on the preferences icon in the Controls pane or by pressing *p*.

Some features are not exposed in the preferences window or UI, so you may need to dive deep into the *configuration file* section to modify them.

**Contents**

- *Settings and configuration*
  - *Graphics settings*
    * *Resolution and mode*
    * *Visual settings*
  - *Scene settings*
  - *Interface settings*
  - *Performance*
  - *Controls*

- *Screenshots*
- *Frame output*
- *Camcorder*
- *Panorama mode*
- *Planetarium mode*
- *Data*
  - *Gaia*
- *System*


### Graphics settings

The *Graphics settings* tab in the preferences window contains most of the graphics settings in Gaia Sky.


Fig. 61: The graphics settings in Gaia Sky.

### 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.

### Visual settings

- **Graphics quality** – This setting governs the size of the textures, the complexity of the models and also the quality of some graphical effects like the star glow or the lens flare. 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. Low-fidelity Milky Way model.
- **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. Medium-fidelity Milky Way model.
- **High** – high-resolution 4K (3840x2160) textures. Graphical effects use a large number of samples. High-fidelity Milky Way model. This may be taxing even on good graphics cards.
- **Ultra** – very high resolution textures (8K, 16K, etc.). Ultra-high-fidelity Milky Way model.
- **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.
- **Point cloud style** – the point cloud rendering style. This affects the rendering of all particle datasets (Oort cloud, SDSS, etc.), stars (including Hipparcos and all Gaia-based catalogs, as well as variable stars) and asteroids.
- **Quality billboards** – in this mode, the data points are rendered as billboards (quads composed of two triangles each which always face the camera) using instancing to save VRAM. This is generally the faster option with modern GPUs. This mode produces *geometrically correct* stars and particles,

which means that they have consistent scene orientations in cubemap mode, eliminating the seams completely. Use this when using the *panorama* or *planetarium* modes.

- **Legacy (point primitives)** – This is the mode used in Gaia Sky before `3.1.7`. It uses point GL primitives (`GL_POINTS`) to render point clouds. The points are rasterized in image space, so they are not consistently projected across the whole field of view. Otherwise, this mode is fine for the regular use of Gaia Sky, and tends to perform better on very old hardware.
- **Line style** – select the line rendering back-end.
- **Quality lines** – use geometry shaders to generate polyline quad-strips, resulting in much better-looking and more consistent lines. Trajectories and orbits are also sent to the GPU once, and updated periodically. The use of geometry shaders may have a slight impact on performance with some graphics cards, but it is typically unnoticeable.
- **Legacy (line primitives)** – use the line primitives offered by the graphics driver. Since the lines are shaded by the driver implementation, they may differ depending on the graphics card. Trajectories and orbits are sent to the GPU in a buffer only once, the rest of the custom lines are computed on the CPU and sent over each frame.
- **Lens flare** – set the strength of the lens flare effect. Set to 0 to disable the lens flare. There are currently three different lens flare options, but they need to be chosen directly in the configuration file. See *this section* for more information.
- **Bloom effect** – this slider controls the amount of bloom (light bleeding from bright to dark areas) to apply to the scene. Bring it all the way down to zero to disable bloom altogether.
- **Unsharp mask factor** – this slider controls the amount of sharpening to apply to the scene with the unsharp mask effect. Increasing the unsharp mask factor makes the visuals sharper but possibly introduces aliasing and visual artifacts. Bring it all the way down to zero to disable the unsharp mask effect.
- **Chromatic aberration amount** – the amount of chromatic aberration to apply to the image. Set to 0 to disable the chromatic aberration effect.
- **Film grain** – the amount of film grain to apply to the image. Set to 0 to disable the film grain effect.
- **Fade time [ms]** – set the time it takes for objects to fade in and out when their visibility is modified, either via the "Object visibility" pane or using the individual visibility toggle. This value is in milliseconds.
- **Elevation (terrain height)** – choose the way elevation (also referred to as terrain height) is represented in Gaia Sky. This only works when the objects has a height map (texture, cubemap or SVT) attached, and also a height scale. If the object has a normal map, normals are computed from this map. Otherwise, the height texture is used to compute the normals.
- **Regular vertex displacement** – displace the object's vertices along the normal vector to represent height. Note that a `heightScale` value, indicating the extent of the displacement with an elevation multiplier of 1, is needed for this to work correctly.
- **Terrain tessellation** – use geometry subdivision by tessellation for large bodies (planets and moons). For bodies with a rough size greater than about 500 Km, tessellation subdivision is used before displacing the vertices. This may be taxing on integrated or old graphics cards. Disable if frame rate is low. Note that a `heightScale` value, indicating the extent of the displacement with an elevation multiplier of 1, is needed for this to work correctly.
- **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** – control the number of objects with self-shadows at any given 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.
- **Virtual textures** – this section contains settings related to the *sparse virtual texturing system* in Gaia Sky.
  - **Cache size** – use this slider to determine the cache size, in tiles. The size of each tile depends on the first virtual texture dataset loaded. Gaia Sky supports only multiple virtual textures in the same scene when all have the same tile size. You can adjust this slider to modify the size of the texture used as cache. The changes apply only the next time you start Gaia Sky.
- **Experimental** – this section contains experimental graphics options:
  - **Post-processing re-projection** – use a post-processing shader to re-project the final image, with a varied choice of projection algorithms:
    - **Disabled** – no re-projection.
    - **Default (simple fisheye)** – a simple fisheye projection algorithm.
    - **Accurate (no full coverage)** – a more accurate projection, but has a coverage of 180°, which is not available with the perspective camera.
    - **Stereographic (screen fit)** - stereographic projection with a screen fit.
    - **Stereographic (long edge fit)** - stereographic projection with a long axis fit.
    - **Stereographic (short edge fit)** - stereographic projection with a short axis fit.
    - **Stereographic (180 fit)** - stereographic projection with a fit to a filed of view of 180°.
    - **Lambert (screen fit)** - Lambert projection with a screen fit.
    - **Lambert (long edge fit)** - Lambert projection with a long axis fit.
    - **Lambert (short edge fit)** - Lambert projection with a short axis fit.
    - **Lambert (180 fit)** - Lambert projection with a fit to a filed of view of 180°.
    - **Orthographic (screen fit)** - orthographic projection with a screen fit.
    - **Orthographic (long edge fit)** - orthographic projection with a long axis fit.
    - **Orthographic (short edge fit)** - orthographic projection with a short axis fit.
    - **Orthographic (180 fit)** - orthographic projection with a fit to a filed of view of 180°.
  - **Dynamic resolution** – in this mode, the resolution of the back-buffer is adapted depending on the frame rate to avoid too drastic slow-downs. The dynamic resolution is adjusted according to some predefined back-buffer scale factors: 1, 0.85 and 0.75. The resolution of the back-buffer is scaled by the next value if the frame rate is below 30, and to the previous

level if it is over 60. This should provide smoother frame-rates on older hardware, and in some GPU demanding situations.

- **Back-buffer scale** – resolution scale factor to apply to the render frame buffer, effectively rendering the scene at a lower or higher resolution in the background, trading off performance and visual fidelity. This setting is disabled when dynamic resolution is enabled.
  - Set the back-buffer scale to **less than one** to render the image with a lower resolution, increasing performance and lowering visual fidelity, and upscale it to the window size.
  - Set the back-buffer scale to a value **greater than one** to render the image with a resolution higher than that of the current window, decreasing performance and increasing visual fidelity, and downscale it to window size.
- **Index of refraction** – set the index of refraction of the sphere in *orthosphere view mode*. The orthosphere is filled up with a material with the given refraction index, with light rays bending and scattering according to their angles of incidence.
- **Screen space reflections** – activate SSR (screen space reflections). In this method, a post-process step traces the reflections for each reflective surface in the image. This has an impact on performance but produces nice-looking reflections on metallic surfaces. If this is off, it falls back to cubemap reflections with a default sky box of the milky way. The default location of the sky box is `$GS_DATA/tex/skybox/gaiasky`.

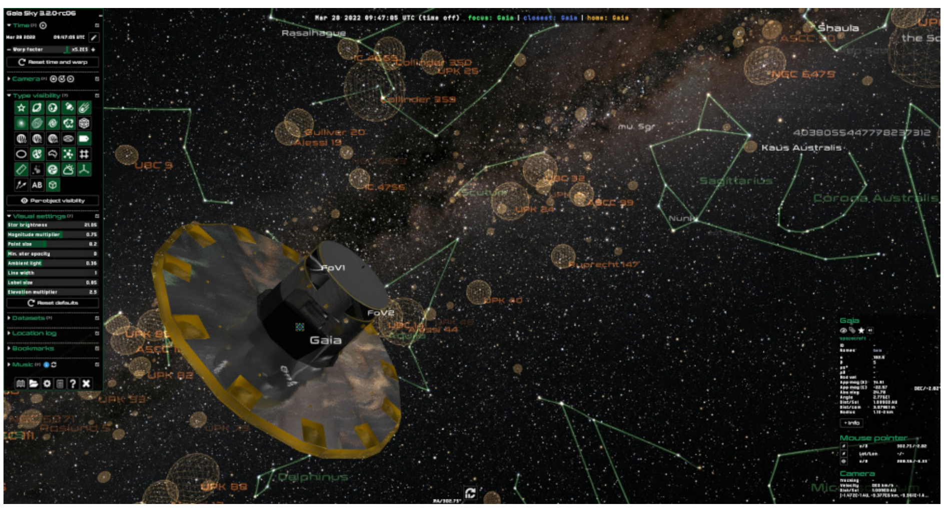



Fig. 62: A rendering of Gaia with SSR activated.

- **Motion blur** – choose the amount of *camera* motion blur to apply to the scene. Set to 0 to disable motion blur. Gaia Sky implements what is know as camera motion blur, where the scene is blurred only depending on the camera motion. Object motion blur is not implemented at the moment.

### Scene settings

The *Scene settings* tab in the preferences window contains settings concerned with the scene configuration as a whole and its objects.

- **Recursive grid** – configure the recursive grid object.
  - **Origin** – choose the origin of the recursive grid, either the reference system origin or the focus object.
  - **Origin projection lines** – if the origin is set to the reference system origin, this check box controls whether projection lines on the fundamental plane and to the object are drawn.
- **Eclipses** – enable and configure the real-time eclipse representation. For more information, visit the *eclipse representation section*.
  - **Enable eclipse representation** – enable or disable the real time in-scene eclipse representation.
  - **Enable outlines for umbra and penumbra** – enable or disable outlines for the umbra regions (red) and the penumbra region (yellow) during eclipses.
- **Stars** – configure aspects tied to stars.
  - **Star glow over objects** – enable the post-processing effect to render the star light effect that spills over occluding objects.
  - **Render star spheres** – enable the rendering of stars as spheres.

### Interface settings

The *Interface settings* tab in the preferences window contains some configuration options related to the user-facing interface, like the language, scale factor, object cross-hairs and pointer guides.

- **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.
  - `bright-green`, a bright theme with greenish tones.
  - `night-red`, a red theme for low-light environments.
- **UI scale factor** – scale the user interface up or down. This slider applies a fractional scaling factor to all user interface elements (not only the fonts!). The scaling is takes effect on the fly when you click on the *Save preferences* button. You can also apply the scaling immediately, without closing the preferences dialog, by clicking on the *Apply* button next to the slider.
- **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*.
- **Preferred distance units** – choose between *parsecs* and *light years* to use as default top units. These apply to the focus info pane (bottom-right), as well as in the projection lines of the recursive grid.
- **Display mode change information pop-up** – enable or disable the appearance of the information pop-up dialog when one of the special modes (panorama, planetarium, stereoscopic, game) is activated.

- **Crosshair** – adjust the visibility of the different crosshairs and markers.
  - **Focus marker** – mark the location of the current focus object.
  - **Closest object marker** – mark the location of the closest object to the camera.
  - **Home object marker** – mark the location of the home object, defined in the configuration file.
- **Pointer guides** – vertical and horizontal guides spanning the full window marking the position of the pointer.
  - **Display pointer guides** – enable or disable the pointer guides.
  - **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.
  - **Color** – choose the color of the pointer guides.
  - **Width** – choose the width of the pointer guide lines.

### Performance

The *Performance settings* tab in the preferences window contains a few settings that impact the performance of the application.

- **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** – adjust the solid angle threshold for when octree nodes become visible. See the *draw distance section* for more information.

More detailed info on performance can be found in the *performance section*.

### Controls

The *Controls* tab in the preferences window contains information about the keyboard, mouse and gamepad controls, and some tools to edit the gamepad mappings.

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

The *Screenshots* tab in the preferences window contains settings on the *screenshots susbsystem*.

**Hint:** Take screenshots any time 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 output resolution of the screenshots. Note that advanced mode requires the scene to be re-rendered at the target resolution, so it is slower.

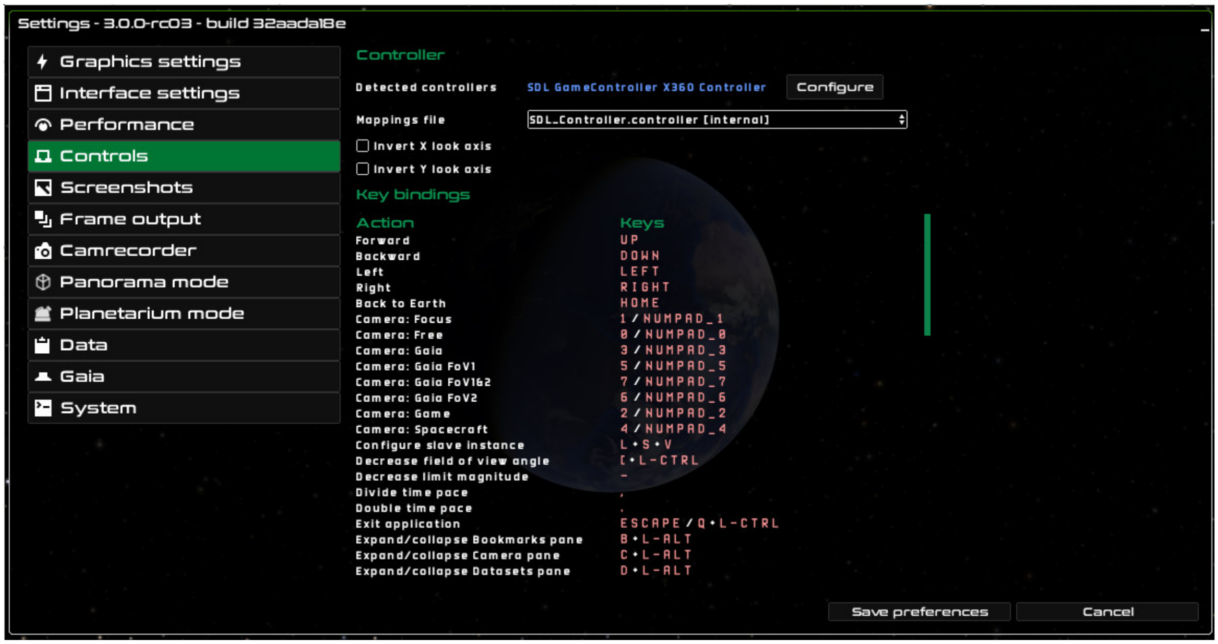



Fig. 63: The controls settings in Gaia Sky.

You can also select the output format (either `JPG` or `PNG`) and the quality (in case of `JPG` format) by using the *Image format* select box and the *Quality* slider.

These are the controls in this tab:

- **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.
- **Image format** – choose the save format, either `JPG` or `PNG`.
- **Quality** – when `JPG` is selected as an image format, use this slider to control its quality setting.

## Frame output

The *Frame output* tab in the preferences window contains settings related to the *frame output system*.

This feature enables the exporting and saving of every frame as a `JPG` or `PNG` image directly to disk. This is useful to produce videos. In the frame output tab you can select the frame save location, the image prefix name, the target frame rate, the mode and the output image resolution (in case of *Advanced* mode). You can also select the output format (either `JPG` or `PNG`) and the quality (in case of `JPG` format) by using the *Image format* select box and the *Quality* slider. Finally, there is a button to reset the integer sequence number.

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

When Gaia Sky is in frame output mode, it does not run in real time but it adjusts the internal clock to meet the configured target FPS (frames per second, or frame rate). Take this frame rate into account when you later use your favourite video encoder (ffmpeg) to convert the frame images into a video.

Here is a list of the available controls:

- **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 integer frame sequence number of the current session to 0. After clicking this, the frame sequence will start over from 0, overwriting any previously existing frames with the same name! This control is useful if you need to re-capture frames.

### Camcorder

The *Camcorder* tab in the preferences window contains settings related to the *camera path recording system*.

The following settings are available:

- **Target FPS** – 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** – enable **automatic frame recording** during playback. This will automatically activate the frame output system (see *Frame output*) during a camera file playback.
- **Keyframe preferences** – bring up a new dialog to adjust some preferences of the camera keyframe system. See *this section* for more information.

### Panorama mode

The *Panorama mode* tab in the preferences window contains settings related to the *panorama mode*.

The following settings are available:

- **Cubemap side resolution** – define the **cube map side resolution** for the 360 mode. With this mode a cube map will be rendered (six individual scenes in directions $+X$, $-X$, $+Y$, $-Y$, $+Z$, $-Z$) 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

The *Planetarium mode* tab in the preferences window contains settings related to the *planetarium mode*.

The following settings are available in the planetarium mode section:

- **Aperture angle [°]** – adjust the aperture angle to suit your dome setup. Can be as high as 360 degrees.
- **Focus angle from zenith [°]** – the angle from the zenith to put the focus of the view.
- **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.

Gaia Sky also supports the spherical mirror projection by defining a warp mesh file:

- **Select warp mesh file** – select a warp mesh file, which contains the distortion data to compensate for the non-planar nature of the projection surface. More information in the *spherical mirror projection section*.

### Data

The *Data* tab in the preferences window contains settings related to the data used in Gaia Sky.

From this tab, you can bring up the *dataset manager window* to download new datasets, and enable and disable the ones that you have available locally. To bring it up, click on the *Dataset manager* button.

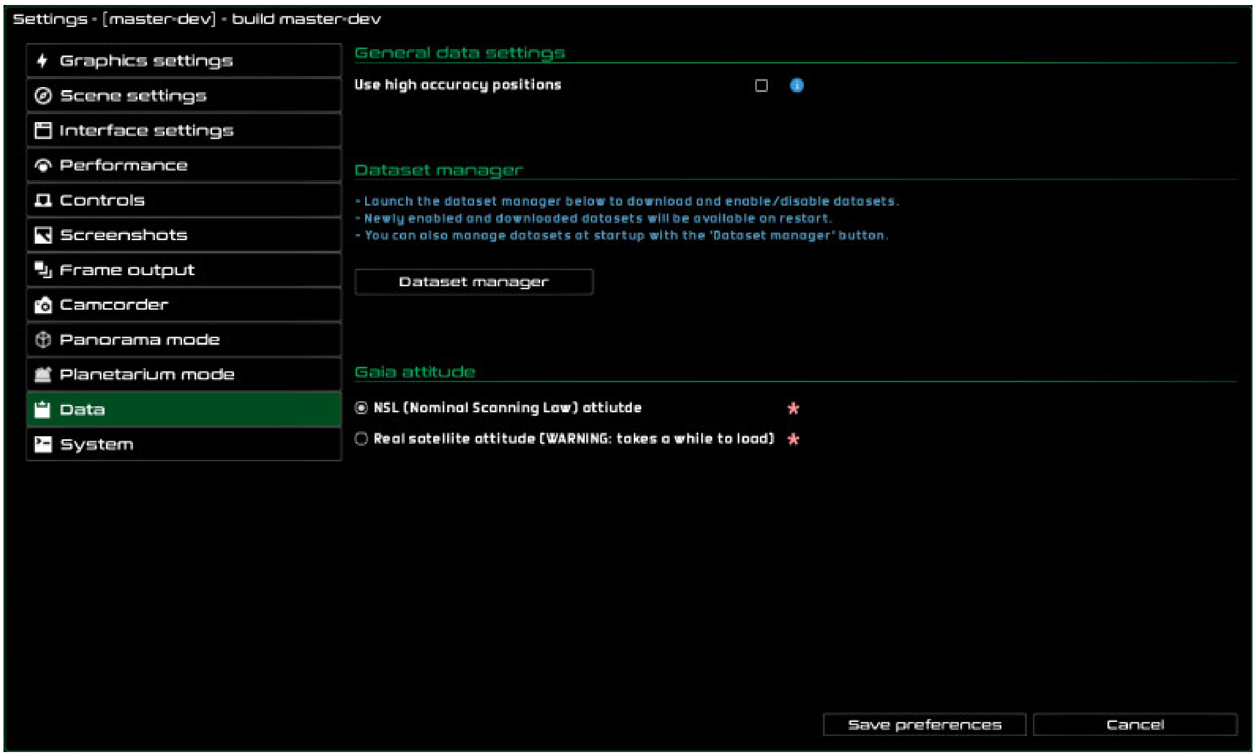



Fig. 64: The data settings in Gaia Sky.

There is a setting available in the data tab:

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

### Gaia

- **Gaia attitude** – you have two options here:
  - `Real satellite attitude` – takes a while to load but it uses the correct phase angles and parameters. In this case, the *planned* attitude of Gaia is used. This may still diverge from the actual attitude of the satellite.
  - `NSL` – analytical implementation of the nominal attitude of the satellite. It behaves the same as the real thing, but the direction to which the satellite is pointing is off.

### System

The *System* tab contains preferences that affect the whole system, or items that do not fit anywhere else.

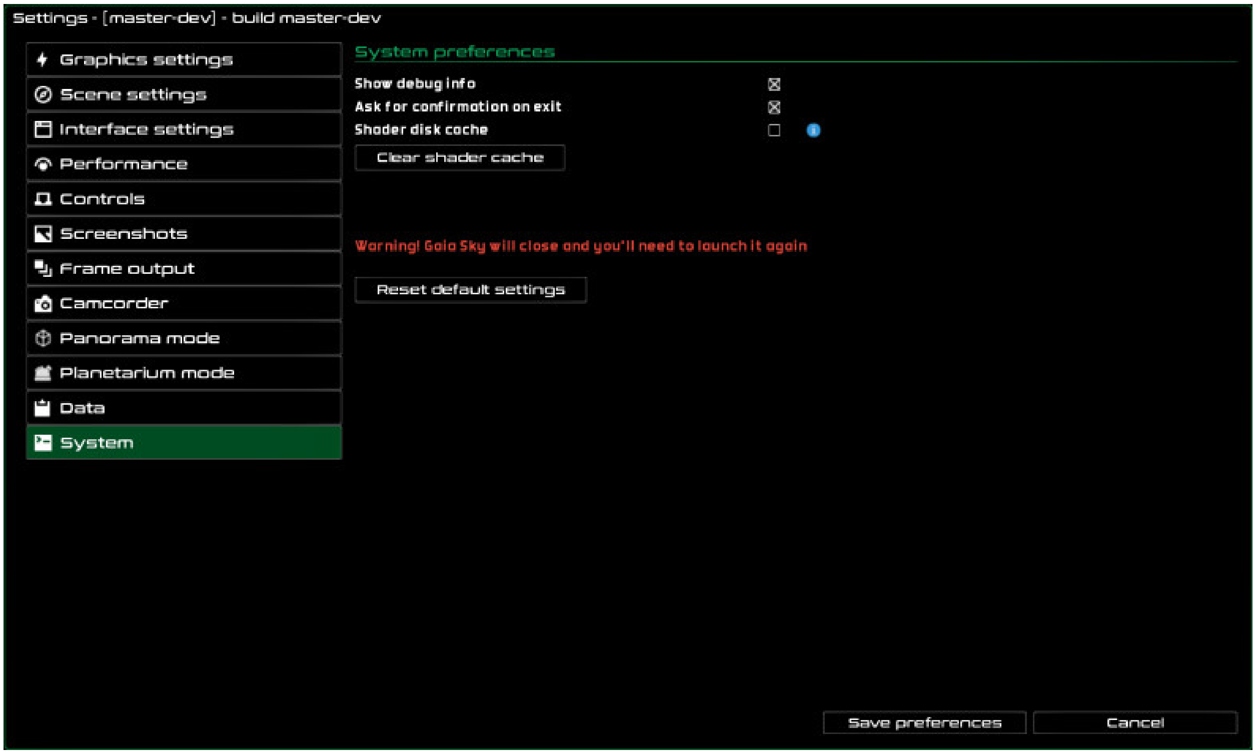



Fig. 65: The system settings in Gaia Sky.

- **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.
- **Ask for confirmation on exit** – whether to ask for confirmation when trying to close Gaia Sky or not.
- **Shader disk cache** – Gaia Sky implements an application-level shader disk cache that caches the binary, compiled shaders to disk to avoid re-compilation and save time. Most graphics drivers already implement this cache at driver level, so this setting is off by default. If you notice that the shader compilation stage at startup is very slow, you can try enabling this.
- **Clear shader cache** – use this button to completely clear the shader cache of Gaia Sky. This will remove all cached binary shaders from the disk. The shaders will be re-cached in the next start up (only if the 'Shader disk cache' checkbox is checked).
- **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.

## 1.4.14 Scripting

Gaia Sky exposes an API that can be accessed via Python scripts and via an *HTTP server*. In this section we focus on the Python method. The API calls can be called from Python programs (scripts), that must be run with the system Python interpreter. They connect to a gateway service offered by a running instance of Gaia Sky.

**Contents**


- *Scripting*
  - *Quick start*
    - *Requirements*
    - *Running a test script*
  - *The Gaia Sky API*
    - *Using the API remotely*
    - *API documentation*
  - *Writing scripts for Gaia Sky*
    - *Backing up and restoring settings*
    - *Logging to Gaia Sky and Python*
    - *Method and attribute access*
    - *Strict parameter types*
    - *Loading datasets from scripts*
    - *Camera transitions*
    - *Synchronizing with the main loop*
    - *Camera and scene runnables*
    - *Overriding object coordinates provider*
    - *More examples*
  - *Running and debugging scripts*


### Quick start

If you just need some examples to get started, look up the test and showcase scripts in scripts folder of the project.

### Requirements

In order to connect to the gateway server, you need a Python `3.5+` interpreter and the Py4J package. You can install it with `pip` as a user package like this:

```
$  pip install --user py4j
```

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 directory from which you run the script does not matter. If all goes well Gaia Sky should be showing a nice tour of the asteroids in the DR2 catalog.

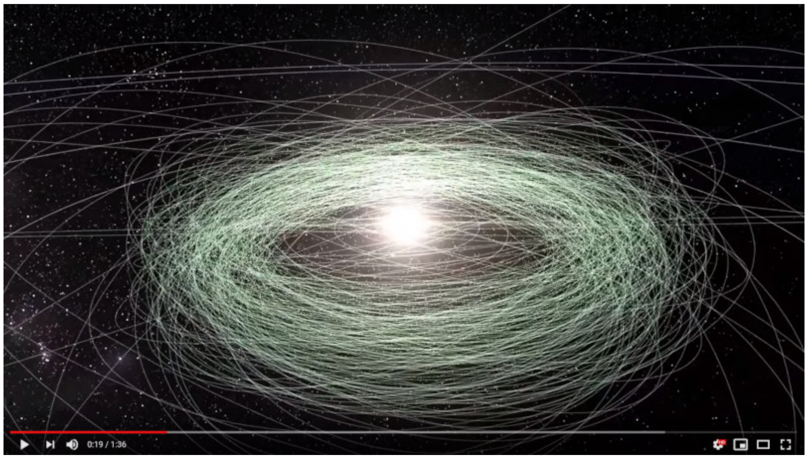


Fig. 66: This script should produce results similar to this video

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 Gaia Sky API

The Gaia Sky **API** (here) contains many more calls to interact with the platform in real time from Python scripts or a REST HTTP server. The API includes calls to:

- Add and remove messages and images to the interface,
- start and stop time, and change the time warp,
- add scene elements like shapes, lines, etc.,
- load full datasets in VOTable, CSV, FITS, or the internal JSON format,
- manage datasets (highlight, change settings, etc.),
- manipulate the camera position, orientation and mode,
- move the camera by simulating mouse actions (rotate around, forward, etc.),
- activate special modes like planetarium or panorama,
- create smooth camera transitions in position and orientation,
- change the various settings and preferences,
- back-up and restore the full configuration state,
- take screenshots, use the frame output mode.

### Using the API remotely

Gaia Sky provides a REST server that enables the remote execution of API calls over HTTP. This is described in *the REST server section*.

### 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.

- Latest API version
- Older API versions (javadoc).

### 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

# User code goes here
[...]

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 the object `gs`.

```
# Disable input
gs.disableInput()
# Stop camera
gs.cameraStop()

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

Find lots of example scripts here.

### Backing up and restoring settings

Typically, scripts modify various program settings when they run (camera speed, star brightness, field of view, etc.). In order to leave Gaia Sky in the state it was before, scripts have the option to back up and restore the entire settings state of Gaia Sky. To do that, the API includes a few calls to push and pull settings states from an internal LIFO stack:

- *backupSettings()* — push the current settings state to the settings stack.
- *restoreSettings()* — restore the top-most settings state from the settings stack so that they become immediately effective. This call re-initializes the user interface of Gaia Sky, so be aware that the UI will be reset.
- *clearSettingsStack()* — clears the settings stack. Calling *restoreSettings()* after this will have no effect.

These calls can be used at the start and end of scripts to back up and restore the user settings, so that everything is left unchanged after a script execution.

```
from py4j.clientserver import ClientServer, JavaParameters

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

# 1. Back up settings before anything
gs.backupSettings()

# 2. Script does things and modifies the settings
[...]

# 3. Restore the settings backed up at point 1.
gs.restoreSettings()

gateway.shutdown()
```

### 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:

```
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:

```
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 attrbiutes. In case you get objects from Gaia Sky, you can't directly call public attributes, but need to access them via public methods:

```
# 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:

```
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")
```

(continues on next page)

(continued from previous page)

```
# Shutdown
gateway.shutdown()
```

Find an example of how to load a star catalog from a script here. This one showcases how to load a dataset with generic particles (only positions).

Additionally, you can also load JSON data files and dataset descriptors made for Gaia Sky (see the *JSON dataset format* section).

### Camera transitions

When writing scripts it is important to be able to transition the camera from one state to another. The camera state is composed by the camera position and the orientation (direction and up vectors). In the API, we include a family of calls named *cameraTransition(...)* (see here), which produce a transition from the current camera state, to the given camera position and orientation, in a given number of seconds (optionally different for position and orientation).

Additionally, two more calls are available to create transitions only in position and only in orientation:

- cameraTransition(pos, dir, up, units, durationPos, smoothTypePos, smoothFactorPos, durationOri, smoothTypeOri, smoothFactorOri, sync).
- cameraPositionTransition(pos, units, duration, smoothType, smoothFactor, sync).
- cameraOrientationTransition(dir, up, duration, smoothType, smoothFactor, sync).

For the rest of this subsection, we refer to the base `cameraTransition(...)` method.

Typically, when the transition must traverse large dynamic ranges of distances, it is necessary to smooth the transitions by starting slow and finishing slow, or starting fast and finishing fast. To that effect, we have included a sub-family of calls which include a smoothing type and factor for position and orientation. The transition duration is also separated by position and orientation.

- API call: cameraTransition(pos, units, dir, up, posDuration, posSmoothType, posSmoothFac, oriDuration, oriSmoothType, oriSmoothFac, sync).

**Duration** — `posDuration` and `oriDuration` are in seconds, and specify the duration for the interpolation in position and orientation, respectively. They may be different, but the call will not return until the longest of the two has finished (if sync is `true`).

**Smoothing type** — `posSmoothType` and `orientationSmoothType` determine the smoothing type. There are two types: **logistic sigmoid** and **logit** (additionally, **none** skips the smoothing). The logistic sigmoid type starts and ends slow, while the logit type starts and ends fast.

Check out this Graphtoy simulation. In it, $f_2$ is the logistic sigmoid (yellow), and $f_3$ is the logit type (green).

The full transition path is mapped to x:[0,1], and we use y:[0,1] given by the smoothing function to generate the sampling.

**Smoothing factor** — `posSmoothFactor` and `orientationSmoothFactor` determine the smoothing factor. In **logistic sigmoid** the factor must be in [12, Inf]. In **logit**, the factor is in [0.09, 0.01]. You can use the Graphtoy utility above to see the effect of the different factors. You can see and modify the expressions in the text fields to the top-right.

You can always find the **target camera state** values by putting the camera in the end position and orientation in Gaia Sky, and running the `get-cam-pos.py` script (under `assets/scripts/tests/`):

```
$  python get-cam-pos.py
```

(continues on next page)

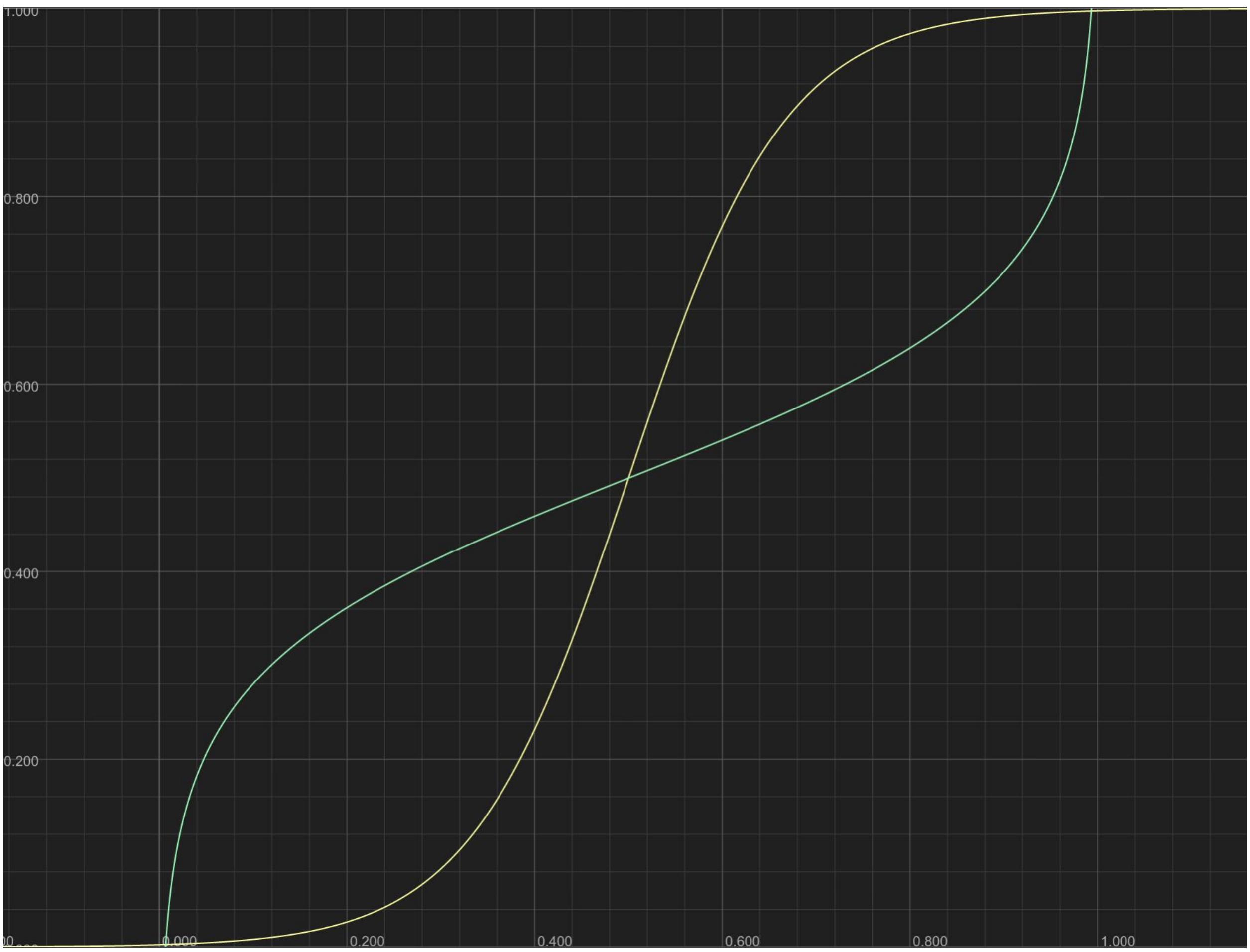



Fig. 67: Smoothing types logistic sigmoid (in yellow) and logit (in green).

(continued from previous page)

```
Camera position:
 -  Internal:     [-5593.0417731364, 13008.1430225486, 1542.9688571213]
 -  Km:           [-5593041773.1363697052, 13008143022.5486202240, 1542968857.1212708950]
 -  AU:           [-37.3871749360, 86.9540651588, 10.3141097317]
 -  Light years:  [-0.0005911850, 0.0013749619, 0.0001630919]
 -  Parsecs:      [-0.0001812581, 0.0004215652, 0.0000500042]

Camera orientation:
 -  Direction:     [0.3965101844, -0.9104556836, -0.1176865406]
 -  Up:     [0.7060462888, 0.2205005155, 0.6729622283]
```

Then, you can create a transition from the current camera state, to the target camera state. Here we have used internal units (first data line in the above snippet).

```
gs.cameraTransition([-5593.0417731364, 13008.1430225486, 1542.9688571213],  # Position
                "internal",                                                 # Units
                [0.3965101844, -0.9104556836, -0.1176865406],               # Direction
                [0.7060462888, 0.2205005155, 0.6729622283],                 # Up
                10.0,                                                       # Transition␣
↪duration in position [s]
                "logisticsigmoid",                                          # Smoothing␣
↪type in position
                60.0,                                                       # Smoothing␣
↪factor in position
                7.0,                                                        # Transition␣
↪duration in orientation [s]
                "logisticsigmoid",                                          # Smoothing␣
↪type in orientation
                12.0,                                                       # Smoothing␣
↪factor in orientation
                True                                                        # Sync
                )
```

### 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 fo 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`:

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

class FrameCounterRunnable(object):
    def __init__(self):
```

(continues on next page)

(continued from previous page)

```
        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 synch with the main thread.

### Camera and scene runnables

The Gaia Sky main loop updates first the camera position and orientation, and then updates the objects in the scene. In order to maintain sufficient precision, the scene is floated at the position of the camera, meaning that the camera is always effectively at the origin of coordinates, and the scene objects are moved around. This means that the effective position of every objects in the scene at every frame depends on the position of the camera.

So far, we have seen the `parkRunnable()` method, which parks a runnable that runs only after the camera-scene update cycle. However, sometimes we need to modify the positions of objects in the scene with respect to other objects. If we use the current method, we will always be using the position in the last frame. However, we need to use the position in the current frame, and we can do so by introducing two new park methods:

- `parkCameraRunnable()` — parks a runnable that runs after the camera has updated, but **before** the scene has done so. Use this to fetch the **predicted** position of an object to have the position in the current frame. (see `fetchPredictedPosition()`).
- `parkSceneRunnable()` — parks a runnable that runs after the camera and scene have updated. This is exactly the same as the `parkRunnable()` we already know.

An example of this can be found here. It needs a couple of JSON data files (also in the repository).

### Overriding object coordinates provider

The positions of most objects in Gaia Sky are computed internally using coordinate providers. It is possible to override the coordinate providers of objects and implement your own in Python. This way of setting the position of an object is the best way to ensure internal consistency and overall system stability. When the coordinates provider is overriden, the user code runs naturally during the scene graph update stage.

To implement a coordinates provider and set it to an object, you first need to create a class that implements `IPythonCoordinatesProvider`, and submit it to Gaia Sky via the API call `setObjectCoordinatesProvider(name, provider)`. The provider class needs to have a `getEquatorialCartesianCoordinates(self, julianDate, outVector)` method, which you need to implement. In it, you need to compute the coordinates of your object for the given Julian date (double-precision floating point number), in the *internal reference system and units*, and put the result in `outVector`, using the method `outVector.set(x, y, z)`.

Here is an example:

```
from py4j.clientserver import ClientServer, JavaParameters, PythonParameters
from py4j.java_collections import ListConverter
import os

# This is the coordinates provider class.
# It implements the method getEquatorialCartesianCoordinates().
class MyCoordinatesProvider(object):

    def __init__(self, gateway):
        self.gateway = gateway
        self.gs = gateway.entry_point
        self.converter = ListConverter()
        self.km_to_u = self.gs.kilometresToInternalUnits(1.0)
        self.pc_to_u = self.gs.parsecsToInternalUnits(1.0)

    def getEquatorialCartesianCoordinates(self, julianDate, outVector):
        # Here we need internal coordinates.
        x_km = 150000000 * self.km_to_u
        z_km = 200000000 * self.km_to_u
        v = [x_km, (julianDate - 2460048.0) * 100.0, z_km]

        # We need to set the result in the out vector.
        outVector.set(v[0], v[1], v[2])
        return outVector

    def toString():
        return "my-coordinates-provider"

    class Java:
        implements = ["gaiasky.util.coord.IPythonCoordinatesProvider"]


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

# Load test star system.
```

(continues on next page)

(continued from previous page)

```
gs.loadDataset("Test star system", os.path.abspath("./particles-body-coordinates.json"))

# Set coordinates provider.
provider = MyCoordinatesProvider(gateway)
gs.setObjectCoordinatesProvider("Test Coord Star", provider)

gs.startSimulationTime()
gs.setCameraFocus("Test Coord Star")

print("Coordinates provider set.")
input("Press a key to finish...")

gs.stopSimulationTime()
# Clean up before shutting down, otherwise Gaia Sky will crash
# due to the closed connection.
gs.removeObjectCoordinatesProvider("Test Coord Star")
gs.removeDataset("Coordinates test system")

gs.sleep(2.0)

gateway.shutdown()
```

You can find this script, along with the necessary JSON data file, here.

### 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
```

### 1.4.15 Frames and screenshots

Gaia Sky includes some utilities to save frames and screenshots to disk.

#### Frame ouptut

**Hint:** Enable and disable the frame ouptut system with *F6*.

Gaia Sky has an in-built method to save every frame to an image file. The purpose of this is to produce high quality videos from the still frames. Of course, you can also produce videos by capturing the window with OBS or any other screen recorder.

To configure the frame output system (mode, image format, quality, etc.), check out the *frame output configuration* section.

Once enabled with *F6*, Gaia Sky starts saving every frame to an image file in the `$GS_DATA/frames` (see *folders*) directory. The system saves every frame until *F6* is hit again.

#### Frame ouptut modes

There are two frame output modes:

- **Simple mode** – save the current screen buffer directly to a file. This means that everything that's on the Gaia Sky window will be in the saved image, including the user interface elements.
- **Advanced mode** - render the current scene to an off-screen buffer with an arbitrary resolution. The resolution can be configured in the preferences window, *Frame output* tab. The advanced mode does **NOT** render user interface elements or any additional objects that are not part of the scene.

#### 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 the format defined in the *screenshot settings*.

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

#### Screenshot modes

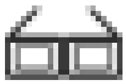
The same two modes available to the frame output system are also available to screenshots.

- **Simple mode** – save the current screen buffer directly to a file. This means that everything that's on the Gaia Sky window will be in the saved image, including the user interface elements.
- **Advanced mode** - render the current scene to an off-screen buffer with an arbitrary resolution. The resolution can be configured in the preferences window, *Screenshots* tab. The advanced mode does **NOT** render user interface elements or any additional objects that are not part of the scene.

### 1.4.16 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

---

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.

Gaia Sky also includes proper support for **VR** headsets through OpenXR. Check out the *VR section* for more info.

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.

**Stereoscopic profiles**

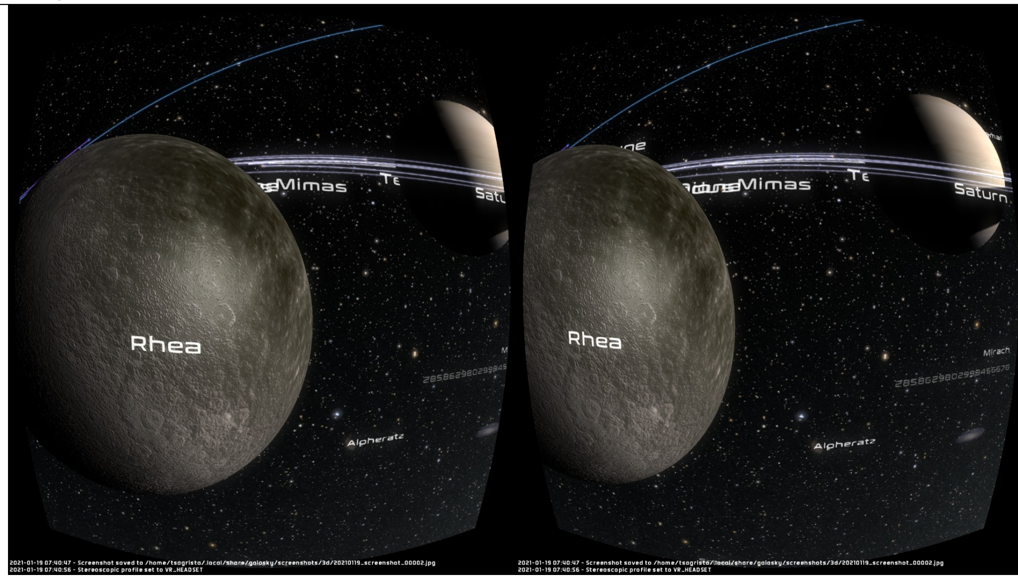
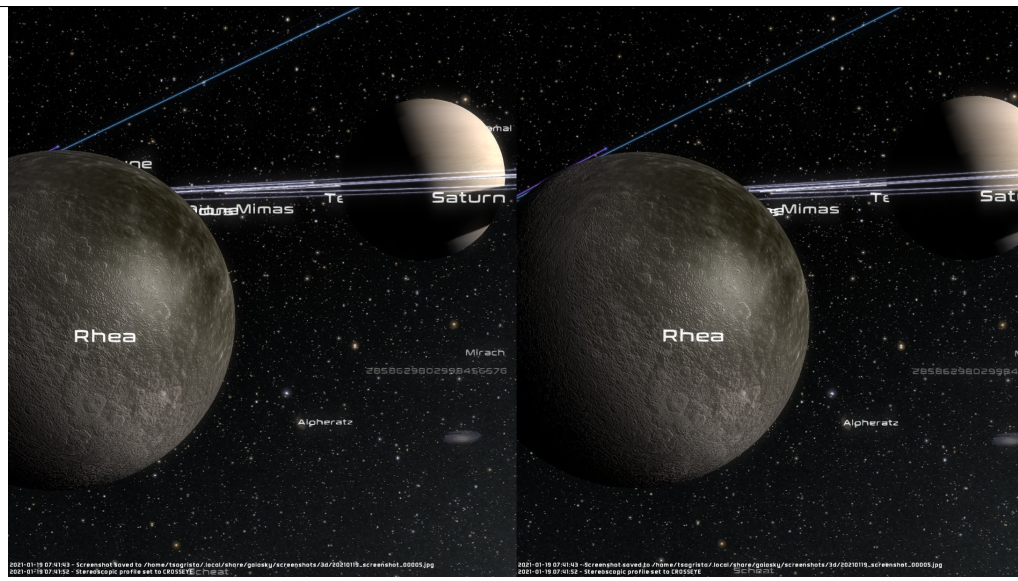
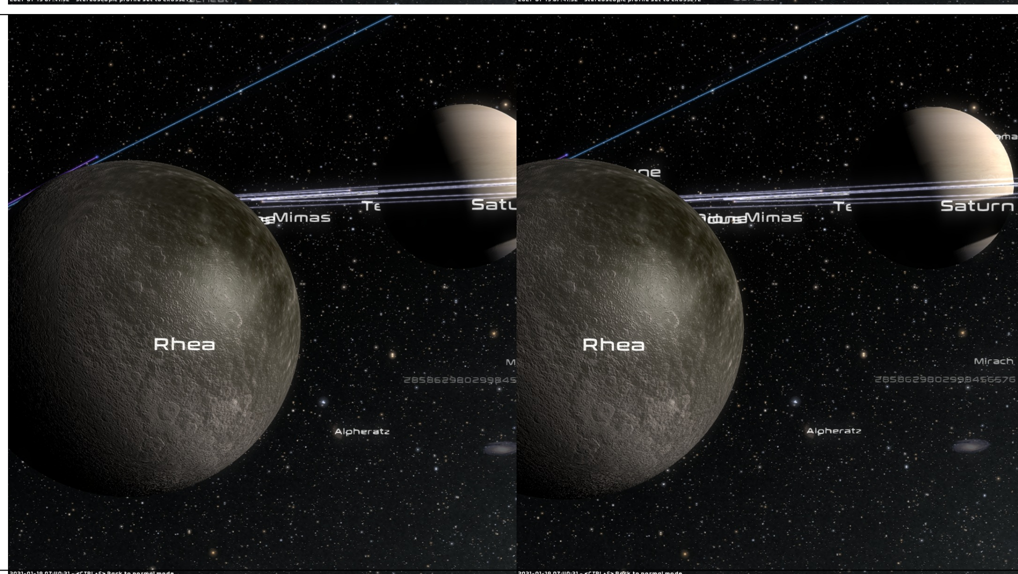
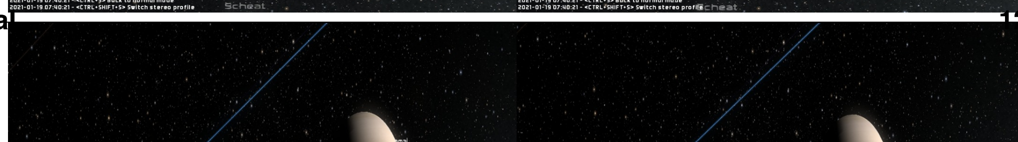
In order to manage all these parameters, we have created 6 stereoscopic profiles which can be selected by the user and are described below.

- `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 strecthed 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

---

| Profile | Image |
| --- | --- |
| **VR headset** |  |
| **Crosseye** |  |
| **Parallel view** |  |
| |  |

### 1.4.17 Planetarium mode

Gaia Sky supports different planetarium modes, depending on the projector setup.

- Single projector:
  - Azimuthal equidistant (fisheye, dome master) projection.
  - Spherical mirror projection.
- Multi-projector:
  - Use the MPCDI standard and connect various Gaia Sky instances.

**Contents**


- *Planetarium mode*
  - *Single-projector setup*
    - *Spherical mirror projection*
      - *File format*
  - *Multi-projector setup*
    - *MPCDI*
    - *Gaia Sky configuration file*
  - *Re-projection shaders*


#### Single-projector setup

Gaia Sky can output a true **azimuthal equidistant (dome master)** and **spherical mirror projected** stream suitable for single-projector dome or mirror 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*.

**Hint:** Please use **'Triangles'** as the *point cloud style* to avoid the presence of seams. Using the legacy `GL_POINTS` mode will result in visible seams.

**Hint:** To activate the planetarium mode, click on the icon in the camera section of the control panel. Exit by clicking again. You can also use the shortcut *ctrl* + *p*. Switch the projection with *ctrl* + *shift* + *p*. If it does not work, remove the *$GS_CONFIG/mappings/keyboard.mappings* file.

**Hint:** *F7* – Save the faces of the current cubemap to image files in the screenshots directory.

Just like the *panorama mode*, this planetarium mode runs by rendering the scene into a cube map (using separate renders for all directions $+X$, $-X$, $+Y$, $-Y$, $+Z$, $-Z$) and renders it using an **azimuthal equidistant (also known as dome master)** projection or a **spherical mirror** projection.

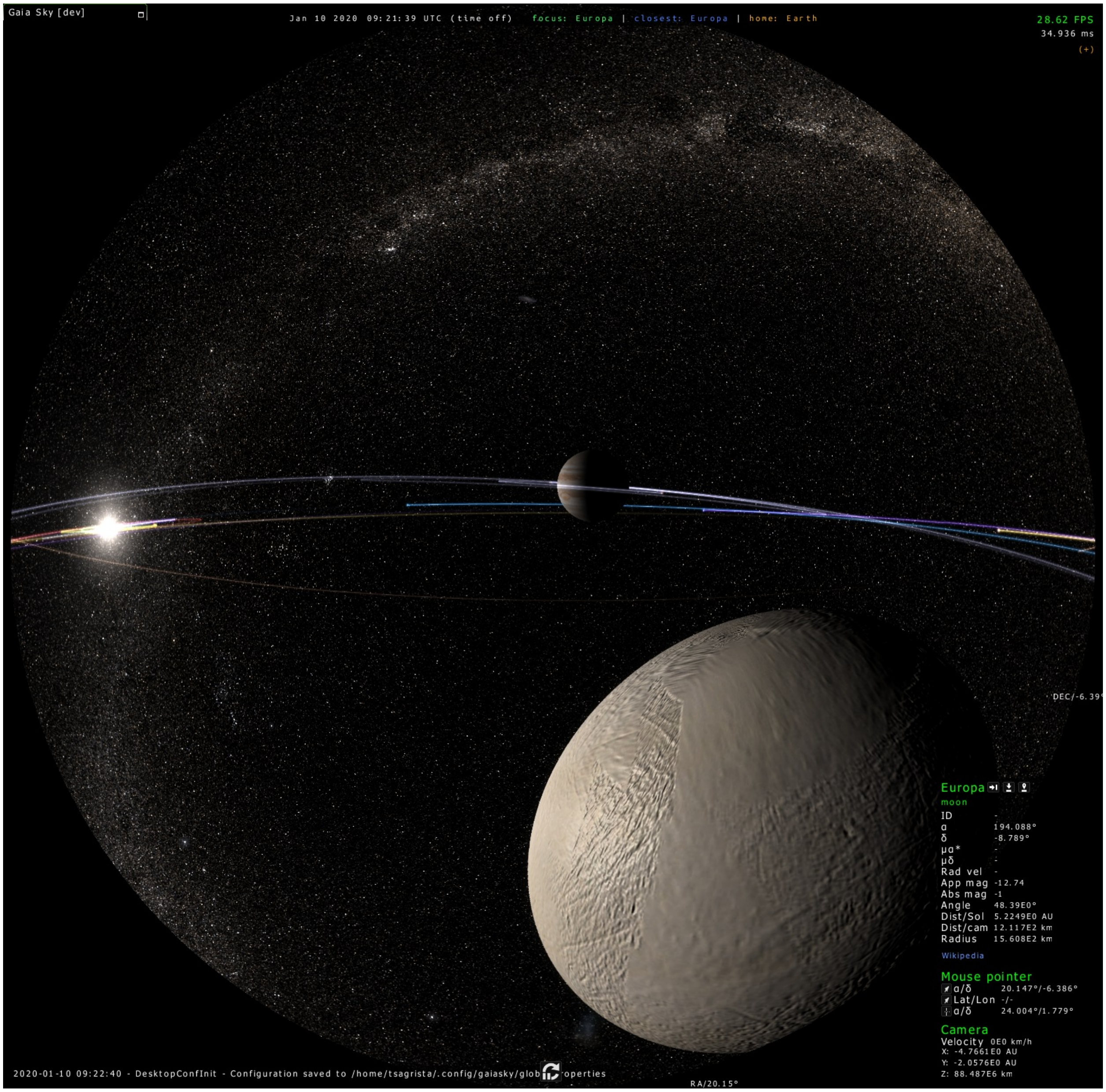



Fig. 68: Planetarium mode with the azimuthal equidistant (fisheye, dome master) projection.

Here are the planetarium mode settings. They can be modified in the preferences window, planetarium tab.

- **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 $50°$ downwards. This setting is not adjustable as of now.

### Spherical mirror projection

Gaia Sky supports the spherical mirror projection, where the image is projected using a regular projector and a spherical mirror. For that, the user needs a warp file which defines the surface deformation. You can find some common warp files in our data repository. The spherical mirror file format is described in this post by its creator, Paul Bourke. We reproduce it below, in the *spherical mirror format subsection*.

- spherical_mirror.mp4 – video of the spherical mirror projection, open in new tab!

In planetarium mode, you can switch the projection mode with *ctrl* + *shift* + *p*.

In the preferences window, *Planetarium mode* tab there is one extra setting that applies only to the spherical mirror projection:

- **Warp mesh file** – select the warp file you want to use. This only applies when the spherical mirror projection is being used.

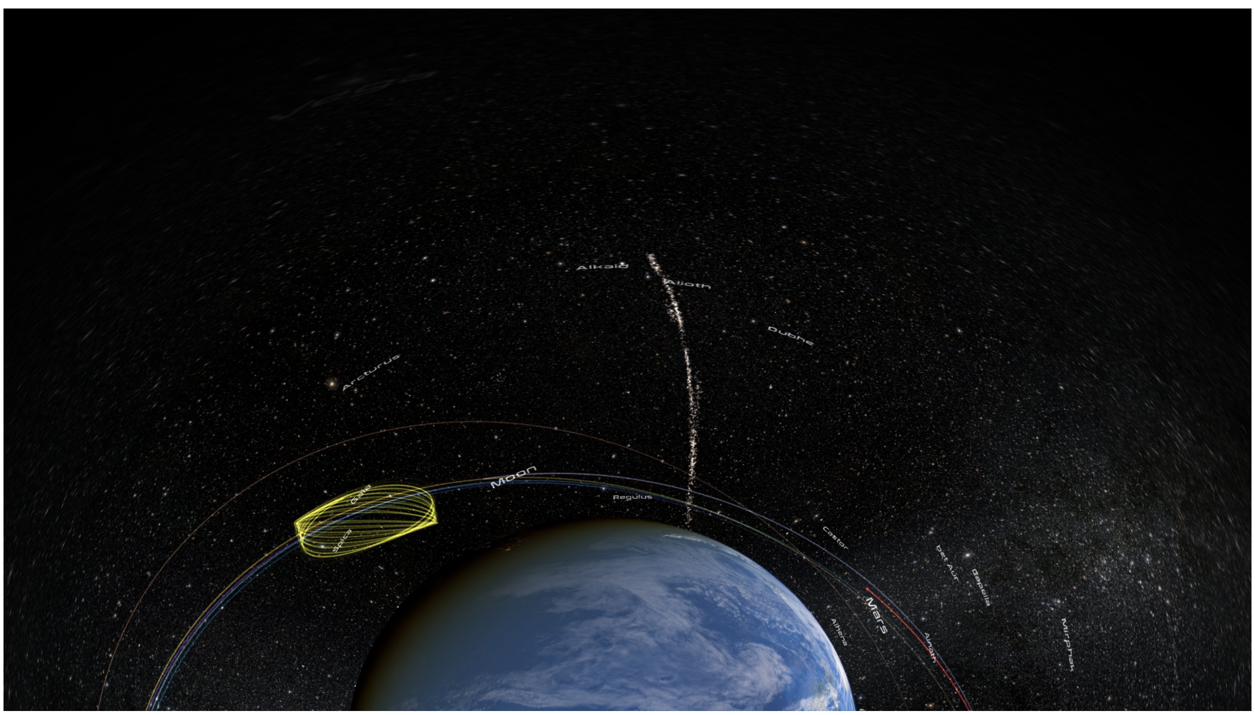



Fig. 69: Planetarium mode with the spherical mirror projection.

### File format

This content is from Paul Bourke's website describing the spherical mirror file format.

- First line contains the mesh type, currently rectangular (2) and polar (1) are supported, see figure 3. The only significant difference between these two is the mesh continuity that occurs for the polar mesh across the 0 and 360 degree boundary.
- Second line contains two integers indicating the mesh dimensions, nx and ny.
- The subsequent lines define the nodes, there should be nx times ny lines. These lines contain 5 values defined as follows.
  - Position x and y of the node in normalised coordinates. The mesh need not exactly match the projected image, in figure 2 it actually extends off the projected region while in figure 4 it matches the 4:3 aspect exactly. In the later case the horizontal range (x) will be +- the aspect ratio and the vertical range (y) will be +- 1 (ie: OpenGL style normalised coordinates).
  - Texture coordinate u and v, these should each range from 0 to 1, they refer to the original input image. Values outside the 0 to 1 range indicate that the node is not to be used, this usually means the mesh cells sharing that node are not used but sometimes it is appropriate to triangulate the mesh for such cells.
  - A multiplicative intensity value applied to each r,g,b colour value. The can be used for simple edge blending and to compensate for brightness variation due to different light path lengths from projector to projection surface. This intensity correction should range from 0 to 1, negative values indicate that the node should not be drawn. So 0 indicates none of the corresponding colour, 1 indicates fully saturated. Nodes with intensities outside this range should not be used. Note that per colour, gamma corrected edge blending requires three separate intensity scale factors, one for each r,g,b. While this is a simple extension to the format it is not included here and left to the reader to implement if required.

### 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
- The blend mask – an 8-bit RGB or RGBA PNG file with the blending mask

The master-slave connection happens via the *REST API server* in Gaia Sky.

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 `config.slave[n].yaml`, 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]
```

#### Re-projection shaders

The planetarium mode can be simulated in a geometrically incorrect manner by using the post-processing re-projection shaders. These work on the final image after the perspective projection, and re-project it using different algorithms. Please, refer to the *re-projection section* for more details.

For some shaders, you may want to use a greater field of view angle than the maximum. You can do so by directly editing the *configuration file*. For example, we can set the field of view to 160° like so:

```
scene:
  camera:
    fov: 160.0
```

Finally, since the re-projection shaders stretch the image, it may be desirable to use a larger resolution for the back buffer. This operation is experimental and not recommended, but it works. Refer to the *back-buffer scale section* for more information.

### 1.4.18 Panorama mode

Gaia Sky includes a panorama mode where the scene is rendered in all directions ($+X$, $-X$, $+Y$, $-Y$, $+Z$, $-Z$) to a cube map.

**Contents**


- *Panorama mode*
  - *Configuration*
  - *Creating panorama images*
    - *Injecting panorama metadata to 360 images*
  - *Creating spherical (360) videos*


**Hint:** To activate the panorama mode, click on the icon in the camera section of the control panel. Exit by clicking again. You can also use *ctrl* + *k*.

This cube map is then projected onto a flat image. The projections available are:

- Equirectangular/spherical.
- Cylindrical.
- Hammer.
- Orthographic – Renders each hemisphere side-by-side.

**Hint:** *ctrl* + *shift* + *k* – Cycle between the different projections.

The final image can be used to create 360 panorama videos with head tracking (see here).

**Hint:** *F7* – Save the faces of the current cubemap to image files in the screenshots directory.

### Configuration

Please, see the *panorama mode configuration* section.

**Hint:** Please use **'Triangles'** as the *point cloud style* to avoid the presence of seams. Using the legacy `GL_POINTS` mode will result in visible seams.

### 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

The program ExifTool can be used to inject the 360 metadata into the images. For example, with a panorama 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
```

### 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/
```

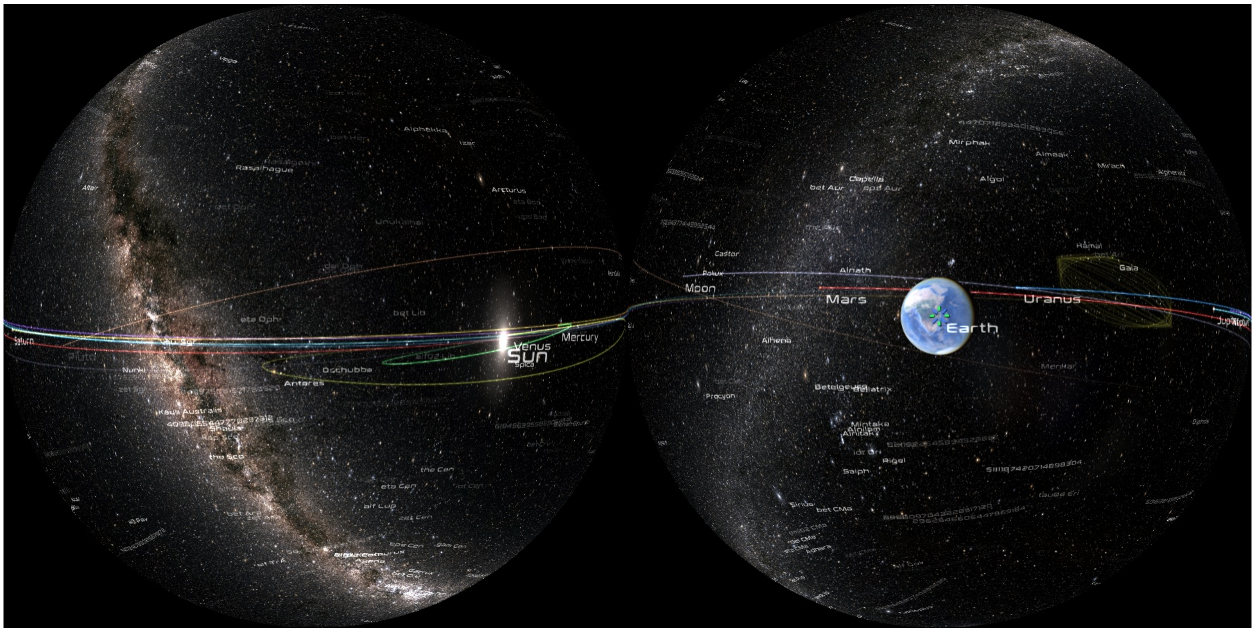



Fig. 70: Panorama image captured with Gaia Sky, using the orthographic projection.

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.

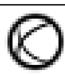
```
$ python spatialmedia -i <inupt_file> <output_file>
```

You are done, your video can now be viewed using any 360 video player.

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

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

### 1.4.19 Orthosphere view mode

**Hint:** Use the button  in the camera pane, or *ctrl* + *j* to enter and exit the orthosphere view mode.

The orthosphere view mode blends the two hemispheres of the orthographic projection in *panorama mode* on top of each other to simulate a full celestial sphere. There are two profiles:

- **Orthosphere** – the base mode, in which both hemispheres are blended on top of each other. Optionally, you can fill up the sphere with a material with a certain *refraction index*. To do so, open the preferences dialog, go to the Graphics configuration tab and scroll down to the experimental section. You will find a "Refraction index" slider to modify it.
- **Orthosphere cross-eye** – a stereoscopic (3D) cross-eye mode which lays the images for each eye side-by-side.

**Hint:** *ctrl* + *shift* + *j* – Cycle between the different profiles.

**Hint:** *F7* – Save the faces of the current cubemap to image files in the screenshots directory.

### 1.4.20 Eclipse representation

Gaia Sky can represent eclipse events between bodies. You can enable and configure eclipses in the preferences window (see the *scene settings page*).

By default, we provide a series of bookmarks to Solar eclipse events (see the *bookmarks documentation*) in the 'Eclipses' folder in the bookmarks pane. Try clicking on any of those, and you should be immediately transported to the time and place of an eclipse.

If highlighting is enabled the penumbra region is highlighted with a yellow line, and the umbra region is highlighted with a red line.

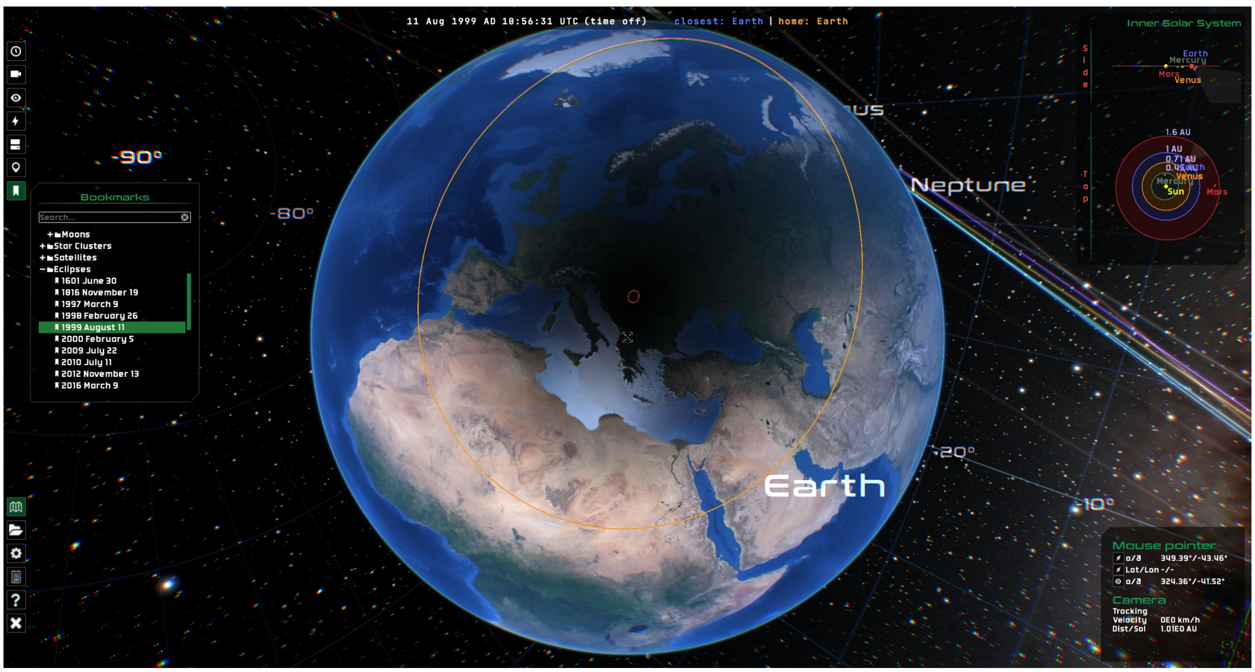



Fig. 71: The Eclipse of August 11, 1999 in Gaia Sky.

### 1.4.21 Bounding shapes

You can add shapes around any focus-able object. To do so, right click on the object you want to add the shape around and a context menu like the following pops up:

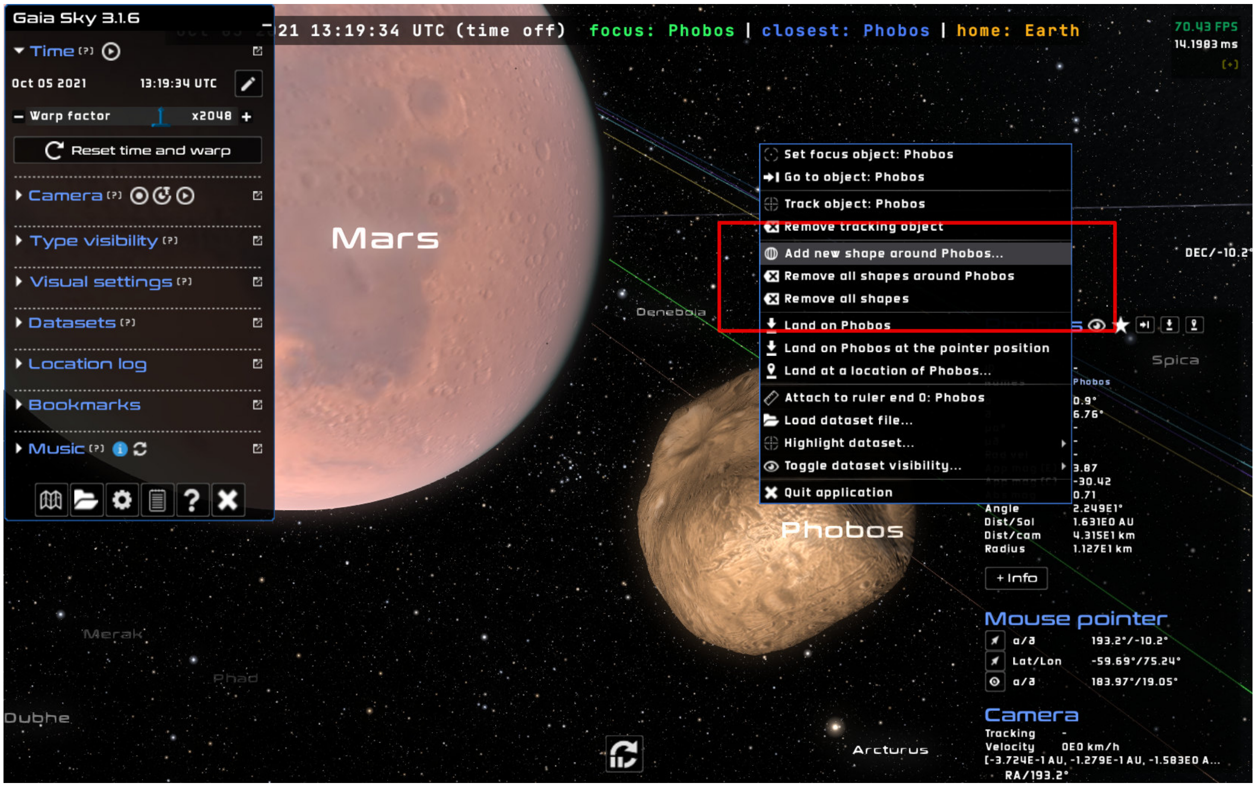



Fig. 72: Add bounding shape, context menu

### Adding bounding shapes

If you select "Add shape around 'object'. . . ", the following dialog shows up:

When adding a shape around an object there are a few properties that we can choose:

- **Object name** – The name of the object. This will show up as the object label if 'Show name label' is checked.
- **Object size** – The size of the object, together with the units of the size.
- **Show name label** – Whether to show the label for the bounding shape or not.
- **Track object position** – When checked, the bounding shape will follow the object around if/when it moves. Otherwise, the shape will stay at the original position.
- **Shape type** – The shape type. Possible shapes are sphere, icosphere, octahedron sphere, cone, cylinder and ring.
- **Color** – The color of the shape.
- **Primitive type* – The primitive to use for rendering. If **LINES**, the shape is shown as a wireframe. If **TRIANGLES**, the shape is rendered as a solid object.
- **Orientation** – The orientation of the shape. Can be one of:
- **Camera** – Use the current camera direction and up vectors to configure the orientation matrix of the shape.
- **Equatorial system** – Use the equatorial system.
- **Ecliptic system** – Use the ecliptic system.
- **Galactic system** – Use the galactic system.

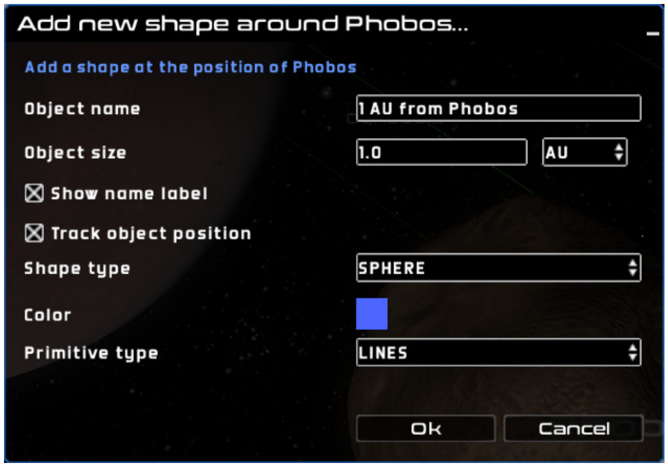



Fig. 73: Add bounding shape dialog

#### Removing shape objects

You can remove shape object using the context menu. You can either only remove the shape objects linked to a particular object with 'Remove all shapes around Object', or remove all the shapes with 'Remove all shapes'.

### 1.4.22 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`.

### 1.4.23 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 **primary-replica** scenario, where one (and only one) instance acts as a primary and one or more instances act as replicas, getting their internal states updated over a network. The user interacts with the primary instance and all replicas are updated accordingly.

**Contents**


- *Connecting Gaia Sky instances*
  - *Configuration*

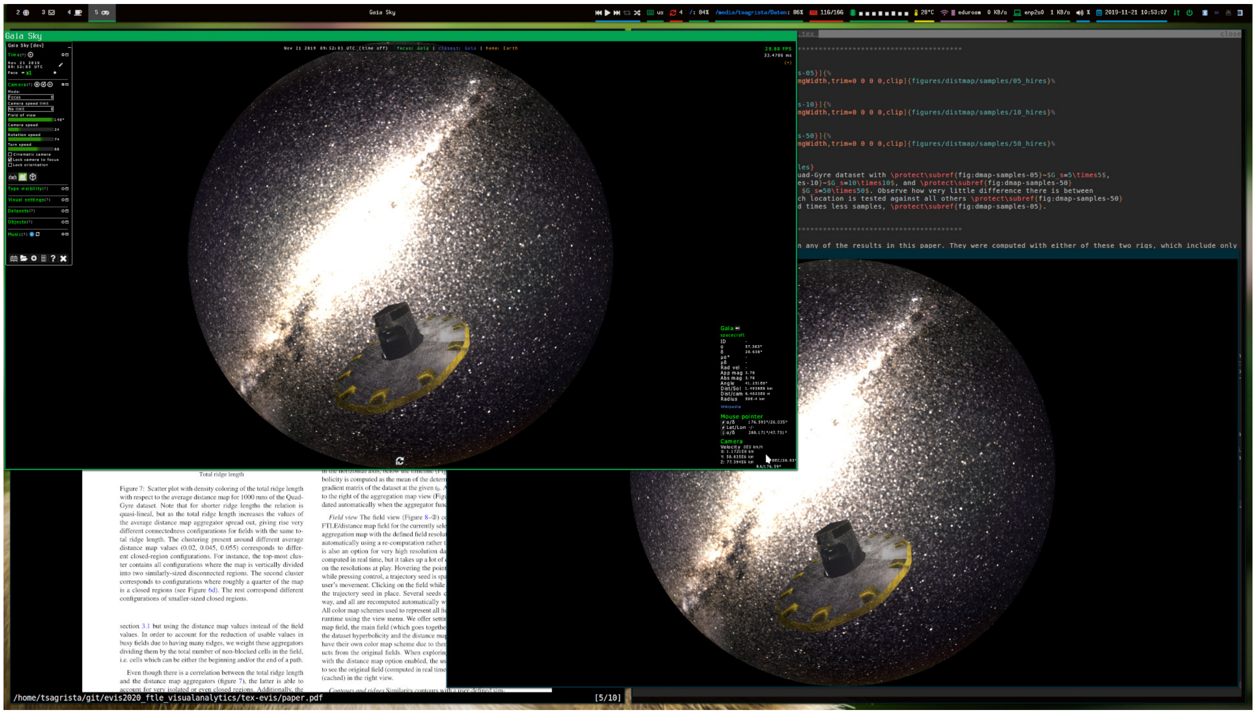



Fig. 74: External view using planetarium mode


- *Configuration: replica instance(s)*
- *Configuration: primary instance*
- *Caveats*


**Note:** In this section we use the words 'primary' and 'master' interchangeably to refer to the main Gaia Sky instance that controls the rest. We also use the words 'slave' and 'replica' to describe the instances that are controlled by the primary.

This section describes only how to configure the primary and the replica 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 replica instance.

The various instances are connected using the *REST API server* feature of Gaia Sky.

**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 (`config.yaml`). You can run Gaia Sky with a specific configuration file by using the `-p` or `--properties` command line flags:

```
$ gaiasky -p ~/.config/gaiasky/config.primary.yaml
```

The next sections explain how to configure the primary and the replica instances.

### Configuration: replica instance(s)

You can have as many replica instances as you want, but here we'll explain the process of setting up two replicas.

1. Copy the current `config.yaml` file in your config folder (see *folders*) into `config.replica0.yaml`. The name is irrelevant, but choose something meaningful. Repeat with `config.replica1.yaml`.
2. Set the property `program::net::slave::active: true` in each file and make sure that `program::net::master::active` is set to `false`.
3. Set the desired port to listen to in `program::net::restPort`. For example, to use the port 13900 just set the property `program::net::restPort: 13900`. Use a different port for each replica (i.e. replica 0 listens to 13900, slave 1 listens to 13901, etc.). For example, to set up a replica in port 13900, make sure that the following lines are in its configuration file:

```
program:
    net:
        restPort: 13900
        master:
            active: false
        slave:
            active: true
```

**The replica instances should be launched before the primary**. Launch the replica(s) with:

```
$ # Launch replica 0
$ gaiasky -p /path/to/config.replica0.yaml
$ # Launch replica 1
$ gaiasky -p /path/to/config.replica1.yaml
```

Once the replica(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 primary instance is starting the scripting server. The replicas are automatically forbidden to do so!

#### Configuration: primary instance

Copy the current `config.yaml` file into `config.primary.yaml` and edit the following lines.

1. Set the property `progra::net::master::active: true` and make sure that `program::net::slave::active` is set to `false`.
2. Add the locations of all desired replicas under the settings `program::net::master::slaves: [URL1, URL2, ...]`.

For example, in order to connect the primary with two replicas, both running locally (`localhost`) on ports 13900 and 13901, add the following to the `config.primary.yaml` file:

```yaml
program:
    net:
        restPort: 13900
        master:
            active: true
            slaves: [http://localhost:13900/api/,http://localhost:13901/api/]
        slave:
            active: false
```

Then, just launch the primary (**after the replicas are running!**):

```
$ gaiasky -p /path/to/config.primary.yaml
```

#### 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.

### 1.4.24 REST API

Gaia Sky provides a REST API feature that exposes the scripting API over the network via an HTTP server. This feature is used to *connect multiple instances* and to enable *the multi-projector setup in planetariums*.

> **Hint:** The REST API feature may permit remote code execution and open your machine to vulnerabilities. Only use the feature in a trusted environment!

**Contents**


- *REST API*
  - *Using the REST API*
  - *Debug*


In order to enable the REST API over an HTTP server in Gaia Sky, you need to modify the *configuration file*. The default location is `~/.config/gaiasky/config.yaml` in Linux and `[User.Home]\.gaiasky\config.yaml` in Windows and macOS. In that file, there is a property `program::net::restPort` (double colons indicate nesting) with the default value of -1. You can enable the REST server by setting this value to a positive integer number which will be the listening port of the server. For instance, we can start Gaia Sky with the REST server listening to the port 34487 with:

```yaml
program:
    net:
        restPort: 34487
```

Then, start Gaia Sky normally. You should see a couple of lines in the logs starting with `RESTServer` informing you that the REST API server is ready. Then, open your browser and point it to `http://localhost:34487/api`. You should get a JSON-formatted page documenting all the API calls available:

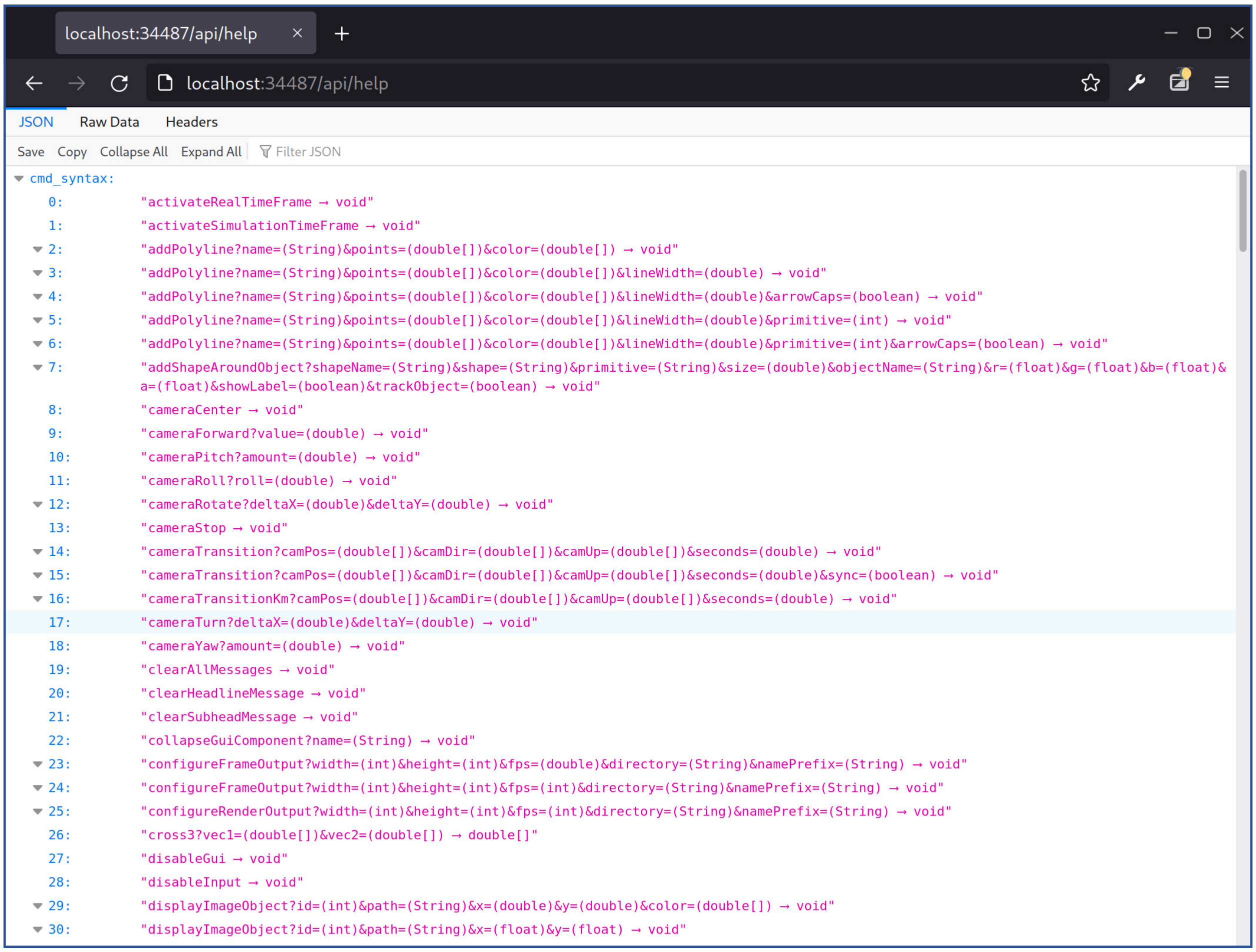



Fig. 75: The help page showing all REST API calls in Firefox

### Using the REST API

The API allows for developing additional software that interfaces with Gaia Sky without the need for language-specific bindings or inter-process communication protocols. Calling methods from the scripting interface `IScriptingInterface` is enabled locally and remotely via HTTP.

The syntax of API commands is set to be close to the Java method interface, but does not cover it in all generality to permit simple usage. Particularly note that the REST server receives strings from the client and will try to convert them into correct types.

Commands require HTTP request parameters having the names for the formal parameters of the script interface methods to allow simple construction of HTTP requests based on the scripting interface source documentation. We use Java reflections with access to the formal parameter names. Accordingly, the code needs to be compiled with `-parameters` (otherwise parameters are named `arg0`, `arg1`, ...).

Both GET and POST requests are accepted. Although GET requests are not supposed to have side effects, we include them for easy usage with a browser.

Issue commands with a syntax like the following:

- `http://localhost:PORT/api/setCameraUp?up=[1.,0.,0.]`
- `http://localhost:PORT/api/getScreenWidth`
- `http://localhost:PORT/api/goToObject?name=Jupiter&angle=32.9&focusWait=2`

Give booleans, integers, floats, doubles, strings as they are, vectors are comma-separated with square brackets around: `true`, `42`, `3.1`, `3.14877`, `Super-string`, `[1,2,3]`, `[Do,what,they,told,ya]`. Note that you might need to escape or URL-encode characters in a browser for this (e.g. spaces or "=").

Response with return data is in JSON format, containing key/value pairs. The `"success"` pair tells you about success/failure of the call, the `"value"` pair gives the return value. Void methods will contain a `"null"` return value. The `"text"` pair can give additional information on the call.

The `"cmd_syntax"` entry you get from the `help` command (e.g. `http://localhost:PORT/api/help`) gives a summary of permitted commands and their return type. Details on the meaning of the command and its parameters need to be found from the *scripting API documentation*.

### Debug

To examine, what happens during an API call, set the default log level of `SimpleLogger` to 'info' or lower (in the build file `core/build.gradle`).

Return values are given as JSON objects that contain key-value pairs:

- `"success"` indicates whether the API call was executed successful or not
- `"text"` may give additional text information
- `"value"` contains the return value or null if there is no return value

For testing with curl, a call like the following allows will deal with URL-encoding. The line below, when issued with a running Gaia Sky instance with the REST API server enabled listening to port 34487, will print the message "Hi, how are you?" in the Gaia Sky window:

```
curl "http://localhost:34487/api/setHeadlineMessage" --data headline='Hi, how are you?'
```

You can clear it with:

```
curl "http://localhost:34487/api/clearHeadlineMessage"
```

### 1.4.25 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:

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

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

#### Linux

- OBS Studio – amazing open source capturing and streaming solution.
- Simple Screen Recorder – the name says it all.

#### Windows

- OBS Studio – amazing open source capturing and streaming solution.
- FRAPS – 3rd party Direct3D and OpenGL recording software.
- NVIDIA Shadowplay – only for NVIDIA cards.

### 1.4.26 SAMP integration

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. The datasets must be curated as described in the *Preparing datasets* section.

#### 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`

## 1.4.27 Procedural planetary surfaces

Gaia Sky is able to procedurally generate planetary surfaces, cloud layers and also atmospheres. These can be applied to planets and moons to modify their looks. The elements that can be procedurally generated are, then, the **model surface**, the **cloud layer** and the **atmosphere**.

**Contents**


- *Procedural planetary surfaces*
  - *Surface generation process*
    - *Seamless (tilable) noise*
    - *Noise parametrization*
  - *Cloud generation process*
  - *Descriptor files*
    - *Randomize all*
    - *Surface description*
      - *Color look-up table*
      - *Noise parameters*
    - *Cloud description*
    - *Atmospheric parameters description*
  - *Procedural generation at runtime*


**Hint:** The techniques and methods behind the procedural generation of planetary surfaces in Gaia Sky are described in detail in this external article.

The procedural generation module is accessible via **two distinct ways**:

1. Specifying the procedural generation parameters in descriptor files. This way allows for the textual definition of bodies and their procedural generation parameters. The files can be loaded at startup and distributed so that other Gaia Sky users can load them.

2. Using the runtime procedural generation window to generate and modify surfaces, clouds and atmospheres. The results are only permanent during the current session and are lost on restart.

But first, let's learn about the process used to generate surfaces and clouds.

### Surface generation process

The surface generation process starts with the generation of the elevation and humidity data. The elevation data is a 2D array containing the elevation value in $[0, 1]$ at each coordinate. The humidity data is the same but it contains the humidity value, which will come in handy for the coloring. First, let's visit our sampling process.

### Seamless (tilable) noise

Usually, noise sampled directly is not tileable. The features do not repeat, and you just can't extend the noise indefinitely because seams are visible. In the case of one dimension, usually one would sample the noise using the only dimension available, $x$.


Fig. 76: Sampling noise in 1D leads to seams

However, if we go one dimension higher, 2D, and sample the noise along a circumference embedded in this two-dimensional space, we get seamless, tileable noise.

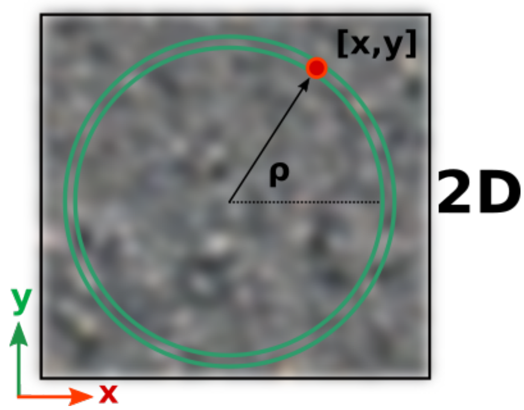



Fig. 77: Sampling noise along a circumference in 2D space is seamless

We can apply this same principle with any dimension $d$ by sampling in $d + 1$. Since we need to create spherical 2D maps, we do not sample the noise algorithm with the $x$ and $y$ coordinates of the pixel in image space. That would produce higher frequencies at the poles and lower around the equator. Additionally, the noise would contain seams, as it does not tile by default. Instead, we sample the 2D surface of a sphere of radius 1 embedded in a 3D volume, so we sample 3D noise. To do so, we iterate over the spherical coordinates $\varphi$ and $\theta$, and transform them to cartesian coordinates to sample the noise:

$$x = \cos\varphi \sin\theta$$
$$y = \sin\varphi \sin\theta$$
$$z = \cos\varphi$$

The process is outlined in this code snippet. If the final map resolution is $N \times M$, we use N $\theta$ steps and M $\varphi$ steps.

```
for (phi = -PI / 2; phi < PI / 2; phi += PI / M){
    for (theta = 0; theta < 2 * PI; theta += 2 * PI / N) {
        n = noise.sample(cos(phi) * cos(theta), // x
                         cos(phi) * sin(theta), // y
                         sin(phi));             // z
        theta += 2 * PI / N;
    }
}
```

### Noise parametrization

The generation is carried out by sampling configurable noise algorithms at different levels of detail, or octaves. To do that, we have some important noise parameters to adjust:

- **seed**—a number which is used as a seed for the noise RNG.
- **type**—the base noise type. Can be **gradient (Perlin) noise**[1], **gradval noise**[2], **simplex**[3], **value**[4] or **white**[5]. Find more information here.
- **fractal type**—the algorithm used to modify the noise in each octave. It determines the persistence (how the amplitude is modified) as well as the gain and the offset. Can be **billow**, **deCarpenterSwiss**, **FBM**, **hybrid multi**, **multi** or **ridge multi**. Find more information here.
- **scale**—determines the scale of the sampling volume. The noise is sampled on the 2D surface of a sphere embedded in a 3D volume to make it seamless. The scale stretches each of the dimensions of this sampling volume.
- **octaves**—the number of levels of detail. Each octave reduces the amplitude and increases the frequency of the noise by using the lacunarity parameter.
- **frequency**—the initial frequency of the first octave. Determines how much detail the noise has.
- **lacunarity**—determines how much detail is added or removed at each octave by modifying the frequency.
- **range**—the output of the noise generation stage is in $[0, 1]$ and gets map to the range specified in this parameter. Water gets mapped to negative values, so adding a range of $[-1, 1]$ will get roughly half of the surface submerged in water.
- **power**—power function exponent to apply to the output of the range stage.

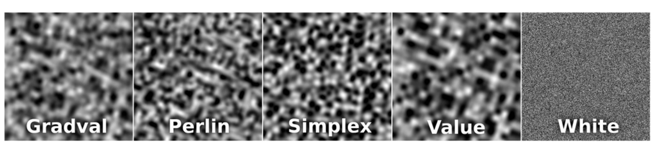



Fig. 78: The different types of noise, sampled raw with no fractals

The final stage of the procedural noise generation clamps the output to $[0, 1]$ again, so that all negative values are mapped to 0, and all values greater than 1 are clamped to 1.

Both elevation and humidity data use the same set of parameters right now. The elevation is used directly as the height texture. The humidity is used, together with the elevation, to determine the color using a look-up table. The humidity value is mapped to the X coordinate, while the elevation value is mapped to Y. Both coordinates are normalized to $[0, 1]$.

[1] https://en.wikipedia.org/wiki/Gradient_noise
[2] Addition of gradient and value noise
[3] https://en.wikipedia.org/wiki/Simplex_noise
[4] https://en.wikipedia.org/wiki/Value_noise
[5] https://en.wikipedia.org/wiki/White_noise

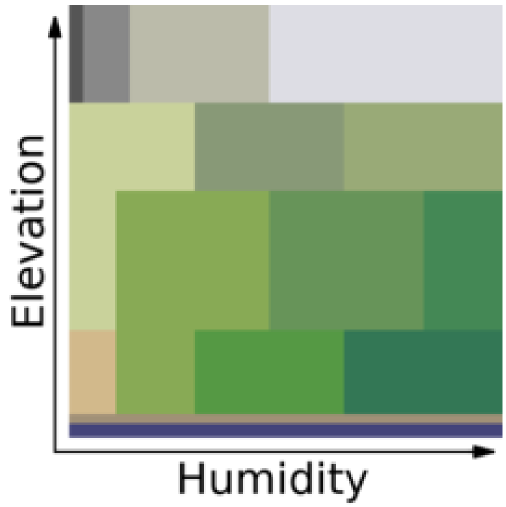



Fig. 79: The look-up table mapping dimensions are elevation and humidity

The look-up can also be hue-shifted by an extra **hue shift** parameter, in $[0°, 360°]$. The shift happens in the HSL color space. Once the shift is established, we generate the diffuse texture by sampling the look-up table and shifting the hue. The specular texture is generated by assigning all heights equal to zero to a full specular value. Remember that all negative values were clamped to zero, so zero essentially equals water in the final height map.

Finally, the normal map is generated from the height map by determining elevation gradients in both X and Y. We use the normal map only when tessellation is unavailable or disabled. Otherwise it is not generated at all.

### Cloud generation process

The clouds are generated with the same algorithm and the same parameters as the surface elevation. Then, an additional `color` parameter is used to color them. For the clouds to look better one can set a larger Z scale value compared to X and Y, so that the clouds are stretched in the directions perpendicular to the rotation axis of the planet.

### Descriptor files

This section describes how to set up the procedural generation using JSON descriptor files and how to express the parameters seen in the previous section in these descriptor files. The format is thoroughly documented in *this section*.

The procedural generation parameters for surfaces and clouds are described in the `material` and `cloud` elements. The `material` element lives inside the `model` element. By contrast, there are no procedural generation parameters that can be set in the `atmosphere` element itself. It just holds the atmospheric scattering parameters. However, the atmosphere element as a whole can be randomized. Let's see how to randomize these elements in the next section.

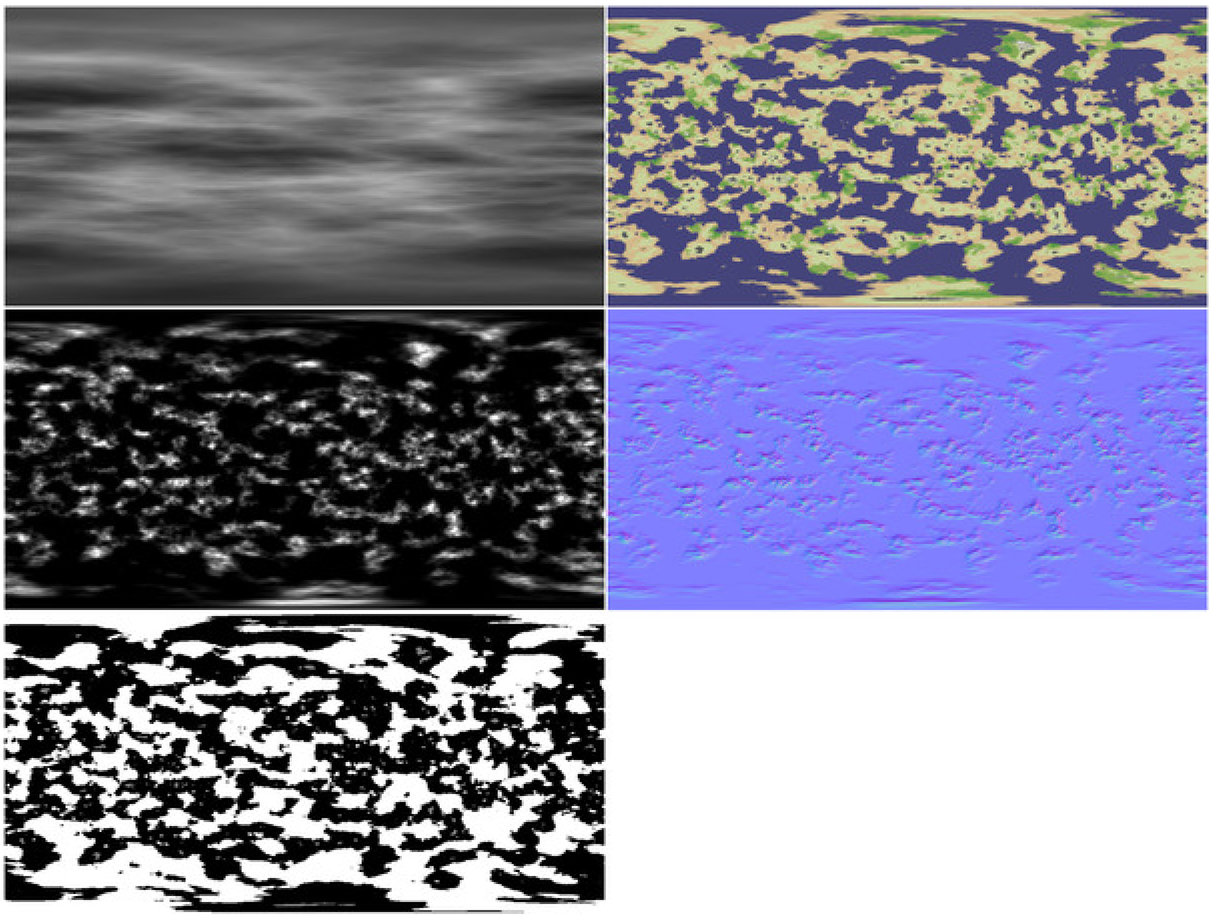


Fig. 80: Left to right and top to bottom, clouds map, diffuse texture, elevation map, normal map and specular map procedurally generated with Gaia Sky.

### Randomize all

The easiest way to add procedural generation to an object is by using the `randomize` element. It is an array which can contain the strings `"model"`, `"cloud"` and `"atmosphere"`. It can optionally be accompanied by a `seed` element, specifying the seeds for each of the elements to randomized. A seed is a 64-bit number used to initialize the RNG (random number generator) so that it always produces the same random number sequence. If you omit the seeds the system will randomly generate them. Otherwise, they are matched to elements by their order of appearance in the arrays. If the seeds array is not long enough, the first seed is used. Let's see an example:

```
{
    "name" : "Exonia f",

    "randomize" : [ "model", "cloud", "atmosphere" ],
    "seed" : [111, 222, 333]
}
```

In the snippet above we have omitted all the usual elements (`color`, `size`, `ct`, etc.) except the name. The last two elements specify the components to randomize and their seeds. In this case, the model would take the seed 111, the cloud would take the seed 222 and the atmosphere would take the seed 333.

If any of the elements were not present in the `randomize` array, it would not be generated. If the element object is present, it will be picked up though, but only if the `randomize` array does not contain it. The `randomize` array has precedence.

### Surface description

Some of the textures in the material element, making up the surface of the body, can be procedurally generated. The procedural generation parameters are specified in the `material` element inside the `model` element. Let's see an example:

```
"model"                 : {
    "args" : [true],
    "type" : "sphere",
    "params" : {
        "quality" : 400,
        "diameter" : 1.0,
        "flip" : false
    },
    "material" : {
        "height" : "generate",
        "diffuse" : "generate",
        "normal" : "generate",
        "specular" : "generate",
        "biomelut" : "data/tex/base/biome-smooth-lut.png",
        "biomehueshift" : -15.0,
        "heightScale" : 14.0,
        "noise" : {
            "seed" : 993390,
            "scale" : 0.1,
            "type" : "simplex",
            "fractaltype" : "ridgemulti",
            "frequency" : 5.34,
            "lacunarity" : 2.0,
```

(continues on next page)

(continued from previous page)

```
            "octaves" : 10,
            "range" : [-1.4, 1.0],
            "power" : 7.5
        },
    }
}
```

Usually, the `diffuse`, `height`, `normal` and `specular` elements contain texture image file locations. However, if they are set with the special token `"genearte"`, they will be procedurally generated by the system using the process described above.

### Color look-up table

The color look-up table is specified in the `biomelut` element as a pointer to a data file. The hue shift is specified in `biomehueshift`, and contains the shift value in degrees.

### Noise parameters

The noise parameters described in *this section* above can be specified in the `noise` attribute. The parameters translate 1-to-1 to what is described above, so they are pretty much already covered. If the noise parameters are not there, they are randomly initialized. These noise parameters are used to produce the elevation data and the humidity data.

### Cloud description

The clouds description goes in the `cloud` attribute. It contains the size of the clouds sphere (in km) and the parameters for the model. Then, in `cloud` we can either specify a texture image file, or we can use the reserved token `"generate"`. If this is there, we can specify the noise parameters just like in the material. If the noise parameters are not there, they are randomized automatically.

```
"cloud" : {
    "size" : 2430.0,
    "cloud" : "generate",
    "params" : {
        "quality" : 200,
        "diameter" : 2.0,
        "flip" : false
    }
    "noise" : {
        "seed" : 1234,
        "scale" : [1.0, 1.0, 0.4],
        "type" : "simplex",
        "fractaltype" : "ridgemulti",
        "frequency" : 4.34,
        "octaves" : 6,
        "range" : [-1.5, 0.4],
        "power" : 10.0
    }
}
```

### Atmospheric parameters description

The format for the atmospheric scattering parameters is documented in *this section*. If the value `atmosphere` is in the array of `randomize`, the atmospheric scattering parameters will be randomized automatically.

### Procedural generation at runtime

This section describes how to modify procedural generation parameters at runtime in real time.

You can bring up the procedural generation dialog by right clicking on any planet and/or moon, and then clicking on *Procedural generation. . . .* This brings up the following dialog.

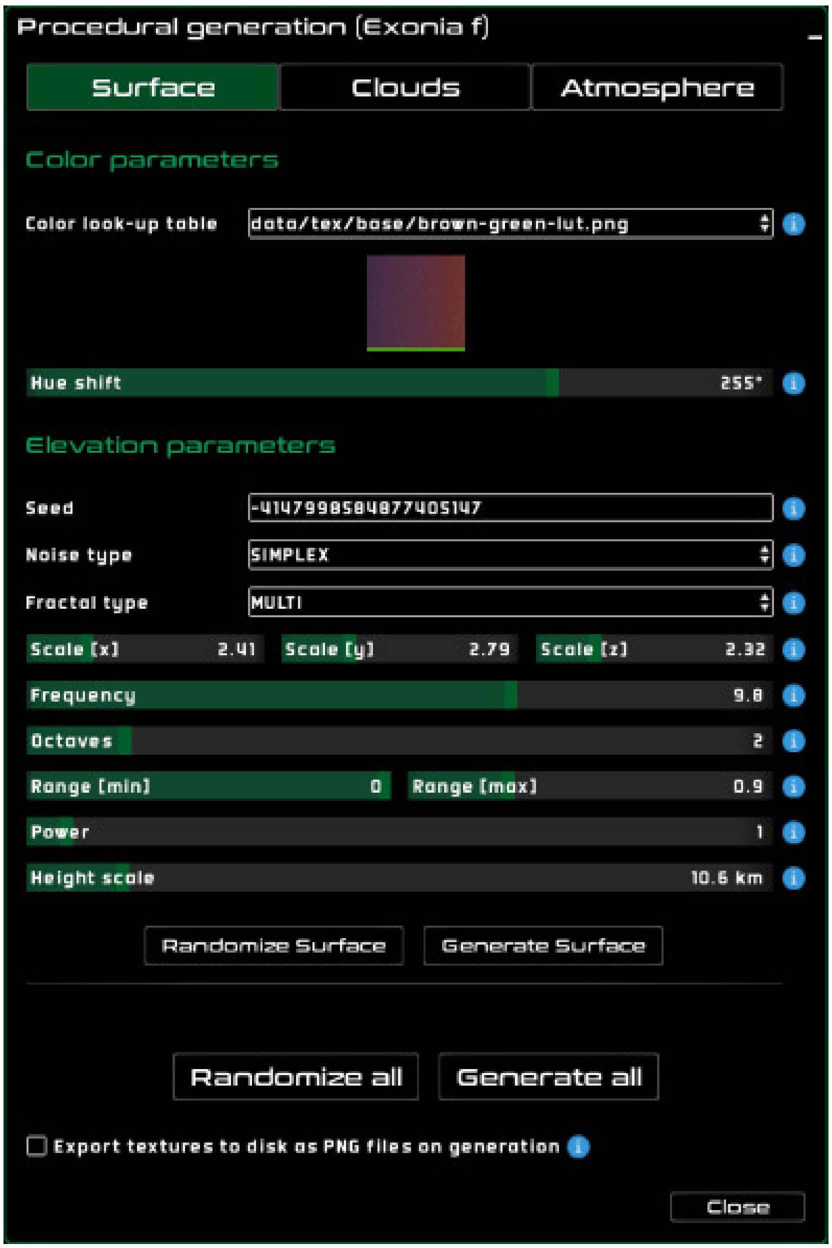



Fig. 81: The procedural generation dialog in Gaia Sky

In it, there are three tabs for surface, clouds and atmosphere. Use the controls in each tab to modify each of the procedural generation and atmospheric scattering parameters. Apply them in real time with the *generate* buttons. Use the *randomize* buttons to randomize all the parameters.

At the bottom, there is a check box to export all the generated textures to image files and save them to disk. The save location is the directory `tex/procedural` within your data directory.

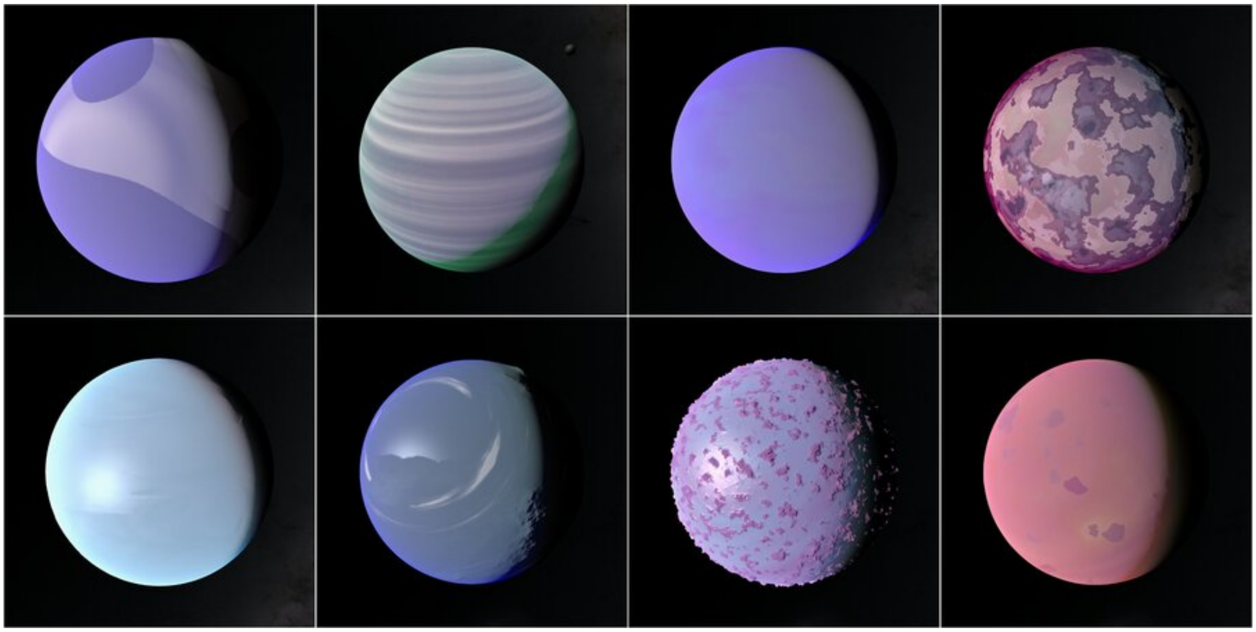


Fig. 82: A few planets created using the *randomize all* button.

## 1.4.28 System logs

Gaia Sky provides a couple of ways of accessing system logs.

### Session log

Gaia Sky always saves the log of the last session to `$GS_DATA/log/gaiasky_log_lastsession.log` (check where `$GS_DATA` is *here*). If you need to check the full log of your last session, you can always find it there.

### Crash reports

If Gaia Sky crashes, a crash report, together with a full session log to `$GS_DATA/crashreports` (check where `$GS_DATA` is *here*). Files with the form `gaiasky_crash_[date].txt` are crash reports, while files with the form `gaiasky_log_[date].txt` are full session logs. You can attach these whenever a crash happens and you want to submit a bug report to our buck tracker.

## 1.5 Advanced topics

Below are some chapters which include in-depth information about some of the internal workings of Gaia Sky. Things like the maximum allocated heap memory, the data format or the internal reference system are covered here.

### 1.5.1 The configuration file

There is a configuration file which stores the settings of Gaia Sky. This file is in the YAML format and is located in `$GS_CONFIG/config.yaml` (see *folders*). The default location is:

- Linux: `~/.config/gaiasky/config.yaml`
- Windows: `C:\Users\[username]\.gaiasky\config.yaml`
- macOS: `~/.gaiasky/config.yaml`

The default config.yaml file in our code repository is annotated with comments describing each setting.

**Contents**


- *The configuration file*
  - *Program settings*
  - *Controls settings*
  - *Graphics settings*
  - *Data settings*
  - *Scene settings*
  - *Post-processing settings*
  - *Proxy settings*


**The following sections document the settings that can only be modified by editing the configuration file itself**. The rest of settings can be edited from within Gaia Sky itself, usually using the preferences window or the control panel. A double colon `::` in the list below indicates nested settings.

#### Program settings

- `program::minimap::inWindow` – enables the rendering of the mini-map in a window.
- `program::net` – this group contains the configuration of the REST server, as well as the master-slave infrastructure. Find more information in the *connect instances* section.
- `program::scriptsLocation` – default location of script files in the file system.
- `program::ui::animationMs` – duration of UI animations in Gaia Sky, in milliseconds.
- `program::url` – contains the URLs for the version check (Codeberg API), the data repository mirror and the data descriptor file.
- `program::net` – contains the configuration of the REST API (port), and the master/slave instances. See *here* for more information.
- `program::offlineMode` – Gaia Sky won't attempt any HTTP connection to the internet in this mode. This means that the data descriptor file containing the information on server datasets can't be fetched. You need to

download the desired datasets manually and extract them in your data folder. More information can be found in our Gaia Sky datasets repository.

- `program::safeMode` – this is activated automatically whenever OpenGL incompatibilities are detected at startup. On macOS, this is on by default. Safe mode disables 'advanced' graphics features like 32-bit float buffers.

### Controls settings

- `controls::gamepad::blacklist` – a list of controller names to blacklist. You can find out the controller names recognized by Gaia Sky in the controls tab of the preferences window.

### Graphics settings

- `graphics::useSRGB` – use the sRGB color space as a frame buffer format. Only supported by OpenGL 3.2 and above. If this is activated, the internal format `GL_SRGB8_ALPHA8` is used. Only available when safe graphics mode is not active.
- `graphics::backBufferScale` – fixed scaling factor for the backbuffer. Increase this to improve image fidelity at the expense of performance. If dynamic resolution (*see this*) is enabled, this setting is ignored. This setting is exposed to the UI as "Dynamic resolution" in the experimental graphics settings section.

### Data settings

- `data::skybloxLocation` – contains the location of the default skybox used for reflections.

### Scene settings

- `scene::renderer::line::glWidthBias` – additive bias to add to the line width when rendering lines using the driver `GL_LINES` method. This is useful because the implementation of `GL_LINES` depends on the vendor (driver), and different implementations may interpret the line width differently.
- `scene::star::textureIndex` – the index of the texture used for stars. Star texture files are PNG files provided by the `default-data` package, and are of the form `$data/default-data/tex/base/star-texture-[NUM].png`.
- `scene::star::group::numLabel` – the maximum number of labels rendered by any star set. Be careful with increasing this value, as it may have very negative effects on performance with LOD catalogs (like most of Gaia DRx).
- `scene::octree::maxStars` – the maximum number of stars loaded at any single time from LOD catalogs.
- `scene::label::number` – controls the global number of stars in the scene by lowering the label solid angle threshold. Increase to get more labels, decrease to get less labels.
- `scene::initialization` – contains the `lazyTexture` and `lazyMesh` properties, which enable the lazy initialization of textures and meshes respectively.

#### Post-processing settings

- `postprocess::bloom::fboScale` – frame buffer scale factor (applied to the current viewport dimensions) to determine the frame buffer size to render the bloom effect.
- `postprocess::lensFlare::type` – choose the type of lens flare shader to use. Possible options are *SIMPLE* (a simple, nice-looking lens flare), *COMPLEX* (uses a complex and more demanding lens flare shader), and *PSEUDO* (uses a pseudo lens flare shader, described [here](https://john-chapman.github.io/2017/11/05/pseudo-lens-flare.html)).
- `postprocess::lensFlare::numGhosts` – number of ghost artifacts of the **pseudo lens flare** shader.
- `postprocess::lensFlare::haloWidth` – halo width of the **pseudo lens flare** shader.
- `postprocess::lensFlare::blurPasses` – number of blur passes for the **pseudo lens flare** shader.
- `postprocess::lensFlare::flareSaturation` – saturation value for the flare in the **pseudo lens flare** shader.
- `postprocess::lensFlare::bias` – bias value for the original image in the **pseudo lens flare** shader.
- `postprocess::lensFlare::texLensColor` – color lookup texture path for the **pseudo lens flare** shader.
- `postprocess::lensFlare::texLensDirt` – dirt texture path for all lens flare effects.
- `postprocess::lensFlare::texLensStarburst` – starburst texture path for all lens flare effects.
- `postprocess::lensFlare::fboScale` – scale of the frame buffer object to render the **pseudo lens flare** effect.
- `postprocess::lightGlow::samples` – number of samples to use to detect the brightness of the underlying star in the light glow effect/shader.
- `postprocess::warpingMesh::pfmFile` – absolute path to a PFM (portable float map) `.pfm` file that contains the warping mesh to apply. For more info, see *Mesh warping*.

#### Proxy settings

- `proxy` – configure an HTTP/HTTPS proxy. Find the full documentation to configure a proxy in the *proxy configuration section*.

### 1.5.2 Proxy configuration

If you need to configure Gaia Sky to use an HTTP, HTTPS, FTP or SOCKS proxy, you need to set it up at the Java virtual machine (JVM) level. The official documentation can be found here.

To configure a proxy, we need to pass some arguments to the JVM. Even though you can directly configure the proxy using JVM arguments, Gaia Sky offers an easier way to set this up using the *configuration file*. Using the configuration file has the advantage that it works the same way across all operating systems and packages.

**Note:** If your proxy requires authentication, please use the direct configuration below. Otherwise Java just ignores the `[protocol].proxy[User|Password]` properties, and the direct method ensures the authentication tokens are set up correctly.

### Use system proxy

The easiest way is to instruct Gaia Sky to use the proxy configured at the operating system level. To do so, open your `config.yaml` file (if you don't know where to find it, see *this*) you need to set the `proxy::useSystemProxies` property to `true` (`::` indicates nesting) in your configuration file:

```
proxy:
    useSystemProxies: true
```

If not set, this setting defaults to `false`.

### Direct configuration

Here you can enter the parameters of your proxy directly. The properties to set depend on the protocol.

#### HTTP

You can set the host, the port, the user credentials and the list of hosts that can bypass the proxy:

```
proxy:
    http:
        host: a.b.c.d
        port: 8080
        username: myname
        password: secret
        nonProxyHosts: a.b.c.d|e.f.g.*|localhost
```

- **host** – the hostname, or address, of the proxy server.
- **port** – the port number of the proxy server. Defaults to 80.
- **username** – the username, if you need authentication.
- **password** – the password, if you need authentication.
- **nonProxyHosts** – the hosts that should be accessed without going through the proxy. The value of this property is a list of hosts, separated by the '|' character. In addition, the wildcard character '*' can be used for pattern matching.

#### HTTPS

You can set the host, the protocol, the user credentials and the list of hosts that can bypass the proxy:

```
proxy:
    https:
        host: a.b.c.d
        port: 8080
        username: myname
        password: secret
        nonProxyHosts: a.b.c.d|e.f.g.*|localhost
```

- **host** – the hostname, or address, of the proxy server.
- **port** – the port number of the proxy server. Defaults to 80.

- **username** – the username, if you need authentication.
- **password** – the password, if you need authentication.
- **nonProxyHosts** – the hosts that should be accessed without going through the proxy. The value of this property is a list of hosts, separated by the '|' character. In addition, the wildcard character '*' can be used for pattern matching.

### SOCKS

You can set the host, the port, the username, the password and the SOCKS version:

```
proxy:
    socks:
        host: a.b.c.d
        port: 8080
        version: 5
        username: myname
        password: secret
```

- **host** – the hostname, or address, of the proxy server.
- **port** – the port number of the proxy server. Defaults to 80.
- **version** – the SOCKS protocol version. Defaults to 5, but can also be set to 4.
- **username** – the username, if you need authentication.
- **password** – the password, if you need authentication.

### FTP

You can set the host, the protocol, the user credentials and the list of hosts that can bypass the proxy:

```
proxy:
    ftp:
        host: a.b.c.d
        port: 8080
        username: myname
        password: secret
        nonProxyHosts: a.b.c.d|e.f.g.*|localhost
```

- **host** – the hostname, or address, of the proxy server.
- **port** – the port number of the proxy server. Defaults to 80.
- **username** – the username, if you need authentication.
- **password** – the password, if you need authentication.
- **nonProxyHosts** – the hosts that should be accessed without going through the proxy. The value of this property is a list of hosts, separated by the '|' character. In addition, the wildcard character '*' can be used for pattern matching.

### 1.5.3 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.

**Contents**


- *Performance*
  - *Maximum heap memory*
    - *Heap memory on Linux*
    - *Heap memory on Windows*
    - *Heap memory on macOS*
    - *Heap memory when running from source*
  - *Graphics performance*
  - *CPU performance*
    - *Multithreading*
    - *Limiting FPS*
    - *Draw distance (levels of detail)*
    - *Smooth transitions*


#### Maximum heap memory

Gaia Sky allocates a maximum heap memory value that can not be circumvented but can be adjusted or modified. If you encounter an `OutOfMemoryError`, chances are that your maximum heap memory is not enough for your usage. The default values are 4 GB (Gaia Sky `3.0.0` and below) and 6 GB (Gaia Sky `3.0.1+`).

In order to modify the maximum heap memory, follow the instructions below depending on your operating system.

#### Heap memory on Linux

On **Linux**, you need to edit the `gaiasky` executable script. It is usually located in `/opt/gaiasky/` when installed from your package manager, or wherever you extracted the package if installed from the `tar.gz`. Edit the script and find the line with `-Xmx?g`, where `?` is the default max heap memory. Change it to your desired value. For example, if you want to increase the maximum heap size to 12 GB, set it to `-Xmx12g`.

If installed using a `.deb` or `.rpm`, you need to edit the `/opt/gaiasky/gaiasky.vmoptions` file and uncomment and edit the line that reads:

```
# -Xmx512m
```

into:

```
-Xmx12g
```

Where `12g` is the desired amount of heap space.

#### Heap memory on Windows

On **Windows**, edit the file `gaiasky.vmoptions` in your Gaia Sky installation folder, and uncomment the line that reads `# -Xmx512m`, setting it to the heap space that you desire. So, in order to set the maximum heap to 12 GB, edit it from:

```
# Enter one VM parameter per line
# For example, to adjust the maixmum memory usage to 512 MB, uncomment the following␣
↪line:
# -Xmx512m
# To include another file, uncomment the following line:
# -include-options [path to other .vmoption file]
```

to:

```
# Enter one VM parameter per line
# For example, to adjust the maixmum memory usage to 512 MB, uncomment the following␣
↪line:
-Xmx12g
# To include another file, uncomment the following line:
# -include-options [path to other .vmoption file]
```

#### Heap memory on macOS

On **macOS**, you need to edit the file `vmoptions.txt` and uncomment the `-Xmx` line to suit your needs.

```
/Applications/Gaia\ Sky.app/Contents/vmoptions.txt
```

So, in order to set the maximum heap to 12 GB, edit the `/Applications/Gaia\ Sky/Contents/vmoptions.txt` from:

```
# Enter one VM parameter per line
# For example, to adjust the maixmum memory usage to 512 MB, uncomment the following␣
↪line:
# -Xmx512m
# To include another file, uncomment the following line:
# -include-options [path to other .vmoption file]
```

to:

```
# Enter one VM parameter per line
# For example, to adjust the maixmum memory usage to 512 MB, uncomment the following␣
↪line:
-Xmx12g
# To include another file, uncomment the following line:
# -include-options [path to other .vmoption file]
```

If you are not comfortable editing files from the terminal, you can just open the `Applications` folder in Finder, right-click on Gaia Sky and select "Show Package Contents". That gives you access to the application folder structure. Use Finder to navigate to `Gaia Sky.app/Contents/` and use your favorite text editor to edit `vmoptions.txt`.

### Heap memory when running from source

If you run from source you need to edit the `core/build.gradle` file. In there, you will find a `GaiaSkyRun` class with a `setup()` method. In this method, is a variable definition called `maxHeapSpace`, whose value you need to modify. The default value is `6g`, for 6 GB of maximum heap space. You can increase it at will.

### 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

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+`. You can configure whether you 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. 83: 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.

Find more in-depth information about this in the *data streaming section*.

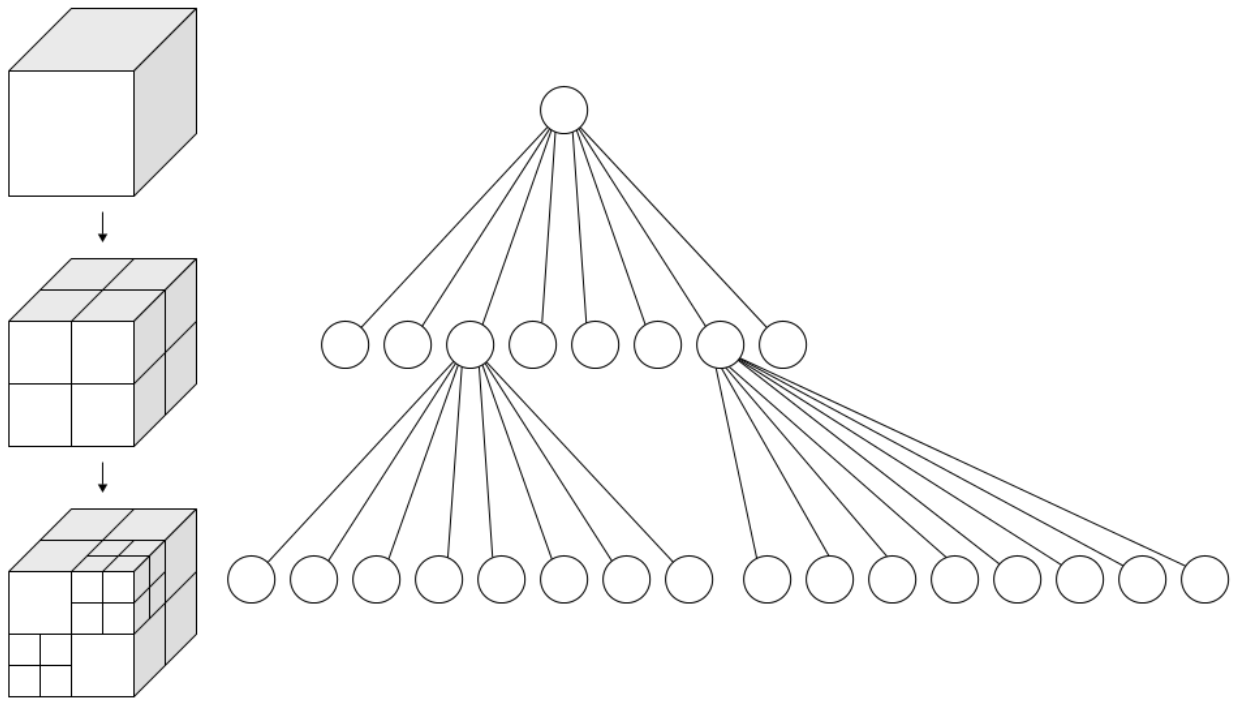


Fig. 84: Octree and levels of detail. Image: Wikipedia.

#### 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 peformance 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 angle and the draw distance angle + 0.4 rad.

### 1.5.4 Graphics performance

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

**Contents**


- *Graphics performance*
  - *Graphics quality setting*
  - *Dynamic resolution*
  - *Star brightness*
  - *Star groups*
    - *Billboards*
    - *Labels*

    * *Velocity vectors*
  - *Model detail*
  - *Bloom, lens flare and light glow*
  - *Antialiasing*


### Graphics quality setting

Please see the *Visual settings* section.

### Dynamic resolution

The dynamic resolution can improve the performance in demanding graphics situations and older hardware. See the *Visual settings* section for more information.

### Star brightness

The **star brightness** setting has an effect on the graphics performance because it causes more or less stars to be rendered using the close-by mode where the floating camera transformation is applied in the CPU and the vertices are computed and sent each frame. The effect on performance should not be too great though, unless your CPU is very old. The star brightness can be increased or decreased from the `Star brightness` slider in the *Visual settings pane* 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.

### Star groups

Star groups are an internal concept in Gaia Sky where a bunch of stars enter and leave the video memory together. Usually, a single catalog is loaded as a single star group, but it is not always the case. The main exception are the level-of-detail catalogs. In these, each octree node (octant) maps to a different star group.

A number of quantities are limited at the star group level. These are the maximum number of quad star billboards, the maximum number of labels and the maximum number of velocity vectors. All of these quantities have a rather strong impact on performance, and can be modified by editing the configuration file directly. They are not exposed in the GUI.

### Billboards

Stars, when close to the camera, are rendered with high quality billboards. Billboards are images which always face the camera (i.e. their normal vector is aligned with the vector that joins the camera position with the object's position).

The number of stars that will be rendered as billboards has a strong impact on performance, as we need to compute the quaternions to rotate the images correctly. This number is capped to a maximum value set in the configuration file. This number is set to 30 stars per star group by default, but you can edit it by editing the following line in your `config.yaml` file.

```yaml
scene:
    star:
        group:
            # Maximum number of billboards per star group
            numBillboard: 30
```

### 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 is used to scale the *threshold angle point*, which determines the solid angle boundary between rendering objects 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.

```
solid_angle > threshold_angle_point / label_factor
```

The label number factor impacts how many labels are displayed. You can modify this value by editing your `config.yaml` file.

```yaml
scene:
    label:
        # Label number factor. Controls how many stars have labels
        number: 1.3
```

Additionally, the maximum number of labels per star group has a huge impact on performance and is also defined in the configuration file. The default value is 50.

```yaml
scene:
    star:
        group:
            # Maximum number of labels per star group
            numLabels: 50
```

### Velocity vectors

When active, velocity vectors can become a big toll on performance. To mitigate that, you can adjust the number of vectors shown using the slider at the bottom of the type visibility pane.

Moreover, the maximum number of velocity vectors per star group is defined in the configuration file. The default value is 500.

```yaml
scene:
    star:
        group:
            # Maximum number of velocity vectors per star group
            numVelocityVector: 500
```

### Model detail

Some models (mainly spherical planets, planetoids, moons and asteroids) are automatically generated when the Gaia Sky is initializing 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:

```
{
    "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 modeling 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`.

```
{
    "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 fairly capable GPUs, but might be on integrated graphics.
- The **lens flare** effect is a bit harder on the GPU, but most modern cards should be able to handle it with no problems. In order of cost, from less costly to more costly, the shaders are *SIMPLE*, *PSEUDO*, *COMPLEX*.
- 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 *graphics configuration section*.

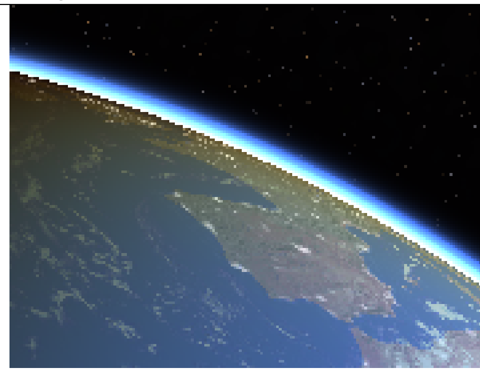
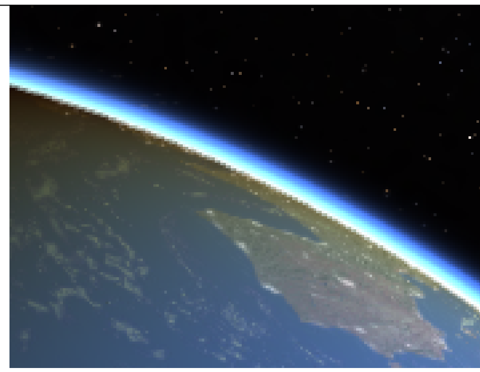
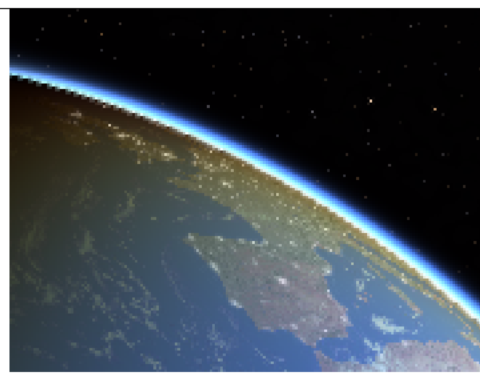
### Antialiasing

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 anti-aliasing 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.

| Name | Type | Description |
|---|---|---|
| **No Antialiasing** | No antialiasing | This has no cost since it does not apply any antialiasing technique. |
| **FXAA** | Post-processing | This has a mild performance cost and produces reasonably good results. If you have a good graphics card, this is super-fast. |
| **NFAA** | Post-processing | 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. |

Here are some sample images.

| Name | Image |
|---|---|
| **No Antialiasing** |  |
| **FXAA** |  |
| **NFAA** |  |

Some graphics drivers allow you to override the anti-aliasing settings of applications with some default configuration (usually MSAA or FXAA). To use this, select *No antialiasing* in Gaia Sky.

Find more information on anti-aliasing in the *Visual settings* section.

### 1.5.5 Internal reference system

The internal cartesian reference system is a right-handed equatorial system with the particularity that the axes labels are unorthodox. Usually, $X$ points to the fundamental direction ($\alpha = 0, \delta = 0$), $Z$ points "up" and $XY$ is the fundamental plane ($\delta = 0$), with $Y = Z \times X$.

In our case, it is $Z$ which points to the fundamental direction ($\alpha = 0, \delta = 0$), $Y$ points up and $XZ$ is the fundamental plane ($\delta = 0$), with $X = Y \times Z$. In order to convert from common equatorial cartesian coordinates $(XYZ)$ to Gaia Sky coordinates $(X'Y'Z')$, you just need to swap the axes:

- $X' = Y$

- $Y' = Z$
- $Z' = X$

Or, what is the same, $(X'Y'Z') = (YZX)$, and $(XYZ) = (Z'X'Y')$.

### Description

So, in Gaia Sky $XZ$ is the equatorial plane ($\delta = 0$). $Z$ points towards the vernal equinox point ($\alpha = 0, \delta = 0$). $Y$ points towards the north celestial pole ($\delta = +90°$). $X$ is perpendicular to both $Z$ and $Y$ and points to $\alpha = +90°$ so that $X = Y \times Z$.

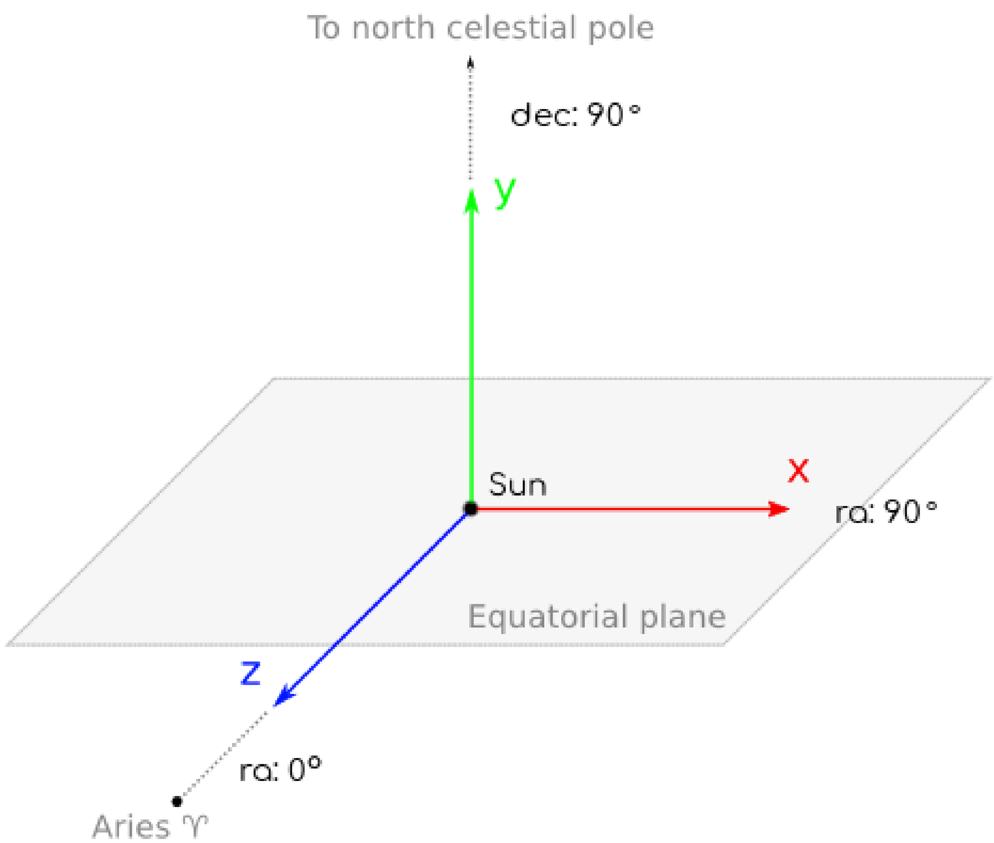



Fig. 85: Gaia Sky reference system

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.

### Internal units

Internally, the objects in Gaia Sky are positioned using Internal Units. The default Internal Units ($iu$) are defined as follows:

- $1iu = 1 * 10^9 m$

When running in Virtual Reality mode, and only for the duration of the session, the Internal Units are scaled as follows:

- $1iu = 1 * 10^5 m$

## 1.5.6 Data format

Gaia Sky needs to first load datasets in order to display data. Dataset files contain objects, which are organized by Gaia Sky into a `scenegraph`. A scenegraph is a tree that contains objects and orders them hierarchically depending on their geometrical and spatial relations.

**Contents**


- *Data format*
  - *Where are the data files defined?*
    - *$data/[dataset-name]/dataset.json example*
    - *default-data/dataset.json example file*
  - *Data loaders*
  - *Catalog formats*
    - *Star catalogs*
      - *Regular star catalogs*
      - *Level-of-detail star catalogs*
    - *Particle catalogs*
  - *JSON data format*
    - *Data morphology*
    - *Objects vs Updates*
    - *Basic attributes*
    - *Proper motions*
    - *Magnitudes*
    - *Labels*
    - *Coordinates and ephemerides*
      - *Orbit coordinates*
      - *Static coordinates*
      - *VSOP87*
      - *VSOP2000*
      - *Chebyshev polynomials*

- *Heliotropic orbits*
- *Moon AA coordinates*
- *Pluto coordinates*
- *Python scripting coordinates*
* *Model objects*
  - *Orientation*
  - *Model*
  - *Clouds*
  - *Atmospheric scattering parameters*
* *Mesh objects*
* *Orbits*
* *Grids and other special objects*
* *Affine transformations*
* *Reference system transformations*
– *Creating your own catalog loaders*
– *Loading data using scripts*


All datasets are partially or totally described in a JSON format. Each dataset lives in its own directory in the data location (referred to as `$data/`, see *folders*), and must contain a description in the file `dataset.json`. If a dataset does not have this file in its directory, it won't be recognized by Gaia Sky.

Below is an example of the contents of the data location.

```
$data/
├─ default-data/
│  ├─ dataset.json
│  └─ ...
├─ catalog-hipparcos/
│  ├─ dataset.json
│  └─ ...
├─ catalog-whitedwarfs-dr2/
│  ├─ dataset.json
│  └─ ...
└─ .../
```

### Where are the data files defined?

Gaia Sky implements a very flexible and open data loading mechanism. The data files to be loaded are defined in a couple of keys in the `config.yaml` *configuration file*, which is usually located in the `$GS_CONFIG` folder (see *folders*). The keys are the following (double-colon indicates nesting):

- `data::dataFiles` – an array containing the list of enabled JSON data files. Each file should be a relative path from the data directory with the prefix `$data/`. For instance, the default dataset, containing the Solar System and some necessary objects for Gaia Sky to run, is specified in the array as `$data/default-data/dataset.json`.

### $data/[dataset-name]/dataset.json example

Dataset descriptor files contain the metadata of a catalog (name, description, version, etc.) and a pointer to the actual data. Below is a made-up file file `dataset.json` which describes my super-awesome dataset.

```
{
"key" : "dataset-key-without-spaces",
"name": "Dataset name",
"version": 1,
"mingsversion" : 30106,
"type": "catalog-gaia",
"description": "The description here.\nCan contain line breaks.",
"releasenotes" : "- What changed since the last version?\n- List here."
"link": "https://arxiv.org/abs/1805.00425",
"check" : "dataset-descriptor.json"
"size": 368633,
"nobjects": 1365,
"check": "$data/my-dataset/dataset.json",
"files": [ "$data/my-dataset" ],
"data": [{
    "loader": "gaiasky.data.JsonLoader",
    "files": [ "$data/my-dataset/particles-particles.json" ]
}]
}
```

Notice that the data (`data::files`) points to another JSON file, which contains some additional info about how to load the data, and a pointer to the actual data file `particles-particles.json`. Here it is:

```
{
"objects": [{
    "name": "My dataset",
    "position": [0.0, 0.0, 0.0],
    "componentType": "Stars",
    "fadeout": [1.0e5, 0.5e8],
    "parent": "Universe",
    "archetype": "StarGroup",
    "provider": "gaiasky.data.group.STILDataProvider",
    "datafile": "$data/my-dataset/catalog/dataset.vot"
    }]
}
```

As you can see, the `STILDataProvider` is the one in charge of loading the data form this dataset, which resides in a VOTable file, `dataset.vot`.

### default-data/dataset.json example file

This is an example of what the default data pack contains. The dataset descriptor file loads different files using different loaders.

```
{
"data" : [
    {
        "loader": "gaiasky.data.JsonLoader",
        "files": [ "$data/default-data/planets-normal.json",
                   "$data/default-data/moons-normal.json",
                   "$data/default-data/satellites.json",
                   "$data/default-data/asteroids.json",
                   "$data/default-data/orbits_planet.json",
                   "$data/default-data/orbits_moon.json",
                   "$data/default-data/orbits_asteroid.json",
                   "$data/default-data/orbits_satellite.json",
                   "$data/default-data/extra-low.json",
                   "$data/default-data/locations.json",
                   "$data/default-data/locations_earth.json",
                   "$data/default-data/locations_moon.json"]
    },
    {
        "loader": "gaiasky.data.GeoJsonLoader",
        "files": [ "$data/default-data/countries/countries.geo.json" ]
    }]
}
```

The `dataset.json` file contains an array, `"data"`, 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. The main loader, the `JsonLoader`, expects JSON files as inputs. Each of these files must have an attribute called `"objects"`, which is an array containing the metadata on the objects to load.

### Data loaders

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 `ISceneLoader` (here). The `loadData()` method of each loader must return a list of 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 JSON files containing planets, satellites, orbits, star catalogs, etc. The `STILDataProvider` (here) loads `VOTables`, `FITS`, `CSV` and other files through the STIL library, `GeoJsonLoader` (here) loads geographic data, and so on.

### Catalog formats

Catalogs refer to datasets which are essentially particle-based (stars, galaxies, etc.). There are several off-the-shelf options to get catalog data in various formats into Gaia Sky. The most important are VOTable, FITS and CSV. They are all handled by the *STIL data provider*. The way they are defined in Gaia Sky is the same any other object is defined, that is, using *JSON descriptor files*.

Let's see an example of the definition of one such catalog (the Oort cloud) using 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],
    "componentType" : "Others",

    "fadeIn" : [0.0004, 0.004],
    "fadeOut" : [0.1, 15.0],

    "profileDecay" : 1.0,

    "parent" : "Universe",
    "archetype" : "ParticleGroup",

    "provider" : "gaiasky.data.PointDataProvider",
    "factor" :  149.597871,
    "dataFile" : "$data/oort-cloud/oortcloud/oort_10000particles.dat"
}
```

This is based on the ParticleSet component, which is fully documented *here*. Let's go over the attributes that appear in this example:

- 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.
- componentType (alias: ct) – The ComponentType (see 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 inetrpolation 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 inetrpolation 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 - dist)^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 scene graph.
- `archetype` (alias: `impl`) – The *archetype* name (or legacy class name, but this should be avoided).
- `provider` – The full name of the data provider class. This must extend `gaiasky.data.api.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.
- `texture` – Optional attribute that points to a texture or directory with textures to render the particles of this set with. If this is available, `profileDecay` is ignored.
- `textures` – Same as `texture`, but with an array of textures and/or directories.

### Star catalogs

Star catalogs are special because, additionally to positional information, they contain extra properties such as proper motions, magnitudes, colors and more. All of these are important to be able to render stars faithfully.

The easiest way to load star catalogs is by loading them from VOTable files. Let's see how these catalogs can be defined in Gaia Sky. For example, the new Hipparcos reduction uses this `dataset.json` file that contains some catalog metadata and pointers to the actual data files:

```
{
  "key": "catalog-hipparcos",
  "name" : "Hipparcos (new reduction)",
  "version" : 4,
  "mingsversion" : 30301,
  "type" : "catalog-star",
  "description" : "Hipparcos new reduction (van Leeuwen, 2007) with curated star names.",
  "releasenotes" : "- Add type in catalog descriptor file.\n- Update to new data format.
↪",
  "link" : "http://adsabs.harvard.edu/abs/2007ASSL..350.....V",
  "size" : 5433174,
  "nobjects" : 177955,
  "check": "$data/catalog-hipparcos/dataset.json",
  "files" : [ "$data/catalog-hipparcos" ],
  "data" : [
  {
    "loader": "gaiasky.data.JsonLoader",
    "files": [ "$data/catalog-hipparcos/particles-hip.json" ]
  }]
}
```

The file `particles-hip.json` contains a single object with the actual pointer to the VOTable data file, and some additional metadata such as the color of labels, a description of the catalog or the data provider:

```
{
  "objects" : [
  {
```

(continues on next page)

(continued from previous page)

```
    "name" : "Hipparcos (new red.)",
    "position" : [0.0, 0.0, 0.0],
    "color" : [1.0, 1.0, 1.0, 0.25],
    "size" : 6.0,
    "labelColor" : [1.0, 1.0, 1.0, 1.0],
    "labelPosition" : [0.0, -5.0e7, -4.0e8],
    "componentType" : "Stars",

    "fadeout" : [21.0e2, 0.5e5],

    "profiledecay" : 1.0,

    "parent" : "Universe",
    "archetype" : "StarGroup",

    "catalogInfo" : {
      "name" : "Hipparcos",
      "description" : "Hipparcos new reduction (van Leeuwen, 2007). 117995 stars.",
      "type" : "INTERNAL"
    },

    "provider" : "gaiasky.data.group.STILDataProvider",
    "dataFile" : "$data/catalog-hipparcos/catalog/hipparcos/hipparcos.vot"
  }
]}
```

### Regular star catalogs

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`, `CSV`, `FITS` or `ASCII` file into Gaia Sky using the `Open file` icon at the bottom of the control panel.

### Level-of-detail star catalogs

Gaia Sky uses level-of-detail structures to represent catalogs with hundreds of millions of stars. This broad and deep topic is covered in its own section:

- *Level-of-detail: Octree*.

### Particle catalogs

Particle catalogs (point cloud data) can be loaded from CSV, VOTable or FITS files (see *STIL data loader*), or also from a fast and compact binary format. More information can be found in the *particle catalogs section*.

### JSON data format

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 object model, which is a scene graph tree.

An example about defining an extrasolar system with a couple of stars orbiting each other and a couple of planets can be found *here*.

### Data morphology

Before starting, we need to do a little detour to cover the data morphology. In Gaia Sky, objects organize and store their data into *components*, conforming to the Entity Component System (ECS) paradigm. Components are simple bags of data (attributes). Objects in Gaia Sky are assigned an *archetype*, which are simple component groups.

- **Archetype** – an *archetype* is a definition of a group of components to create objects of a certain type. Ever object has **one and only one** archetype. The archetype is specified with the `"archetype"` or the `"impl"` attributes, and take in the name of the archetype (case sensitive!). For instance, the *Planet* archetype contains, amongst others, the components *Base*, *Model*, *Coordinates* and *Atmosphere*.
- **Component** – a *component* is a simple bag of data. For example, the *Base* component contains a color, the object type, and the object name or names. Components are mostly hidden from users, as they are not defined directly in the JSON data files. Instead, components define what attributes are accepted by each object. An exhaustive list of all the attributes per component, together with descriptions and data types, is provided in *Components*.

Depending on the archetype of an object (i.e. depending on the components it has), objects are processed differently by different systems. Additionally, archetypes can extend other archetypes, so that the extending archetype gets all the components defined in the parent.

You can find a full description of all the archetypes and components in the data format, together with their attributes, here:

- *Archetypes*.
- *Components*.

### Objects vs Updates

Every JSON file that contains objects must have a named array as the only top-level object in the file. Depending on the name of this array, two things can happen:

- `objects` – when the array is named *objects*, it contains new objects to load.
- `updates` – when the array is named *updates*, it contains updates to pre-existing objects.

Objects in `updates` arrays are kept and applied at the end of the loading stage, when all objects in `objects` array have been loaded. They are matched by name.

So far, only the following objects and attributes can be updated:

- `material` – material and all its sub-attributes. In particular all, regular textures, cubemaps and virtual textures: - `diffuse`, `diffuseCubemap`, `diffuseSVT`. - `specular`, `specularCubemap`, `specularSVT`. - `normal`, `normalCubemap`, `normalSVT`. - `height`, `heightCubemap`, `heightSVT`. - `emissive`, `emissiveCubemap`, `emissiveSVT`. - `metallic`, `metallicCubemap`, `metallicSVT`. - `roughness`, `roughnessCubemap`, `roughnessSVT`.
- `cloud` – describes the cloud layer. Can also have a virtual texture. - `diffuse`, `diffuseCubemap`, `diffuseSVT`.
- `atmosphere` – all its direct attributes.
- `rotation` – all its direct attributes.

See the *virtual textures section* for some examples.

### Basic attributes

All *archetypes* have the *Base*, the *Body* and the *GraphNode* components. These components hold basic attributes, which can be specified with these (usually required) keys:

- `name` – The name of the object. You can specify multiple names in a string array by using the `names` key.
- `color` – The color of the object. This will translate to the line color in orbits, to the color of the point for planets when they are far away and to the color of the grid in grids.
- `componentType` (alias: `ct`) – The `ComponentType` (see 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.
- `archetype` (alias: `impl`) – The *archetype* name (or legacy 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:

- `sizeKm` (with variants `sizePc`, `sizeM`, `sizeAU`, `size`) – The diameter of the entity. The unitless version uses internal units.
- `pos` (with variants `position`, `positionKm`, `positionPc`, `posKm`, `posPc`) – The position of the object. This is the position at epoch (if the object has proper motion), or just a static position. Given in cartesian coordinates in the internal reference system.
- `labelColor` – Color of the label of this object.

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,
    "componentType" : "Equatorial",

    "parent" : "Universe",
    "archetype" : "SphericalGrid"
}
```

### Proper motions

Objects of an *archetype* with a *ProperMotion component* can define proper motion attributes:

- `muAlpha` (`muAlphaMasYr`) – The $\mu_{\alpha\star}$, in mas/yr.
- `muDelta` (`muDeltaMasYr`) – The $\mu_{\delta}$, in mas/yr.
- `radialVelocity` (`radialVelocityKms`) – The radial velocity in km/s.
- `epochJd` (`epochYear`) – The proper motion epoch as a Julian date, or as a year fraction (2015.5).

### Magnitudes

Objects of an *archetype* with a *Magnitude component* can define an apparent and absolute magnitudes.

- `appMag` – The apparent magnitude.
- `absMag` – The absolute magnitude.

The apparent and absolute magnitudes are only used in celestial bodies. In stars, if only one is set, the other is computed automatically. If both are set, **consistency is not checked together with the distance**. Also in stars, the absolute magnitude is used to compute a pseudo-size which is used for rendering purposes only. See the *star rendering section* for more information.

### Labels

All labels in Gaia Sky are applied the component type of the object they are attached to, plus the "Labels" component type. Here are some of the attributes related to labels. Attributes marked with a star (*) can only be applied to objects whose *archetype* has a *Label component*.

- `label`* – Whether to render the label at all. Takes in a boolean.
- `labelColor` – The color of the label, as a RGBA array.
- `forceLabel` – Whether to force-display the label for this entity, regardless of distance and size.
- `labelPositionPc`* (`labelPositionKm`, `labelPosition`) – Override the position of the label.
- `labelFactor`* – Factor to apply to the size of the label for this object.
- `labelMax`* – Internal rendering factor, should not be set externally unless you know what you are doing.
- `textScale`* – Internal rendering factor, should not be set externally unless you know what you are doing.

### Coordinates and ephemerides

Within the `coordinates` object (see the *Coordinates component*) 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 few implementations that can be used with the `"impl"` attribute. They are described in the following sub-sections.

### Orbit coordinates

Using `OrbitLintCoordinates`, the coordinates of the object are linearly interpolated using its orbit, which is defined in a separated entity. See the *orbits section* for more info. The `name` of the orbit entity must be given. For instance, the Hygieia moon uses orbit coordinates.

```
{
    "coordinates" : {
        "impl" : "gaiasky.util.coord.OrbitLintCoordinates",
        "orbitName" : "Hygieia orbit"
    }
}
```

### Static coordinates

Use `StaticCoordinates` to specify a static position (or a position at epoch for entities with proper motion). This is equivalent to using the top-level `pos` attribute, which also specifies the position at epoch. Static coordinates can also be applied a transformation using the `transformMatrix` and `transformName` attributes. The position may be given in cartesian or spherical coordinates.

- `position` (with variants: `positionKm`, `positionPc`) – position in cartesian coordinates in the internal reference system.
- `positionEquatorial` – equatorial coordinates ($\alpha$ [deg], $\delta$ [deg], and distance [parsecs]).
- `positionEcliptic` – ecliptic coordinates ($\lambda$ [deg], $\beta$ [deg], and distance [parsecs]).
- `positionGalactic` – ecliptic coordinates (l [deg], b [deg], and distance [parsecs]).

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

### VSOP87

All classes that extend `AbstractVSOP87` provide ephemerides for the major planets. These implement the `VSOP87` analytical solution. Our implementation of VSOP87 contains a class for each body, with all the terms hard-coded. For instance, to set up VSOP87 ephemeris for the Earth use the following:

```
{
    "coordinates" : {
        "impl" : "gaiasky.util.coord.vsop87.EarthVSOP87",
        "orbitName" : "Earth orbit"
    }
}
```

#### VSOP2000

Implementation of the analytical planetary solution VSOP2000. You just need to point to the file for the particular bodies. Data files are available for Mercury, Venus, Earth, Moon, the Earth-Moon barycentre, Mars, Jupiter, Saturn, Uranus, Neptune and Pluto in ftp://syrte.obspm.fr/francou/vsop2000/. In order to use those, use the VSOP2000 class with the desired data file.

```
{
    "coordinates" : {
        "impl" : "gaiasky.util.coord.vsop2000.VSOP2000",
        "dataFile" : "/path/to/data/vsop2000-p03.dat",
        "orbitName" : "Earth orbit"
    }
}
```

#### Chebyshev polynomials

Implementation of Chebyshev polynomials using the coefficients to compute the Ephemeris. Just like VSOP2000, each body needs a different data file, containing the coefficients for the body.

```
{
    "coordinates" : {
        "impl" : "gaiasky.util.coord.chebyshev.ChebyshevEphemeris",
        "dataFile" : "/path/to/data/EARTH.position.data",
        "orbitName" : "Earth orbit"
    }
}
```

#### Heliotropic orbits

Provides coordinates of objects in heliotropic orbits using those orbits' data, like Gaia or JWST. The implementation is in `HeliotropicOrbitCoordinates`.

#### Moon AA coordinates

`MoonAACoordinates` contains a special implementation of the algorithm described in the book Astronomical Algorithms by Jean Meeus that provides the position of the Moon.

#### Pluto coordinates

`PlutoCoordinates` is a special implementation, described here, which provides very fast but not very accurate positions for Pluto.

### Python scripting coordinates

`PythonBodyCoordinates`, reserved for coordinate providers implemented in Python via scripting. This object uses `IPythonCoordinatesProvider` instances implemented in a Python script to source coordinates. For more information, see *this section*.

### Model objects

Planets, moons, asteroids, etc. all use the model object `Planet` (here). This provides a series of utilities that make their JSON specifications look similar.

### Orientation

Orientations in Gaia Sky may be given in two different formats:

- *Rigid rotation* – the orientation is described with basic rotation parameters such as the period, inclination, axial tilt, etc.
- *Quaternion orientation* – the orientation of the object is sourced from an entity that provides quaternions.

**Rigid rotation**

The `rigidRotation` map (aliased to `rotation`) describes, as you may imagine, the rigid rotation of the body in question by means of a series of parameters. Rotations are stored in the *Orientation component*, and use the `rigidRotation` map. A rigid 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:

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

**Quaternion orientation**

Some objects like satellites are typically oriented using quaternions. Since every satellite may have its own attitude analytical implementation, we support a generic way of providing quaternions, based on the orientation provider and source. The orientation provider contains the name of a class that extends OrientationServer.

We provide two general implementations, based on the spherical linear interpolation of quaternions (QuaternionSlerpOrientationServer) and on normalized linear interpolation (QuaternionNlerpOrientationServer. These implementations read the quaternion data from a CSV file in the following format:

```
time-iso-8601,x,y,z,w
```

For example, a valid quaternion slerp file would be `quaternions.csv`:

```
#time,x,y,z,w
2020-01-01T12:00:00Z,0.0,0.0,0.0,1.0
2020-02-01T12:00:00Z,1.0,0.0,0.0,0.0
2020-03-01T12:00:00Z,0.0,1.0,0.0,0.0
2020-04-01T12:00:00Z,0.0,0.0,1.0,0.0
2020-05-01T12:00:00Z,0.0,0.0,0.0,1.0
```

Each line contains:

- **Time [ISO-8601]** – the time of the quaternion.
- **x** – x component.
- **y** – y component.
- **z** – z component.
- **w** – w component.

Once we have the file, we can use it in our object by using the *Orientation* properties `orientationProvider` and `orientationSource`.

```
{
    "orientationProvider": "gaiasky.data.orientation.QuaternionSlerpOrientationServer",
    "orientationSource": "path/to/quaternions.csv"
}
```

We also offer a specific implementation of OrientationServer for Gaia in the form of the GaiaAttitudeServer.

### Model

This section describes the format to specify models, but omits the procedural generation attributes. These are documented in the *procedural generation section*.

The model object describes the model which must be used to represent the entity. Models are described in the *Model component*, and can have two origins:

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

```
{
    "model": {
      "args" : [true],
      "model" : "$data/default-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.

```
{
    "model": {
      "args" : [true],
```

(continues on next page)

(continued from previous page)

```
      "type" : "sphere",
      "staticLight" : true,
      "useColor" : "false",
      "ambientLevel" : 0.5,
      "ambientColor" : [0.1, 0.3, 0.6, 1.0],
      "params" : {
        "quality" : 180,
        "diameter" : 1.0,
        "flip" : false
        },
      "material" : {
        "diffuse" : "$data/default-data/tex/base/earth-day*.jpg",
        "diffuseCubemap" : "$data/default-data/tex/cubemap/earth-day*",
        "diffuseSVT" : {
          "location" : "$data/virtualtex-earth-diffuse/tex",
          "tileSize" : 1024
        },
        "specular" : "$data/default-data/tex/base/earth-specular*.jpg",
        "normal" : "$data/default-data/tex/base/earth-normal*.jpg",
        "emissive" : "$data/default-data/tex/base/earth-night*.jpg",
        "hieght" : "$data/default-data/tex/base/earth-height*.jpg",
        "heightScale" : 8.12,
        "reflection" : [ 1.0, 1.0, 0.0 ]
    }
}
```

- `type` – the type of model. Possible values are `sphere`, `disc`, `cylinder` and `ring`.
- `staticLight` – this attribute takes in a boolean (`true` or `false`) or a floating point number. If present, this disconnects the ambient light of this model from the global ambient level, **and** does not apply directional lighting to the model. If set to `true`, the ambient level of this model is set to the default 0.6. Otherwise, it is set to the given floating-point value in [0,1].
- `useColor` – a boolean that indicates that the object color (in the `color` attribute) is to be used as the model color. If this is true, the object color is set as a model diffuse color attribute.
- `ambientLevel` – a single floating-point value with the ambient light level (in [0,1]) to apply to this model. If present, the model is disconnected from the global ambient light setting.
- `ambientColor` – a 3- or 4-component color (RGB or RGBA) for the ambient light of this model. If present, the model is disconnected from the global ambient light setting.
- `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.
  - `diffuse` – the diffuse texture to use.
  - `diffuseCubemap` – the location of the 6 sides of the diffuse cubemap to use. Takes precedence over `diffuse`. The sides must be images with the `_bk.jpg`, `_ft.jpg`, `_up.jpg`, `_dn.jpg`, `_rt.jpg`, `_lf.jpg` suffixes. The file formats can be JPG or PNG. Can be applied to all channels (specular, normal, emissive, height, metallic, roughness, etc.) More information on this can be found in the *cubemaps section*.
  - `diffuseSVT` – an object with a `location` (path) and a `tileSize` (integer). Defines a diffuse virtual texture for this model. Can be applied to more channels. More information on this cam be found in the

*virtual texture section*.

- specular – the specular map to produce specular reflections. This attribute also accepts a specular index or a specular color (RGB). More than one can be specified.
- normal – normal map to produce extra detail in the lighting.
- emissive – emissive texture, color or value. For planets, this acts as the night texture, which is applied to the part of the model in the shade. This attribute also accepts an emissive color (RGB) and an emissive index.
- height – height map which will be represented with tessellation or parallax mapping (see *elevation (height)*) and whose scale is defined in heightScale (in Km).
- heightScale – indicates the extent, in Km, of the top mapping value in the height map (corresponding to full white, or RGB [1,1,1]). When the elevation multiplier slider is set to 1, the highest point in the height map is displaced by this amount of kilometers.
- roughness – roughness texture, color or value for the PBR shader.
- metallic – metallic texture or value for the PBR shader.
- ao – ambient occlusion texture for the PBR shader.
- diffuseScattering – a color (vec[3]) or a single floating point number with the diffuse scattering value for this model. Diffuse scattering weighs the diffuse color and re-emits it if there are no shadows.
- occlusionMetallicRoughness – occlusion/metallic/roughness texture (in R, G and B channels respectively) texture for the PBR shader. Follows the glTF specification.
- 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.

Additionally, we may use the following attributes for **ringed** models, also in the material group:

- ringDiffuse – diffuse texture for the ring.
- ringNormal – diffuse texture for the ring.
- ringDiffuseScattering – a color (vec[3]) or a single floating point number with the diffuse scattering value for this model. Diffuse scattering weighs the diffuse color and re-emits it if there are no shadows.

## Clouds

This defines the clouds layer (see *Cloud component*). It can be *procedurally generated* or described with textures. Here we deal only with the textures mode. Let's see:

```
"cloud"         : {
    "size" : 6395.0,
    "cloud" : "$data/default-data/tex/base/earth-cloud*.jpg",

    "params" : {
        "quality" : 200,
        "diameter" : 2.0,
        "flip" : false
    }
}
```

Contains the `size` of the cloud model, its `parameters` (`quality`, `diameter`, etc.) and the clouds texture. The clouds are combined with the planet using the equation:

$$\bar{C}_{result} = \bar{C}_{source} \times \bar{F}_{source} + \bar{C}_{destination} \times \bar{F}_{destination}$$

where $\bar{F}_{source}$ is 1, and $\bar{F}_{destination}$ is $1 - \bar{C}_{source}$.

### Atmospheric scattering parameters

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

```
{
    "atmosphere" : {
        "size" : 6600.0,
        "wavelengths" : [0.650, 0.570, 0.475],
        "m_Kr" : 0.0025,
        "m_Km" : 0.0015,
        "m_eSun" : 1.0,
        "fogdensity" : 2.5,
        "fogcolor" : [1.0, 0.7, 0.6],

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

The parameters are the following:

- `size` – radius of the sphere model used for the atmosphere, in km.
- `wavelengths`– the values of $\frac{1}{\lambda^4}$ for the red, green and blue channels. These are the Rayleigh scattering rates of different light wavelengths.
- `Kr` – Rayleigh scattering constant.
- `Km` – Mie scattering constant.
- `eSun` – the brightness of the illuminating star.
- `fog density` – density of the simulated fog when inside the atmosphere.
- `fog color` – the color of the fog.

### Mesh objects

Gaia Sky supports Galaxy-size arbitrary meshes. These are usually used to represent iso-density surfaces for stars, dust or HII regions, among others. Mesh objects have all the regular attributes of model bodies (`name`, `description`, `color`, `size`, etc.). Additionally, we offer three shading modes for meshes:

- `additive` – renders the mesh with transparency via additive blending. The DR2 hot star and HII density meshes use this shading mode.

- `dust` – renders an opaque mesh. An opacity value is computed for the edges ($V \cdot N$), where $V$ is the pixel view vector from the camera and $N$ is the normal vector at that pixel. Opacity is rendered using dithering to avoid sorting issues.
- `regular` – renders the mesh with the regular, general-purpose per-pixel lighting shader.

In order to specify the shading mode, a new top-level attribute `"shading" : "additive|dust|regular"` must be used:

```
{
    "name" : "DR3 star density",
    "description" : "Star density iso-surface based on DR3 data",
    "color" : [0.95, 0.2, 0.2, 0.75],
    "size" :  3.0856775814913705E7,
    "labelcolor" : [0.95, 0.1, 0.1, 1.0],
    "shading" : "regular",
    "labelposition" : [1000.0, 0.0, 0.0],
    "componentType" : "Meshes",

    "fadeout" : [60000.0, 90000.0],

    "parent" : "Universe",
    "archetype" : "MeshObject",

    "transformName" : "galacticToEquatorialF",

    "model"         : {
        "args" : [true],
        "staticLight" : true,
        "model" : "$data/mesh-dr3-stardenstiy/meshes/dr3/star_density.obj"
    }
}
```

### 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. Orbits are stored in the *Trajectory component*.

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

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

    "parent" : "Sol",
    "archetype" : "Orbit",
    "provider" : "gaiasky.data.orbit.OrbitFileDataProvider",
```

(continues on next page)

(continued from previous page)

```
    "orbit" : {
        "source" : "$data/default-data/orb.VENUS.dat",
    }
}
```

If the orbit is defined with its orbital elements, the elements need to be specified in the `orbit` object. For example, the orbit of **Phobos**.

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

    "parent" : "Mars",
    "archetype" : "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,
        "mu" : 9.9e18
    }
}
```

The orbit object is represented with the orbital elements:

- `period` – in days.
- `epoch` – in Julian days.
- `semiMajorAxis` – in km.
- `eccentricity` – no units.
- `inclination` – in degrees.
- `ascendingNode` – in degrees.
- `argOfPericenter` – in degrees.
- `meanAnomaly` – in degrees.
- `mu` – $G * M$ of central body (gravitational constant). Defaults to the Sun's. This will be anyway automatically recomputed from the period ($T$) and the semi-major axis ($a$), if set, as $\mu = 4 * \pi^2 a^3 / T^2$.

Note that if the `epoch` is fully defined, the `argOfPericenter` is not needed.

The orbital elements of extrasolar systems are typically given in a special reference system. In this reference system, the **reference plane** is the plane whose normal is the line of sight vector from the Sun to the planet or star for whom the orbit is defined. The **reference direction** is the direction from the object to the north celestial pole projected on the reference plane. In order to apply such a transformation automatically, Gaia Sky provides an additional attribute "model" to objects of the *archetype* Orbit. When this attribute has the value "extasolar_system", the abovementioned transformation is applied automatically. The default value of "model" is "default". Below is an example:

```
{
   "name": "J0805+4812 star orbit",
   "color": [
     1.0,
     0.0,
     1.0,
     1.0
   ],
   "componentType": [
     "Orbits",
     "Stars"
   ],
   "parent": "J0805+4812 Center",
   "archetype": "Orbit",
   "provider": "gaiasky.data.orbit.OrbitalParametersProvider",
   "model": "extrasolar_system",
   "newMethod": true,

   "trail" : true,
   "trailMap" : 0.5,

   "fadeDistanceUp" : 50.0,
   "fadeDistanceDown" : 100.0,

   "orbit": {
     "period": 735.9078666506588,
     "epoch": 2457397.4170103897,
     "semiMajorAxis": 48464416.969729796,
     "eccentricity": 0.4203524474493615,
     "inclination": 107.89617887219426,
     "ascendingNode": 175.09066812003118,
     "argOfPericenter": 326.7130552945523,
     "meanAnomaly": 0.0,
     "mu": 9.9998e+21
   }
 }
```

At object level, we can set the following attributes to control some visual properties:

- trail – fades the orbit as it gets further away from the object in the direction opposite to travel. By default, the end closest to the object position is mapped to 1, and the furthermost position from the object is mapped to 0.
- trailMap – the bottom mapping position for the trail. The orbit trail assigns an opacity value to each point of the orbit, where 1 is the location of the object and 0 is the other end. This mapping parameter defines the location in the orbit (in [0,1]) where we map the opacity value of 0. Set to 0 to have a full trail. Set to 0.5 to have a trail that spans half the orbit. Set to 1 to have no orbit at all.
- fadeDistanceUp – orbits with a body fade out as the camera get closer to the body. This is the far distance, in body radius units, where the orbit starts the fade (mapped to 1). This attribute only has effect if this trajectory has a body attached to it.
- fadeDistanceDown – orbits with a body fade out as the camera get closer to the body. This is the near distance, in body radius units, where the orbit finishes the fade (mapped to 0). This attribute only has effect if this trajectory

has a body attached to it.

### 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.

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

    "parent" : "Universe",
    "archetype" : "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.

### Affine transformations

Model objects (meshes, shapes, models, etc.) of an *archetype* that contains an *AffineTransformations component* can define arbitrary affine transformations (rotation, translation, scale) in any order, as top-level attributes. These transformations will be applied to the local transformation matrix of the model in the same order they are defined in the JSON file.

The supported attributes, and their names, are:

- `translate` – contains a 3-vector with a translation in *internal units*.

  *Example:* `"translate" :  [2.0, 0.0, 5.0]`.

- `translatePc` – contains a 3-vector with a translation in parsecs.

  *Example:* `"translatePc" :  [2.0, 0.0, 5.0]`.

- `translateKm` – contains a 3-vector with a translation in parsecs.

  *Example:* `"translateKm" :  [25000.0, 0.0, 0.0]`.

- `scale` – contains either a single floating-point value or a 3-vector with the scaling factor, in local model coordinates.

  *Example:* `"scale" :  [1.0, 1.0, 2.0]`, scales the model $\times 2$ in the Z component.

- `rotate` – contains a 4-vector where the first three components are the rotation axis, and the last component is the rotation angle in degrees.

  *Example:* `"rotate":  [1.0, 0.0, 0.0, 45.0]`, rotates the model $45°$ around the $\begin{pmatrix}1 & 0 & 0\end{pmatrix}$ vector.

For instance, the following JSON object,

```json
{
  "name": "Object name",
  "color": [ 0.4, 0.4, 1.0, 0.4 ],
  "labelColor": [ 0.4, 0.4, 1.0, 1.0 ],
  "labelFactor": 0.0004,
  "sizePc": 1.0,
  "componentType": "Others",
  "parent": "Parent name",
  "archetype": "ShapeObject",
  "renderGroup": "MODEL_VERT_ADDITIVE",

  "rotate": [ 0.0, 1.0, 0.0, 60.0 ],
  "scale": [ 2.0, 0.5, 0.5 ],

  "coordinates": {
    "impl": "gaiasky.util.coord.StaticCoordinates",
    "position": [ 0.0, 0.0, 0.0 ]
  },
  "model": {
    "args": [ true ],
    "staticLight": true,
    "type": "sphere",
    "primitiveType": "lines",
    "blendMode": "additive",
    "params": { "quality": 30, "diameter": 1.0, "flip": true }
  }
}
```

defines a sphere which is scaled $\times 2$ in X and $\times 0.5$ in Y and Z, and rotated around the Y axis $60°$.

### Reference system transformations

Gaia Sky uses an equatorial *internal reference system*. Objects of an *archetype* that contains a *RefSysTransform component* can specify reference system transformations directly in the JSON file as top-level attributes using the `"transformName"` or `"transformFunction"` names. These attributes take in a string with the name of the transformation. The possible values are:

- "transformFunction": **"equatorialToEcliptic"**
- "transformFunction": **"equatorialToGalactic"**
- "transformFunction": **"eclipticToEquatorial"**
- "transformFunction": **"eclipticToGalactic"**
- "transformFunction": **"galacticToEquatorial"**
- "transformFunction": **"galacticToEcliptic"**

These essentially get transform to a 4x4 matrix with the necessary reference system rotation.

Additionally, it is possible to specify the 16 values of the matrix themselves, in **column-major** order, using the top-level attributes `"transformMatrix"` or the alias `"transformValues"`.

*Example:*

```json
{
  "name": "Object name",
  "color": [ 0.4, 0.4, 1.0, 0.4 ],
  "labelColor": [ 0.4, 0.4, 1.0, 1.0 ],
  "labelFactor": 0.0004,
  "sizePc": 1.0,
  "componentType": "Others",
  "parent": "Parent name",
  "archetype": "ShapeObject",
  "renderGroup": "MODEL_VERT_ADDITIVE",

  "transformMatrix": [0.7660444431189781, 0.0, -0.12855752193730788, 0.0,
                      0.0,                0.5,  0.0,                 0.0,
                      0.6427876096865394, 0.0,  0.15320888862379564, 0.0,
                      0.0,                0.0,  0.0,                 1.0],
  "coordinates": {
    "impl": "gaiasky.util.coord.StaticCoordinates",
    "position": [ 0.0, 0.0, 0.0 ]
  },
  "model": {
    "args": [ true ],
    "staticLight": true,
    "type": "sphere",
    "primitiveType": "lines",
    "blendMode": "additive",
    "params": { "quality": 30, "diameter": 1.0, "flip": true }
  }
}
```

### Creating your own catalog loaders

If you want to load your data files into Gaia Sky, chances are that the *STIL data provider* can already do it.

If you still need to create your own loader, keep reading.

In order to create a loader for your catalog, one only needs to provide an implementation to the `ISceneLoader` (here) interface.

```java
public interface ISceneLoader {
  public List<Entity> loadData() throws FileNotFoundException;
  public void initialize(String[] files, Scene scene) throws RuntimeException;
}
```

The main method to implement is `List<Entity> loadData()` (here), which must return a list of `Entity` objects.

But how do we know which file to load? You need to create a `dataset.json` descriptor 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[], Scene)` (here) method knowing that all the properties defined in the `dataset.json` file with your catalog loader as a prefix will be passed in the `Properties p` 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 `config.yaml` file (usually under `$GS_CONFIG`, see *folders*) and add your new file to the property `data::catalogFiles`. You can also drop your new catalog into a subdirectory in the data directory and enable it using the dataset manager in Gaia Sky.

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.

You can use existing loaders as examples, such as the `OctreeLoader` (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.

### 1.5.7 STIL data loader

Gaia Sky supports the loading of catalog data in **VOTable**, **FITS**, **CSV** and **ASCII**. using the STIL library. To ensure the catalogs are loaded correctly, some preparation might be needed in the form of UCDs and/or column names and units. The following sections describe the expected UCDs and column names for the different data types and units.

**Contents**


- *STIL data loader*
  - *Object IDs*
  - *Object names*
  - *Positions*
  - *Proper motions and radial velocities*
  - *Magnitudes*
  - *Colors*
  - *Variability*
  - *Other columns*


The class in charge of loading data using STIL is the STILDataProvider.

**Note:** In all cases, UCDs take precedence over column names. That is, if a UCD is present for a given column, the column name is ignored. This means that if the UCD is incorrect, the column data won't be recognized and used even if the column name is correct.

#### Object 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

### Object names

Names are taken from the columns `name`, `proper`, `proper_name`, `common_name` and `designation`.

Also, the regular expression `"(name|NAME|refname|REFNAME)((_|-)[\\w\\d]+)?"` is matched against column names to find names. This matches anything which starts with `name` or `NAME` or `refname` plus an optional suffix starting with a hyphen or an underscore.

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 UCDs supported:

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

The units should be specified as column metadata. If units are not there, Gaia Sky will use degrees for coordinate angles (ra, dec, lat, lon, etc.), mas for parallaxes and parsecs for distances.

If UCDs are not possible (i.e. CSV format), the sky positions should be given in the equatorial system and have the following column names and units:

- Right ascension: `ra`, `right_ascension`, `alpha` in degrees
- Declination: `dec`, `de`, `declination`, `delta` in degrees

To work out the distance, it looks for the UCDs `pos.parallax` and `pos.distance`. If either of those are found, they are used. If no UCDs are to be found, the column names `plx`, `parallax`, `pllx` and `par` are accepted. If there are no parallaxes, the default parallax of 0.04 mas is used. As previously mentioned, parallaxes are in mas by default, and distances are in parsecs, unless stated otherwise in column unit metadata.

With respect to cartesian coordinates, it recognizes the UCDs `pos.cartesian.x|y|z`, and they are interpreted in the equatorial system by default.

### 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.

Apparent magnitudes are converted to absolute magnitudes with:

$$M = m - 5log_{10}(d_{pc}) + 5$$

where $M$ is the absolute magnitude and $m$ is the apparent magnitude. $d_{pc}$ is the distance to the star in parsecs.

The absolute magnitude is then converted to a pseudo-size with an algorithm that converts first to a luminosity, and then adjusts the size with an experimental calibration.

#### 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 as the color index.

The conversion from color index to RGB is done by converting the XP (BP-RP) color index to $T_{eff}$, and then the $T_{eff}$ to RGB, using the `xp_to_teff()` and `teff_to_rgb()` methods implemented here.

#### Variability

Variable stars are loaded if light curves (magnitude vs time) and periods are found in the column list. The magnitude list, time list and period are looked up using their column names:

- **Magnitude list**: A list of [mag] is expected under `g_transit_mag`, `g_mag_list`, `g_mag_series`
- **Time list**: A list of Julian dates (offset from J2010, i.e. $t = JD - 2455197.5$) under `g_transit_time`, `time_list`, `time_series`
- **Period**: A period in Julian days under `pf`, `period`

Only variable stars with a period will be loaded. The rest will be skipped.

#### Other columns

All the columns which do not fit in the aforementioned categories are loaded as extra attributes. These attributes can be used for filtering and color mapping the dataset.

Right now, additional physical quantities (mass, flux, effective temperature ($T_{eff}$), radius, etc.) fall into the 'other columns' category and are also loaded as extra attributes.

### 1.5.8 Star catalog formats

Star catalogs can be loaded from well-known formats (VOTable, CSV, etc.) using the *STIL data loader*, or they can use a binary format tailor-made for Gaia Sky. In general, the binary format loads **much** faster and is more compact. That's why we use it for our big level-of-detail star catalogs based on Gaia data.

**Contents**

- *Star catalog formats*

- *Binary format specification*
  - *Metadata file*
    - *Version 0*
    - *Version 1*
  - *Star particle files*
    - *Version 0*
    - *Version 1*
    - *Version 2*
- *LOD catalog processing*
  - *Catalogs*
  - *Distances*
  - *Magnitude/color corrections*


This section discusses the level-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 $\theta$. $\theta$ is actually the minimum visual solid angle (as seen from the camera) of an octant for it to be observed and its stars to be rendered. Larger $\theta$ values lead to less octants being observed, and smaller $\theta$ 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 , $\theta$.
- The maximum number of loaded stars, $\nu$. This is in the configuration file (`$GS_CONFIG/config.yaml`) under the key `scene::octree::maxstars`. The default value balances the *maximum heap memory space* and the default data set.

So basically, a low $\theta$ (below 50-60 degrees) means lots of observed octants and lots of stars. Setting $\theta$ 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 *maximum heap space*.

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 $\theta$, 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 $\theta$. That is why high values for $\theta$ are recommended. Usually, values between 60 and 80 are fine, depending on the dataset and the machine.

$\theta$ Draw distance, minimum visual solid angle for octants to be rendered

$\nu$ Maximum number of stars in memory at a given time

### Binary format specification

Gaia catalogs contain typically hundreds of millions of stars. They are too large to fit in your neighbor's consumer GPU. In order to be able to represent such catalogs in real time, Gaia Sky implements a level-of-detail algorithm backed by an octree. The data format of all level-of-detail catalogs is a custom binary format to make it more compact and fast to load. This binary format can, however, also be used for smaller star catalogs. This section contains its specification.

There are two types of files: the metadata (`metadata.bin`) and the particle files (`particles_xxxxxxx.bin`). The metadata file contains all the nodes of the octree (called octants). Each octant points to a particle file, containing its particles. The number in the particle file name is the identifier of the octant. Additionally, the particle files can also be used for standalone smaller star catalogs.

The distance units are *internal units*.

#### Metadata file

The metadata reader is implemented here. The metadata file contains the information of the octants of the octree. The metadata format has currently two possible versions, 0 and 1, which are automatically detected by Gaia Sky.

#### Version 0

Version 0 (legacy) does not contain its version number in the file itself. Instead, if the first four bytes interpreted as an integer are zero or positive, version 0 is assumed. The format is the following.

- 1 single-precision integer (32-bit) – number of octants in the file
- For each octant:
  - 1 single-precision integer (32-bit) – Page ID - ID of current octant
  - 3 single-precision float (32-bit * 3) – X, Y, Z cartesian coordinates in internal units
  - 1 single-precision float (32-bit) – Octant half-size in X
  - 1 single-precision float (32-bit) – Octant half-size in Y
  - 1 single-precision float (32-bit) – Octant half-size in Z
  - 8 single-precision integer (32-bit * 8) – IDs of the 8 children (-1 if no child)
  - 1 single-precision integer (32-bit) – Level of octant (depth)
  - 1 single-precision integer (32-bit) – Cumulative number of stars in this node and its descendants
  - 1 single-precision integer (32-bit) – Number of stars in this node
  - 1 single-precision integer (32-bit) – Number of children nodes

#### Version 1

Version 1 was introduced in Gaia Sky `3.0.4`, and starts with a negative integer in the first four bytes, typically -1. Then comes the version number. The main difference with the legacy version is that the page IDs are encoded with a 64-bit integer instead of 32.

- 1 single-precision integer (32-bit) – special token number `-1`, signaling the presence of a version number
- 1 single-precision integer (32-bit) – version number (`1` in this case)
- 1 single-precision integer (32-bit) – number of octants in the file

- For each octant:
  - 1 double-precision integer (64-bit) – Page ID - ID of current octant
  - 3 single-precision float (32-bit * 3) – X, Y, Z cartesian coordinates in internal units
  - 1 single-precision float (32-bit) – Octant half-size in X
  - 1 single-precision float (32-bit) – Octant half-size in Y
  - 1 single-precision float (32-bit) – Octant half-size in Z
  - 8 double-precision integer (64-bit * 8) – IDs of the 8 children (-1 if no child)
  - 1 single-precision integer (32-bit) – Level of octant (depth)
  - 1 single-precision integer (32-bit) – Cumulative number of stars in this node and its descendants
  - 1 single-precision integer (32-bit) – Number of stars in this node
  - 1 single-precision integer (32-bit) – Number of children nodes

### Star particle files

A particle file contains the information of a number of stars. These can be the stars belonging to a particular octant in a LOD octree, or all the stars in a particular star catalog.

The class in charge of loading and writing binary star particle files is the BinaryDataProvider.

The binary readers/writers are implemented in the following files:

- Interface (BinaryIO)
- Base implementation (BinaryIOBase)
- Version 0 (DR1 and DR2, now outdated)
- Version 1 (used in the currently public eDR3 catalogs, same as version 0, but without tycho IDs)
- Version 2 (the new version, much more compact and small)

Version 0 was used in DR2, version 1 was used mainly in the first batch of eDR3. Version 2 is used in the second batch of eDR3 and future DRs. Versions 0 and 1 are not annotated, so they are detected using the file name. Starting from version 2, the version number is in the file header, using a special token (negative integer).

### Version 0

The version 0 is specified below. It contains a header with the number of stars and then a bunch of data for each star. It contains a 3-integer set which is the Tycho identifier, mainly for compatibility with TGAS.

- 1 single-precision integer (32-bit) – number of stars in the file
- For each star:
  - 3 double-precision floats (64-bit * 3) – X, Y, Z cartesian coordinates in internal units
  - 3 double-precision floats (64-bit * 3) – Vx, Vy, Vz - cartesian velocity vector in internal units per year
  - 3 double-precision floats (64-bit * 3) – mualpha, mudelta, radvel - proper motion
  - 4 single-precision floats (32-bit * 4) – appmag, absmag, color, size - Magnitudes, colors (encoded), and size (a derived quantity, for rendering)
  - 1 single-precision integer (32-bit) – HIP number (if any, otherwise negative)

- 3 single-precision integer (32-bit * 3) – Tycho identifiers
- 1 double-precision integer (64-bit) – Gaia SourceID
- 1 single-precision integer (32-bit) – namelen -> Length of name
- namelen * char (16-bit * namelen) – Characters of the star name, where each character is encoded with UTF-16

### Version 1

Version 1 is the same as version 0 but without the Tycho identifiers.

- 1 single-precision integer (32-bit) – number of stars in the file
- For each star:
  - 3 double-precision floats (64-bit * 3) – X, Y, Z cartesian coordinates in internal units
  - 3 double-precision floats (64-bit * 3) – Vx, Vy, Vz - cartesian velocity vector in internal units per year
  - 3 double-precision floats (64-bit * 3) – mualpha, mudelta, radvel - proper motion
  - 4 single-precision floats (32-bit * 4) – appmag, absmag, color, size - Magnitudes, colors (encoded), and size (a derived quantity, for rendering)
  - 1 single-precision integer (32-bit) – HIP number (if any, otherwise negative)
  - 1 double-precision integer (64-bit) – Gaia SourceID
  - 1 single-precision integer (32-bit) – namelen -> Length of name
  - namelen * char (16-bit * namelen) – Characters of the star name, where each character is encoded with UTF-16

### Version 2

This version is much more compact, and it uses smaller data types when possible. The header contains a token integer (-1) marking the following version number, plus the number of stars.

- 1 single-precision integer (32-bit) – special token number `-1`, signaling the presence of a version number
- 1 single-precision integer (32-bit) – version number (`2` in this case)
- 1 single-precision integer (32-bit) – number of stars in the file
- For each star:
  - 3 double-precision floats (64-bit * 3) – X, Y, Z cartesian coordinates in internal units
  - 3 single-precision floats (32-bit * 3) – Vx, Vy, Vz - cartesian velocity vector in internal units per year
  - 3 single-precision floats (32-bit * 3) – mualpha, mudelta, radvel - proper motion
  - 4 single-precision floats (32-bit * 4) – appmag, absmag, color, size - Magnitudes, colors (encoded), and size (a derived quantity, for rendering)
  - 1 single-precision integer (32-bit) – HIP number (if any, otherwise negative)
  - 1 double-precision integer (64-bit) – Gaia SourceID
  - 1 single-precision integer (32-bit) – namelen -> Length of name

  - namelen * char (16-bit * namelen) – Characters of the star name, where each character is encoded with UTF-16

The RGB color of stars uses 8 bits per channel in RGBA, and is encoded into a single float using the libgdx Color class.

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

### LOD catalog processing

All LOD catalogs are based on one of the Gaia data releases (DR2, DR3, etc.), and they also include the brighter stars from the Hipparcos catalog. The official Gaia-Hipparcos crossmatch is used to identify stars that are contained in both catalogs. In this case, the parallax is taken from the source that has the smaller parallax error. The rest of de data is taken from the Gaia catalog, but some attributes are merged (for instance, the final star contains both the HIP number and the Gaia source id).

The LOD catalogs are generated using a program written in Rust, called `gaiasky-catgen`. The source code can be found in this repository. In the LOD generation process, each star is processed individually. The catalog is filtered according to the input parameters, and some corrections are applied to star attributes.

### Catalogs

For each Gaia data release, we offer a selection of subsets which contain different cuts of the whole data. These subsets are typically computed using the criterion of parallax relative error, which measures how large the error in parallax is with respect to the parallax value. We define a cut-off value, $s$, which is the maximum percentage of the parallax allowed for the errors, in [0,1]:

$$err_{pllx} < pllx * s$$

where $err_{pllx}$ is the parallax error, and $pllx$ is the parallax for that source. As we mentioned above, $s$ is the cut-off percentage value, in [0,1]. The cut-off value is usually split into two different values, one for **bright** stars and one for **faint** stars. What are bright stars and what are faint stars?

- **Bright** – $G_{mag} < 13.1$
- **Faint** – $G_{mag} >= 13.1$

So, for example, the **DR3 default** catalog contains all stars up to 20%/1.5% parallax relative error for bright/faint stars. This means that all bright stars where the error is not larger than 20% of the parallax are included, and all faint stars where the error is not larger than 1.5% of the parallax are also included.

See all the LOD catalogs we offer in our data server:

- Current catalogs (DR3).

### Distances

In most catalogs, distances are derived from parallaxes, using the formula

$$d[pc] = 1000/pllx[mas].$$

All parallaxes are zero-point corrected as instructed in the official DR documentation before being converted to distances. Sometimes, some parallaxes are negative. In this case, Gaia Sky opts for keeping the star and assigning it a default parallax of 0.04 mas, which corresponds to 25 kpc instead of discarding it.

However, some catalogs use distances determined elsewhere by different methods and injected into the generation process as additional columns. This is the case for the geometric (Bayesian) distances and the photometric distances catalogs.

### Magnitude/color corrections

Extinction and reddening factors are applied to star magnitudes and colors, respectively.

When the extinction value $A_g$ is present in the catalog or in an additional column, it is applied directly to the magnitude. Otherwise, we default to the following analytical extinction,

$$A_g = \min(3.2, \frac{150}{|\sin(b)|} * 5.9e-4),$$

where $b$ is the galactic longitude of the star.

Similarly, we apply the reddening value $E_{BP-RP}$ when it is in the catalog or in an additional column. Otherwise, we fall back to the following analytical determination, based on the extinction:

$$E_{BP-RP} = \min(1.6, A_g * 2.9e-4).$$

## 1.5.9 Particle catalog formats

In order to load simple particles (also referred to as point clouds), Gaia Sky accepts catalogs in common formats like VOTable or CSV (see *the STIL data loader section*), but also in a tailor-made binary format that is fast and compact. This binary format can load simple particles (of type `PARTICLE`, only contain a position) and also extended particles (of type `PARTICLE_EXT`, which contain positions, but also proper motions, colors, magnitudes, etc.). This format is used, for instance, in the most recent versions of the SDSS catalogs.

The class in charge of loading and writing binary particle catalogs is the BinaryPointDataProvider.

The binary format loads much faster than regular VOTable or CSV files, and is described below.

- 1 single-precision integer (32-bit) – number of particles in the file
- 1 byte (8-bit) – boolean (1: true, 0: false) indicating whether to use extended particles or not
- For each particle:
  - 1 double-precision integer (64-bit) – particle identifier
  - 1 single-precision integer (32-bit) – namelen -> Length of name
  - namelen * char (16-bit * namelen) – characters of the particle name. Each character is encoded with UTF-16
  - 1 double-precision float (64-bit) – right ascension [deg]
  - 1 double-precision float (64-bit) – declination in [deg]
  - 1 double-precision float (64-bit) – distance [pc]
  - if extended particles:
    - 1 single-precision float (32-bit) – $\mu_{\alpha\star}$ [mas/yr]
    - 1 single-precision float (32-bit) – $\mu_\delta$ [mas/yr]
    - 1 single-precision float (32-bit) – radial velocity [km/s]
    - 1 single-precision float (32-bit) – apparent magnitude
    - 1 single-precision float (32-bit) – packed color
    - 1 single-precision float (32-bit) – particle pseudo-size

The packed color format uses 8 bits per channel in RGBA, and is encoded into a single-precision floating point number using the libgdx Color class.

### 1.5.10 Archetypes

Below is a table with all the archetypes in Gaia Sky. For each archetype, we list its parent (if any) and its *Components*.

A general description of archetypes and components is provided in *Data morphology*.

All archetypes are: *SceneGraphNode*, *Universe*, *CelestialBody*, *ModelBody*, *Planet*, *Particle*, *Star*, *Satellite*, *HeliotropicSatellite*, *GenericSpacecraft*, *Spacecraft*, *StarCluster*, *Billboard*, *BillboardGalaxy*, *VertsObject*, *Polyline*, *Orbit*, *HeliotropicOrbit*, *FadeNode*, *GenericCatalog*, *MeshObject*, *BackgroundModel*, *SphericalGrid*, *RecursiveGrid*, *BillboardGroup*, *Text2D*, *Axes*, *Loc*, *Area*, *ParticleGroup*, *StarGroup*, *Constellation*, *ConstellationBoundaries*, *CosmicRuler*, *OrbitalElementsGroup*, *Invisible*, *OctreeWrapper*, *Model*, *ShapeObject*, *KeyframesPathObject*, *VRDeviceModel*.

Table 2: Archetypes table

| Archetype | Parent | Components |
|---|---|---|
| **SceneGraphNode** | | *Base*<br>*Body*<br>*GraphNode*<br>Octant<br>Render |
| **Universe** | | *Base*<br>*Body*<br>*GraphNode*<br>GraphRoot |
| **CelestialBody** | *SceneGraphNode* | *Celestial*<br>*Magnitude*<br>*Coordinates*<br>*Orientation*<br>*Label*<br>*SolidAngle*<br>*Focus*<br>Billboard |

continues on next page

Table 2 – continued from previous page

| Archetype | Parent | Components |
|---|---|---|
| **ModelBody** | *CelestialBody* | *Model*<br>*RenderType*<br>*ModelScaffolding*<br>*AffineTransformations* |
| **Planet** | *ModelBody* | *Atmosphere*<br>*Cloud* |
| **Particle** | *CelestialBody* | *ProperMotion*<br>*RenderType*<br>*ParticleExtra* |
| **Star** | *Particle* | Hip<br>Distance<br>*Model*<br>*ModelScaffolding* |
| **Satellite** | *ModelBody* | ParentOrientation |
| **HeliotropicSatellite** | *Satellite* | TagHeliotropic |
| **GenericSpacecraft** | *Satellite* | *RenderFlags* |
| **Spacecraft** | *GenericSpacecraft* | *MotorEngine* |

continues on next page

Table 2 – continued from previous page

| Archetype | Parent | Components |
|---|---|---|
| **StarCluster** | *SceneGraphNode* | *Model*<br>Cluster<br>*SolidAngle*<br>*ProperMotion*<br>*Label*<br>*Focus*<br>Billboard |
| **Billboard** | *ModelBody* | TagBillboardSimple<br>*Fade* |
| **BillboardGalaxy** | *Billboard* | TagBillboardGalaxy |
| **VertsObject** | *SceneGraphNode* | Verts |
| **Polyline** | *VertsObject* | Arrow<br>Line |
| **Orbit** | *Polyline* | *Trajectory*<br>*RefSysTransform*<br>*AffineTransformations*<br>*Label* |
| **HeliotropicOrbit** | *Orbit* | TagHeliotropic |

continues on next page

Table 2 – continued from previous page

| Archetype | Parent | Components |
| --- | --- | --- |
| **FadeNode** | *SceneGraphNode* | *Fade*<br>*Label* |
| **GenericCatalog** | *FadeNode* | *DatasetDescription*<br>Highlight<br>*RefSysTransform*<br>*AffineTransformations* |
| **MeshObject** | *FadeNode* | *Mesh*<br>*Model*<br>*DatasetDescription*<br>*RefSysTransform*<br>*AffineTransformations* |
| **BackgroundModel** | *FadeNode* | TagBackgroundModel<br>*RefSysTransform*<br>*Model*<br>*Label*<br>*Coordinates*<br>*RenderType* |
| **SphericalGrid** | *BackgroundModel* | GridUV |
| **RecursiveGrid** | *SceneGraphNode* | GridRecursive<br>*Fade*<br>*RefSysTransform*<br>*Model*<br>*Label*<br>Line<br>*RenderType* |

continues on next page

Table 2 – continued from previous page

| Archetype | Parent | Components |
|---|---|---|
| **BillboardGroup** | *GenericCatalog* | *BillboardSet*<br>*Coordinates* |
| **Text2D** | *SceneGraphNode* | *Fade*<br>*Title*<br>*Label* |
| **Axes** | *SceneGraphNode* | *Axis*<br>*RefSysTransform*<br>Line |
| **Loc** | *SceneGraphNode* | *LocationMark*<br>*Label* |
| **Area** | *SceneGraphNode* | Perimeter<br>Line<br>TagNoProcessGraph |
| **ParticleGroup** | *GenericCatalog* | *ParticleSet*<br>TagNoProcessChildren<br>*Focus* |
| **StarGroup** | *GenericCatalog* | *StarSet*<br>*Model*<br>*Label*<br>Line<br>*Focus*<br>Billboard |

continues on next page

Table 2 – continued from previous page

| Archetype | Parent | Components |
|---|---|---|
| **Constellation** | *SceneGraphNode* | *Constel*<br>Line<br>*Label*<br>TagNoProcessGraph |
| **ConstellationBoundaries** | *SceneGraphNode* | *Boundaries*<br>Line |
| **CosmicRuler** | *SceneGraphNode* | Ruler<br>Line<br>*Label* |
| **OrbitalElementsGroup** | *GenericCatalog* | OrbitElementsSet<br>TagNoProcessChildren |
| **Invisible** | *CelestialBody* | *Raymarching*<br>TagInvisible |
| **OctreeWrapper** | *SceneGraphNode* | *Fade*<br>*DatasetDescription*<br>Highlight<br>Octree<br>Octant<br>TagNoProcessChildren<br>*AffineTransformations* |

continues on next page

Table 2 – continued from previous page

| Archetype | Parent | Components |
|---|---|---|
| **Model** | *SceneGraphNode* | *Model*<br>*Focus*<br>*RenderType*<br>*Coordinates*<br>*SolidAngle*<br>*RefSysTransform*<br>*AffineTransformations* |
| **ShapeObject** | *Model* | *Shape*<br>*Label*<br>Line |
| **KeyframesPathObject** | *VertsObject* | Keyframes<br>*Label* |
| **VRDeviceModel** | *SceneGraphNode* | VRDevice<br>*Model*<br>Line<br>TagNoClosest |

### 1.5.11 Components

This section lists all components, together with a description and all the attributes. For each attribute, we provide a description and list its units and possible aliases. We also note the *Archetypes* that have the component.

A general description of archetypes and components is provided in *Data morphology*.

All components are: *Base*, *Body*, *GraphNode*, *Coordinates*, *Orientation*, *Celestial*, *Magnitude*, *ProperMotion*, *SolidAngle*, *Shape*, *Trajectory*, *ModelScaffolding*, *Model*, *Atmosphere*, *Cloud*, *RenderFlags*, *MotorEngine*, *RefSysTransform*, *AffineTransformations*, *Fade*, *DatasetDescription*, *Label*, *RenderType*, *BillboardSet*, *Title*, *Axis*, *LocationMark*, *Constel*, *Boundaries*, *ParticleSet*, *StarSet*, *ParticleExtra*, *Mesh*, *Focus*, *Raymarching*.

### Base

Defines basic attributes common to all objects.

This component is in the following archetypes: *SceneGraphNode*, *Universe*.

Table 3: Base attributes

| Attribute | Description | Aliases |
|---|---|---|
| `id` | The ID of the object, typically set automatically by Gaia Sky. | |
| `name` | A single object name, used to identify the object and as a label text, if any. If the object already has names, this attribute overrides the first one in the name list. The first name in the list is also used as the i18n key for translations. | |
| `names` | A list of names, used to identify the object. It overrides the full name list. The first name in the list is used as a label text, and as a i18n key. | |
| `altName` | Adds a new name to the name list of this object, at the end. | `altname` |
| `opacity` | Static opacity value. Typically, this gets overwritten internally in the update process. | |
| `componentType` | The content type string (or list) for this object. Content types control the visibility of objects. Examples of content types are 'Planets', 'Asteroids', 'Stars', 'Labels', etc. | `ct`, `componentTypes` |

### Body

Defines physical body attributes common to all objects.

This component is in the following archetypes: *SceneGraphNode*, *Universe*.

Table 4: Body attributes

| Attribute | Description | Aliases |
|---|---|---|
| `position` | The position of the object. This is the position at epoch if the object has a proper motion, or just a static position. Given in the internal reference system and in internal units by default (see aliases for other units). | `pos`, `positionKm`, `posKm`, `positionPc`, `posPc` |
| `size` | The diameter of the entity (in some archetypes this is the radius). The default attribute uses internal units (see aliases for other units). | `sizeKm`, `sizePc`, `sizepc`, `sizeM`, `sizeAU`, `diameter`, `diameterKm`, `diameterPc` |
| `radius` | The half-size. See size attribute. | `radiusKm`, `radiusPc` |
| `color` | The color of the entity, as a RGBA quartet. Used as the general color of the entity. The last value in the list, alpha, also acts as a transparency value. The color is also applied to the object label unless 'labelColor' is specified. | |
| `labelColor` | The color of the label of this entity. If set, the label of this entity uses this color. Otherwise, it uses the global entity color. | `labelcolor` |

### GraphNode

Defines attributes pertaining to the scene graph hierarchy.

This component is in the following archetypes: *SceneGraphNode*, *Universe*.

Table 5: GraphNode attributes

| Attribute | Description | Aliases |
|---|---|---|
| `parent` | Name of the parent entity in the scene graph. Positions for every object are typically relative to the position of the parent. In some cases, the orientation of the parent is also contemplated. | |

### Coordinates

Defines attributes that provide coordinates and positions to objects.

This component is in the following archetypes: *CelestialBody*, *BackgroundModel*, *BillboardGroup*, *Model*.

Table 6: Coordinates attributes

| Attribute | Description | Aliases |
|---|---|---|
| `coordinatesProvider` | The coordinates provider object for this object. The coordinates provider computes the position of the object for each time. This is an object containing, at least, the full reference to a Java class that implements `IBodyCoordinates` in the "impl" attribute. Examples are `gaiasky.util.coord.StaticCoordinates` or `gaiasky.util.coord.OrbitLintCoordinates`. See *Coordinates and ephemerides* for more information. | `coordinates` |

### Orientation

Defines the orientation model of objects. Can be defined as a rigid rotation (given parameters like rotation period, axial tilt, etc.) or via quaternion-based orientations.

This component is in the following archetypes: *CelestialBody*, *Satellite*.

Table 7: Orientation attributes

| Attribute | Description | Aliases |
|---|---|---|
| `rotation` | The rotation object for this object. This attribute describes a rigid body rotation. This is given in the form of a map with the attributes `angularVelocity`, `period`, `axialtilt`, `inclination`, `ascendingNode` and `meridianAngle`. See *Orientation* for more information. | `rigidRotation` |
| `orientationProvider` | Provider class for the quaternion orientations. | `provider`, `attitudeProvider` |
| `orientationSource` | Location of the data file(s), necessary to initialize the quaternion orientation provider. | `attitudeLocation` |

### Celestial

Defines attributes common to all celestial objects (stars, planets, moons, etc.).

This component is in the following archetypes: *CelestialBody*.

Table 8: Celestial attributes

| Attribute | Description | Aliases |
|---|---|---|
| `wikiName` | The name to look up this object in the wikipedia, if any. If this is set, a '+ info' button appears in the focus info interface when this object is the focus, enabling the user to pull information on the object directly from Gaia Sky and display it in a window. | `wikiname` |
| `colorBV` | The color index B-V of this object. This is only ever used in single particles/stars, and when no 'color' attribute has been specified. If that is the case, we convert the B-V index into an RGB color and use it as the object's global color. | `colorbv`, `colorBv`, `colorIndex` |

### Magnitude

Defines magnitude attributes, both apparent and absolute.

This component is in the following archetypes: *CelestialBody*.

Table 9: Magnitude attributes

| Attribute | Description | Aliases |
|---|---|---|
| `appMag` | The apparent magnitude. If it is not given, it is computed automatically from the absolute magnitude (if present) and the distance. | `appmag`, `apparentMagnitude` |
| `absMag` | The absolute magnitude. If it is not given, it is computed automatically from the apparent magnitude (if present) and the distance. In single stars, the absolute magnitude is used to compute the pseudo-size. See the 'star rendering' section for more information. | `absmag`, `absoluteMagnitude` |

### ProperMotion

Defines proper motion attributes.

This component is in the following archetypes: *Particle*, *StarCluster*.

Table 10: ProperMotion attributes

| Attribute | Description | Aliases |
|---|---|---|
| `muAlpha` | Proper motion in right ascension, the $\mu_{\alpha\star}$, in mas/yr. | `muAlphaMasYr` |
| `muDelta` | Proper motion in declination, the $\mu_{\delta}$, in mas/yr. | `muDeltaMasYr` |
| `radialVelocity` | The radial velocity, in km/s. | `rv`, `rvKms`, `radialVelocityKms` |
| `epochJd` | The epoch as a Julian date. For instance, 2015.5 corresponds to a Julian date of 2457206.125. | |
| `epochYear` | The epoch as a year plus fraction (e.g. 2015.5). This gets converted to a Julian date internally. | |

### SolidAngle

Defines solid angle thresholds for the various rendering modes.

This component is in the following archetypes: *CelestialBody*, *StarCluster*, *Model*.

Table 11: SolidAngle attributes

| Attribute | Description | Aliases |
|---|---|---|
| `thresholdNone` | Solid angle threshold to start rendering this object at all. Mainly for internal use. Gets overwritten during initialization. | |
| `thresholdPoint` | Solid angle threshold boundary between rendering the object as a point and as a quad. Mainly for internal use. Gets overwritten during initialization. | |
| `thresholdQuad` | Solid angle threshold boundary between rendering the object as a quad and as a model. Mainly for internal use. Gets overwritten during initialization. | |

### Shape

Defines attributes related to shape objects

This component is in the following archetypes: *ShapeObject*.

Table 12: Shape attributes

| Attribute | Description | Aliases |
|---|---|---|
| `track` | Shape objects can use the position of other objects as their own. This is useful when, for example, we want to add a wireframe sphere around an object. This attribute contains the name of the object whose position we are to track. | `trackName` |

### Trajectory

Defines attributes related to orbits and trajectory objects. See *Orbits* for more information.

This component is in the following archetypes: *Orbit*.

Table 13: Trajectory attributes

| Attribute | Description | Aliases |
|---|---|---|
| `orbitProvider` | In *Orbit* archetype objects, this is the fully-qualified Java class that provides orbit data. This class needs to implement `IOrbitDataProvider`. **Values:** `gaiasky.data.orbit.OrbitalParametersProvider` – orbit is defined with orbital elements. `gaiasky.data.orbit.OrbitFileDataProvider` – orbit defined from a file of samples. See *Orbits* for more information. | `provider` |
| `orbit` | The orbit component, containing some additional information, like the orbital elements, the period, etc. See *Orbits* for a full description of the format and possible values of this attribute. | |
| `orientationModel` | The orientation model of this orbit. **Values:** `default` – the default orientation. `extrasolar_system` – orientation for extrasolar systems. See *Orbits* for more information. | `model` |
| `onlyBody` | In object-less orbits (orbits not attached to any object), it may be interesting to not render the orbit itself as a line, but only a point at the head of that orbit in the current time. If this attribute is set to true, the orbit is rendered as a single point at the head. Useful essentially to render many particles using orbital elements. **This attribute is deprecated, use bodyRepresentation instead**. | `onlybody` |
| `bodyRepresentation` | The body representation type for this orbit/trajectory. This only works with orbits defined via orbital elements. **Values:** `only_orbit` – the body is not visually represented at all. `only_body` – only the body is visually represented, no line. `body_and_orbit` – both body and orbit line are represented. | |
| `bodyColor` | Body color. Color to use to represent the body in orbital elements trajectories, when the `bodyRepresentation` attribute enables the representation of the body for this trajectory. | `pointColor`, `pointcolor` |
| `pointSize` | The size of the point at the head of the trajectory in object-less orbits (orbits that are not attached to any object). Examples of this are asteroids, where orbits are defined via the orbital elements, and not all orbits are attached to an asteroid object for performance purposes. In these cases, the size of the point at the head of the orbit is set in this property. | `pointsize` |
| `closedLoop` | Define whether the loop must be closed or not (i.e. join the end point with the start point). Defaults to true, which is the value for periodic orbits. | |
| `orbitTrail` | Whether to fade the orbit as it gets further away from the head (or object), in the opposite direction of travel. By default, the head is mapped to an opacity of 1, and the tail is mapped to an opacity of 0. | `orbittrail`, `trail` |
| `trailMap` | Modify the tail mapping value in case `orbitTrail` is set to `true`. This mapping parameter defines the location in the orbit (in [0,1]) where we map the opacity value of 0. The default value is 0. Set it to 0.5 to have a trail from the object to half the orbit. This enables having shorter trails, while improving the performance due to less lines being rendered. | |
| `trailMinOpacity` | Minimum opacity level of the whole orbit in trail mode. Only active if `orbitTrail` is set to `true`. This parameter defines the minimum opacity of the orbit (in [0,1]). This enables having trails where the faint end maps to | |

### ModelScaffolding

Defines attributes related objects with 3D models.

This component is in the following archetypes: *ModelBody*, *Star*.

Table 14: ModelScaffolding attributes

| Attribute | Description | Aliases |
|---|---|---|
| `referencePlane` | The reference plane to use for this model object. **Values:** `ecliptic` `galactic` `equatorial` | `refPlane`, `refplane` |
| `randomize` | A list with the components of this model that need to be randomized via procedural generation. Can contain 'model', 'atmosphere', and/or 'cloud'. | |
| `seed` | In case the 'randomize' attribute is defined, this attribute defines the RNG seed to use. | |
| `sizeScaleFactor` | Scale factor to apply to the 3D model of this object. Mainly used internally. Using the model or object attributes directly to specify the size is recommended. | `sizescalefactor` |
| `locVaMultiplier` | Solid angle multiplier for children location objects (Loc) of this object. If set, this scales the solid angle of the object for children locations. | `locvamultiplier` |
| `locThresholdLabel` | Threshold label value for children locations. Mainly used internally, should not be touched externally. | `locThOverFactor`, `locthoverfactor` |
| `shadowValues` | Override the distance, the camera near and the camera far values of the light camera for this object in the shadow mapping pass. All are given in units of the radius of the object. | `shadowvalues` |

### Model

Defines the actual model of objects with 3D models. See *Model* for more information.

This component is in the following archetypes: *ModelBody*, *Star*, *StarCluster*, *MeshObject*, *BackgroundModel*, *RecursiveGrid*, *StarGroup*, *Model*, *VRDeviceModel*.

Table 15: Model attributes

| Attribute | Description | Aliases |
|---|---|---|
| `model` | Model definition for this object. See the *Model* documentation for more information. | |

### Atmosphere

Defines the atmosphere of a planet or moon. See the *Atmospheric scattering parameters* documentation for more information.

This component is in the following archetypes: *Planet*.

Table 16: Atmosphere attributes

| Attribute | Description | Aliases |
|---|---|---|
| `atmosphere` | Atmosphere definition for this object. See the *Atmospheric scattering parameters* documentation for more information. | |

### Cloud

Defines the cloud layer of a planet or moon. See the *Clouds* documentation for more information.

This component is in the following archetypes: *Planet*.

Table 17: Cloud attributes

| Attribute | Description | Aliases |
|---|---|---|
| `cloud` | Cloud layer definition for this object. See the *Clouds* documentation for more information. | |

### RenderFlags

Defines rendering flags.

This component is in the following archetypes: *GenericSpacecraft*.

Table 18: RenderFlags attributes

| Attribute | Description | Aliases |
|---|---|---|
| `renderQuad` | Whether to render this entity as a billboard (quad). | `renderquad` |

### MotorEngine

Defines machines for the spacecraft mode.

This component is in the following archetypes: *Spacecraft*.

Table 19: MotorEngine attributes

| Attribute | Description | Aliases |
|---|---|---|
| `machines` | Provides machine definitions for the spacecraft mode. Check out the spacecraft object definition in the default data pack for more information. | |

### RefSysTransform

Defines an arbitrary reference system transformation via a 4x4 matrix. See *Reference system transformations* for more information.

This component is in the following archetypes: *Orbit*, *GenericCatalog*, *MeshObject*, *BackgroundModel*, *RecursiveGrid*, *Axes*, *Model*.

Table 20: RefSysTransform attributes

| Attribute | Description | Aliases |
|---|---|---|
| transformFunction | Defines a transformation matrix to apply to the position of the object. The name of the transformation to apply. **Values:** equatorialToEcliptic eclipticToEquatorial equatorialToGalactic galacticToEquatorial eclipticToGalactic galacticToEcliptic See *Reference system transformations* for more information. | transformName |
| transformMatrix | The 16 values of the 4x4 transformation matrix, in column-major order. See *Reference system transformations* for more information. | transformValues |

### AffineTransformations

Defines arbitrary affine transformations, applied in the order they are defined. See *Affine transformations* for more information.

This component is in the following archetypes: *ModelBody*, *Orbit*, *GenericCatalog*, *MeshObject*, *OctreeWrapper*, *Model*.

Table 21: AffineTransformations attributes

| Attribute | Description | Aliases |
|---|---|---|
| matrix | A generic 4x4 matrix transform that will be applied to the sequence of affine transformations. The matrix values need to be in column-major order. See *Affine transformations* for more information. | transformMatrix |
| translate | A translation vector, in internal units (see aliases for other units). See *Affine transformations* for more information. | translatePc, translateKm |
| rotate | A rotation axis and angle, in degrees. The vector is expected as [X, Y, Z, angle]. See *Affine transformations* for more information. | |
| scale | A scale transformation. Can be a 3-vector or a single value. See *Affine transformations* for more information. | |
| transformations | Describe the transformations directly in a map, with 'impl', and whatever attributes. The usage of the attributes 'translate', 'scale' and 'rotate' is strongly recommended over this. | |

### Fade

Defines the properties that control the fading in and out of the object.

This component is in the following archetypes: *Billboard*, *FadeNode*, *RecursiveGrid*, *Text2D*, *OctreeWrapper*.

Table 22: Fade attributes

| Attribute | Description | Aliases |
|---|---|---|
| fadeIn | The starting and ending fade-in distances, in parsecs, from the reference system origin (unless 'fadeObjectName' or 'fadePosition' are defined, in which case the distances are relative to the given object or position), where the object starts and ends its fade-in transition. | fadein |
| fadeInMap | The alpha/opacity values to which the fade-in starting and ending distances are mapped. They default to [0,1]. | |
| fadeOut | The starting and ending fade-out distances, in parsecs, from the reference system origin (unless 'fadeObjectName' or 'fadePosition' are defined, in which case the distances are relative to the given object or position), where the object starts and ends its fade-out transition. | fadeout |
| fadeOutMap | The alpha/opacity values to which the fade-out starting and ending distances are mapped. They default to [1,0]. | |
| fadeObjectName | The name of the object to be used to compute the current distance for the fade in and out operations. | positionobjectname |
| fadePosition | The position, in the internal reference system and internal units, to be used to compute the current distance for the fade in and out operations. | |

### DatasetDescription

Contains metadata about the dataset represented by this object. All objects with this component get an entry in the datasets list.

This component is in the following archetypes: *GenericCatalog*, *MeshObject*, *OctreeWrapper*.

Table 23: DatasetDescription attributes

| Attribute | Description | Aliases |
|---|---|---|
| catalogInfo | A map containing the metadata for the catalog represented by this object. The map can contain the attributes 'name', 'description', 'type' (INTERNAL\|SCRIPT\|LOD\|SAMP\|UI), 'nParticles', 'sizebytes'. See *Catalog formats* for more information. | datasetInfo, cataloginfo |
| addDataset | Whether to add the dataset to the dataset manager or not. Typically used with star and particle sets that already belong to a higher-level dataset. | addToDatasetManager |

### Label

Defines attributes that control how labels are processed and rendered. See *Labels* for more information.

This component is in the following archetypes: *CelestialBody*, *StarCluster*, *Orbit*, *FadeNode*, *BackgroundModel*, *RecursiveGrid*, *Text2D*, *Loc*, *StarGroup*, *Constellation*, *CosmicRuler*, *ShapeObject*, *KeyframesPathObject*.

Table 24: Label attributes

| Attribute | Description | Aliases |
|---|---|---|
| `label` | A boolean to disable or enable label rendering for this object. | |
| `label2d` | Unused, here for backwards compatibility. | |
| `labelPosition` | Override the position at which to render this label, in the internal reference system and internal units (see aliases for more unit options). If this is not given, the position of the object is used. | `labelposition`, `labelPositionPc`, `labelPositionKm` |
| `forceLabel` | Force-display the label of this object, regardless of its solid angle. If 'true', the label for this object is always displayed. | |
| `labelFactor` | Factor to apply to the size of the label for this object. | |
| `labelBias` | Bias to compute the label visibility. >1 to increase visibility. | |
| `labelMax` | Internal rendering factor, should not be set externally. | |
| `textScale` | Internal rendering factor, should not be set externally. | |

### RenderType

Defines attributes that control rendering operations for this object.

This component is in the following archetypes: *ModelBody*, *Particle*, *BackgroundModel*, *RecursiveGrid*, *Model*.

Table 25: RenderType attributes

| Attribute | Description | Aliases |
|---|---|---|
| `renderGroup` | This is an internal property used to fine-tune exactly the environment and shader to use to render the object. See RenderGroup.java for more information. | `rendergroup` |

### BillboardSet

Defines attributes related to billboard set objects.

This component is in the following archetypes: *BillboardGroup*.

Table 26: BillboardSet attributes

| Attribute | Description | Aliases |
|---|---|---|
| `data` | A list of `BillboardDataset` objects. Mainly used for the Milky Way model. Each object contains 'impl', 'file', 'type', 'size', 'intensity', 'layers', and 'maxsizes'. See the Milky Way object in the `universe.json` file in the default data pack for an example. | |

### Title

Defines attributes related to two-dimensional titles.

This component is in the following archetypes: *Text2D*.

Table 27: Title attributes

| Attribute | Description | Aliases |
|---|---|---|
| `scale` | Scale factor to scale up or down the title. | |
| `lines` | Whether to render frame lines above and below the title or not. | |
| `align` | The horizontal alignment of the title. Center (1), left (8) or right (16). | |

### Axis

Defines attributes related to reference system axes.

This component is in the following archetypes: *Axes*.

Table 28: Axis attributes

| Attribute | Description | Aliases |
|---|---|---|
| `axesColors` | A 3x3 matrix with the color for each of the axes in the reference system. | |

### LocationMark

Defines attributes related to locations and location mark objects.

This component is in the following archetypes: *Loc*.

Table 29: LocationMark attributes

| Attribute | Description | Aliases |
|---|---|---|
| `location` | A 2-dimensional position [longitude, latitude] on the surface of the parent, in degrees. | |
| `distFactor` | The distance from the center of the object, in case of non-spherical parent objects. This is essentially the radius, as the locations are given in [longitude, latitude, radius]. Typically the radius is that of the object, but this parameter overrides it. | |

### Constel

Defines attributes related to constellation objects.

This component is in the following archetypes: *Constellation*.

Table 30: Constel attributes

| Attribute | Description | Aliases |
|---|---|---|
| `ids` | Contains a list of segments (a list of lists of points) with the HIP identifiers for each of the stars of this constellation. | |

### Boundaries

Defines attributes related to constellation boundary objects.

This component is in the following archetypes: *ConstellationBoundaries*.

Table 31: Boundaries attributes

| Attribute | Description | Aliases |
|---|---|---|
| `boundaries` | Contains a list of lists of sky coordinates $(\alpha, \delta)$, in degrees, defining the lines of the constellation boundaries. | `boundariesEquatorial` |

### ParticleSet

Defines attributes related to particle set objects, which contain a point cloud.

This component is in the following archetypes: *ParticleGroup*.

Table 32: ParticleSet attributes

| Attribute | Description | Aliases |
|---|---|---|
| `provider` | The class to be used to load the data. This class must implement `IParticleGroupDataProvider`. This should have the fully-qualified class name. For instance, `gaiasky.data.group.STILDataProvider`. | |
| `providerParams` | Parameters to be passed into the provider class. | `providerparams` |
| `meanPosition` | The mean position of this particle set, in the internal reference system and internal units (see aliases for more units). If not given, this is computed automatically from the particle positions. | `meanPositionKm`, `meanPositionPc`, `pos`, `posKm`, `posPc`, `position` |
| `dataFile` | The path to the data file with the particles to be loaded by the provider. | `datafile` |
| `factor` | A multiplicative factor to apply to the positions of all particles during loading. | |
| `numLabels` | Number of labels to render for this particle group. Defaults to the configuration setting. | |
| `profileDecay` | The profile decay of the particles in the shader. Controls how sudden is the color and intensity falloff from the center. | `profiledecay` |
| `colorNoise` | Noise factor for the color, in [0,1]. This randomly generates colors from the main color. The larger the color noise, the more different the generated colors from the main color. | `colornoise` |
| `particleSizeLimits` | Minimum and maximum solid angle limits of the particles in radians. They are used as $(dist * tan(\alpha_{min}), dist*tan(\alpha_{max}))$. The minimum and maximum values must be in [0,1.57]. | `particlesizelimits`, `particleSizeLimitsDeg` |
| `colorMin` | The color of the particles at the closest distance, as RGBA. If this is set, the color of the particles gets interpolated from `colorMin` to `colorMax` depending on the distance of the particles to the origin. | |
| `colorMax` | The color of the particles at the maximum distance, as RGBA. If this is set, the color of the particles gets interpolated from `colorMin` to `colorMax` depending on the distance of the particles to the origin. | |
| `colorFromTexture` | If true, color of this particle depends on the texture assigned to it. This is useful when using 'textureAttribute', for instance, where the texture is assigned depending on the value of an attribute for this object. This feature requires a non-zero 'colorNoise', as it is used to generate the colors. | |
| `fixedAngularSize` | Set a fixed angular size for all particles in this set, as a solid angle in radians (see aliases for other units). | `fixedAngularSizeDeg`, `fixedAngularSizeRad` |
| `renderSetLabel` | Enable or disable the global label of this particle set. If true, the name of this particle set is rendered at the given label position. | |
| `renderParticles` | Disable particle rendering by setting this to false. Labels, in case of star sets, will still be rendered. | |
| `texture` | Texture file to render the particles of this group. This can also point to a directory, in which case all the image files within are used (they must have the same dimensions). If this is provided, `profileDecay` is ignored. | |
| `textures` | List of texture files to render the particles of this group. | |
| | If more than one texture is provided, each particle is assigned a texture randomly. This can also point to one or more directories, in which case all the image files within are used. All images must have the same dimensions. If this is provided, `profileDecay` is ignored. | |

### StarSet

Defines attributes related to star set objects, which contain a star catalog or group.

This component is in the following archetypes: *StarGroup*.

Table 33: StarSet attributes

| Attribute | Description | Aliases |
|---|---|---|
| `provider` | The class to be used to load the data. This class must implement `IParticleGroupDataProvider`. This should have the fully-qualified class name. For instance, `gaiasky.data.group.STILDataProvider`. | |
| `providerParams` | Parameters to be passed into the provider class. | `providerparams` |
| `meanPosition` | The mean position of this particle set, in the internal reference system and internal units (see aliases for more units). If not given, this is computed automatically from the particle positions. | `meanPositionKm`, `meanPositionPc`, `pos`, `posKm`, `posPc`, `position` |
| `dataFile` | The path to the data file with the particles to be loaded by the provider. | `datafile` |
| `factor` | A multiplicative factor to apply to the positions of all particles during loading. | |
| `profileDecay` | The profile decay of the particles in the shader. Controls how sudden is the color and intensity falloff from the center. | `profiledecay` |
| `colorNoise` | Noise factor for the color, in [0,1]. This randomly generates colors from the main color. The larger the color noise, the more different the generated colors from the main color. | `colornoise` |
| `particleSizeLimits` | Minimum and maximum solid angle limits of the particles in radians. They are used as $(dist * tan(\alpha_{min}), dist * tan(\alpha_{max}))$. The minimum and maximum values must be in [0,1.57]. | `particlesizelimits`, `particleSizeLimitsDeg` |
| `colorMin` | The color of the particles at the closest distance, as RGBA. If this is set, the color of the particles gets interpolated from `colorMin` to `colorMax` depending on the distance of the particles to the origin. | |
| `colorMax` | The color of the particles at the maximum distance, as RGBA. If this is set, the color of the particles gets interpolated from `colorMin` to `colorMax` depending on the distance of the particles to the origin. | |
| `fixedAngularSize` | Set a fixed angular size for all particles in this set, as a solid angle in radians (see aliases for other units). | `fixedAngularSizeDeg`, `fixedAngularSizeRad` |
| `renderParticles` | Disable particle rendering by setting this to false. Labels, in case of star sets, will still be rendered. | |
| `epochJd` | The epoch for the positions of this star group as a Julian date. | `epoch` |
| `variabilityEpochJd` | The light curve epoch for the variable stars in this star group as a Julian date. | `variabilityEpoch` |
| `numLabels` | Number of labels to render for this star group. Defaults to the configuration setting. | |

### ParticleExtra

Defines attributes related to single particles and single star objects.

This component is in the following archetypes: *Particle*.

Table 34: ParticleExtra attributes

| Attribute | Description | Aliases |
|---|---|---|
| `primitiveRenderScale` | Artificial scale factor for the size of this particle during rendering. | |
| `tEff` | Effective temperature of the star or body, in Kelvin. | `teff` |

### Mesh

Defines attributes related to meshes and iso-density surfaces. See *Mesh objects* for more information.

This component is in the following archetypes: *MeshObject*.

Table 35: Mesh attributes

| Attribute | Description | Aliases |
|---|---|---|
| `shading` | Shading mode for the mesh. **Values:** `additive` – additive blending. `dust` – opaque mesh with dither transparency at the edges. `regular` – regular general-purpose PBR shader. | |
| `additiveBlending` | Sets the shading mode to 'additive'. | `additiveblending` |

### Focus

Defines attributes related to objects that can be focussed.

This component is in the following archetypes: *CelestialBody*, *StarCluster*, *ParticleGroup*, *StarGroup*, *Model*.

Table 36: Focus attributes

| Attribute | Description | Aliases |
|---|---|---|
| `focusable` | Defines whether the object is focusable or not. By default, this is on. | |

### Raymarching

Defines attributes related to ray-marched objects.

This component is in the following archetypes: *Invisible*.

Table 37: Raymarching attributes

| Attribute | Description | Aliases |
|---|---|---|
| shader | Path to the fragment shader GLSL file to use to render this object. The fragment shader is processed for each pixel in the image, and must produce a ray-marched representation of the object. The file must have one of the following extensions: `.glsl`, `.frag`, `.fragment`, `.glslf`, `.fsh`. The fragment shader file is typically distributed with the dataset, and has the form `$data/[dataset-name]/path/to/file.glsl`. | raymarchingShader |
| additionalTexture | Texture file to pass to the raymarching shader as additional texture. This is usually a noise texture, but can be anything, really. | raymarchingTexture |

## 1.5.12 Star rendering

This section provides a bird's eye view of the star rendering process implemented in Gaia Sky, with pointers to source files implementing the different aspects of it.

The star rendering process in Gaia Sky consists of two parts. First, we compute a pseudo-size for each star, and then we use all the pseudo-sizes in the star shaders to render the stars.

### Pseudo-size determination

We determine the pseudo-size from each star based on the star's apparent magnitude. First, we get the apparent magnitude as seen from the Sun from whatever star catalog. Then, we correct it using extinction data (if available, see *the magnitude/color corrections section*).

The Java code that implements this (`STILDataProvider` class, used to load external catalogs in CSV or VOTable into Gaia Sky) does not include magnitude/color corrections, the Rust code in the catalog generator program does:

- Magnitude corrections - load.rs#L799.

Once we have the corrected apparent magnitude, we convert it to an absolute magnitude with the common formula

$$M = m - 5(log_{10} d_{pc} - 1),$$

where $M$ is the absolute magnitude and $m$ is the apparent one. Finally, we do a conversion from absolute magnitude to *pseudo*-size using the luminosity,

$$L = L_0 * 10^{-0.4 * M_o},$$

where $M_o$ is the bolometric magnitude. Obviously, this is not physically accurate, as the bolometric magnitude should include the contributions of the radiation at all wavelengths, but we found it works quite well in practice for rendering stars. Then we apply a constant factor and a square root, but this is tailored to Gaia Sky's rendering and should probably be adapted to your own renderer. The routine we use is here:

- Magnitude to pseudo-size routine - load.rs#L826.

Or in Java, look at method `absoluteMagnitudeToPseudoSize(double)` of `STILDataProvider` here.

### Star shader and rendering

That is only half of the picture though. That gets us the pseudo-size from the apparent magnitude.

On the rendering side, Gaia Sky has the option of rendering stars using billboards (quads implemented as two triangles sharing two vertices) or using native driver points (`GL_POINTS`). The code using `GL_POINTS` is faster but has some important drawbacks like points being drawn in screen space, which ignores effects like perspective distortion, so we focus here on the billboard quads.

We pass the pseudo-size $p$ into the shader and compute the solid angle $\alpha$ from this size and the current distance $d$ from the camera to the star:

$$\alpha = \text{atan}(p/d)$$

Since we are dealing with distant stars most of the time, we can probably get away with using the small-angle formula (i.e. omit atan). We found it does not have much of an impact on performance on relatively modern GPUs, and it gives us accurate angles when stars get closer. Then, we use the solid angle, together with the pseudo-size and some additional parameters (like the brightness power, which applies a power function to the solid angle to artificially widen the difference between bright and faint stars) to work out the quad size. This is implemented in various shaders. For example, have a look at:

- Star shader code - star.group.quad.vertex.glsl#L83

The whole star rendering process involves many parameters (i.e. see the *visual settings section*) and is quite complex, but with the topics discussed in this section you should have a solid understanding of what is going on behind the scenes.

## 1.5.13 Defining an extrasolar system

In this little example we will define a made-up extrasolar system with two stars orbiting the common barycenter and four planets doing the same. We call the stars *Exonia A* and *Exonia B*, and the planets *Exonia c*, *d*, *e* and *f*. Additionally, we'll give *Exonia c* a small moon called *Exonia c1*. The system we'll create in this short tutorial can be downloaded directly from the download manager in Gaia Sky (with Gaia Sky 3.2.0+) or manually here.

**Contents**


- *Defining an extrasolar system*
  - *Initial set up*
  - *Defining the objects*
    - *Stars*
    - *Planets*


### Initial set up

First, we need to create the JSON file where we define the objects. Go to the `$data` directory (see *System Directories*) and create the directory and file `system-exonia/dataset.json`:

```
cd $data
mkdir system-exonia && cd system-exonia
echo "{'objects':[]}" > dataset.json
```

We have created a file with no objects. Since we have added the `dataset.json` file, Gaia Sky will recognize the dataset (now it is still empty, but we'll get to that), and you'll be able to select it in the dataset manager at startup.

The dataset descriptor file contains some metadata about the new catalog. Let's have a look:

```
{
    "key" : "system-exonia",
    "name" : "Exonia extrasolar system",
    "type" : "system",
    "version" : 2,
    "description" : "A made-up, partially procedurally
                    generated extra-solar system with
                    two stars, three planets and a moon.",
    "size" : 2600000,
    "nobjects" : 7,
    "data" : [
        {
            "loader": "gaiasky.data.JsonLoader",
            "files": [ "$data/system-exonia/system-exonia.json" ]
        }
    ]
}
```

As we can see, it has the catalog name, its version, its type, the description, the size, the number of objects and a pointer to the actual data file. All JSON files in Gaia Sky are loaded with the `JsonLoader` class, so that part is more or less constant.

### Defining the objects

Now we are ready to start adding our objects. First, since all the objects orbit the barycenter of the system, we need an object to represent its position. This object must not be visible, but it must allow us to represent a position in space, and it must be the parent of all other objects in the system. We can do that with an object of the archetype `Invisible`.

```
{
    "objects" : [
        {
            "names" : ["Exonia Center"],
            "componentType" : ["Others"],
            "size" : 6.0e4,

            "parent" : "Universe",
            "archetype" : "Invisible",

            "coordinates" : {
                "impl" : "gaiasky.util.coord.StaticCoordinates",
                "positionPc" : [20.0, 90.0, 10.0]
            }
        }
    ]
}
```

This object has the name *Exonia Center*. We will use this name in the `"parent"` attribute when we crate the rest of the objects. Since the barycenter of this system will not move, we use `StaticCoordinates`. Here he have used the property "positionPc", so we need to enter the position in parsecs in the *internal reference system*. We could have used "positionKm" to use kilometers, or simply "position" to use *internal units*.

The internal reference system is based on the equatorial system, so we need equatorial Carteisan coordinates. However, we can also use ecliptic or galactic cartesian coordinates by specifying a transformation:

```
{
    "coordinates" : {
       "impl" : "gaiasky.util.coord.StaticCoordinates",
       "positionPc" : [20.0, 90.0, 10.0],
       "transformName" : "galacticToEquatorial"
    }
}
```

Here, the properties "transformName" and "transformFunction" can be used interchangeably to specify the transformation.

Finally, we also accept spherical equatorial, galactic and ecliptic coordinates. They are set using "positionEquatorial" (alpha, delta, distance), "positionEcliptic" (l, b, distance) and "positionGalactic" (gal_lon, gal_lat, distance). The angles are in degrees and the distances are in parsecs. If you choose to use spherical coordinates, please do not specify a transformation!

### Stars

Catalog stars in Gaia Sky are not single objects, so we can't just use them. If we need a star to do fancy things (e.g., move around in an orbit, have children exoplanets, etc.), we need to define a model object using the archetype Star. In our case, for each star we need an *orbit* object (as we want the star to move in an elliptical orbit) and a *star* object. The orbit is in charge of constructing the trajectory data and rendering it, while the star is the model object representing the star itself. We have two stars in the system, *Exonia A* and *Exonia B*. More information on the orbit format can be found *here*. Let's see how we define them and then we go over the attributes:

```
{
    "name" : "Exonia A orbit",
    "color" : [1.0, 0.0, 1.0, 0.8],
    "componentType": [ "Orbits", "Stars" ],

    "parent" : "Exonia Center",
    "archetype" : "Orbit",
    "provider" : "gaiasky.data.orbit.OrbitalParametersProvider",

    "transformFunction" : "galacticToEquatorial",
    "newmethod": true,
    "orbit"                 : {
        "period" : 130.7655287755297,
        "epoch" : 2455198.0,
        "semimajoraxis" : 77112085.246326,
        "eccentricity" : 0.28862,
        "inclination" : 7.134,
        "ascendingnode" : 3.91,
        "argofpericenter" : 0.84,
        "meananomaly" : 307.80
    }
},
{
```

(continues on next page)

(continued from previous page)

```
    "names" : ["Exonia A"],
    "color" : [1.0, 0.9213, 0.8818, 1.0],
    "colorbv" : 0.656,
    "componentType" : "Stars",

    "absmag" : 2.85,
    "appmag" : 8.73,

    "parent" : "Exonia Center",
    "archetype" : "Star",

    "coordinates" : {
        "impl" : "gaiasky.util.coord.OrbitLintCoordinates",
        "orbitname" : "Exonia A orbit"
    }
},
{
    "name" : "Exonia B orbit",
    "color" : [0.0, 1.0, 1.0, 0.8],
    "componentType": [ "Orbits", "Stars" ],

    "parent" : "Exonia Center",
    "archetype" : "Orbit",
    "provider" : "gaiasky.data.orbit.OrbitalParametersProvider",

    "transformFunction" : "galacticToEquatorial",
    "newmethod" : true,
    "orbit"                  : {
        "period" : 120.7655287755297,
        "epoch" : 2455198.0,
        "semimajoraxis" : 71112085.246326,
        "eccentricity" : 0.4,
        "inclination" : 0.93415027853740,
        "ascendingnode" : 130.74959784132,
        "argofpericenter" : 140.31984971,
        "meananomaly" : 0.40983227102735
    }
},
{
    "names" : ["Exonia B"],
    "color" : [0.9, 0.8213, 0.7818, 1.0],
    "colorbv" : 0.656,
    "componentType" : "Stars",

    "absmag" : 4.85,
    "appmag" : 10.73,

    "parent" : "Exonia Center",
    "archetype" : "Star",

    "coordinates" : {
        "impl" : "gaiasky.util.coord.OrbitLintCoordinates",
```

(continues on next page)

(continued from previous page)

```
        "orbitname" : "Exonia B orbit"
    }
}
```

Note that we have two orbits and two stars. Orbits are objects of archetype `Orbit`. Each orbit is essentially a vessel for the orbital elements (period, semi-major axis, eccentricity, etc.). More information on the format and units can be found *here*. Since our *Internal reference system* is an equatorial system, we add the `"galacticToEquatorial"` transformation so that our reference plane is actually the galactic plane instead of Earth's equatorial plane.

Single stars are objects of archetype `Star`. Note that the parent of all objects is the invisible object *Exonia Center*. Additionally, the coordinates of the stars are provided by the respective orbit objects in the "coordinates" key. The rest is adding the star parameters like color, magnitudes, etc.

Colors can be specified in RGB or using the B-V color index. If both are specified, RGB takes precedence. In case Gaia Sky only finds the B-V color index, it translates it to RGB using this procedure.

Star absolute magnitudes and sizes, `"absmag"` and `"size"`, are intertwined. If only the absolute magnitude is specified, it is converted to a radius. The conversion is callibrated with the Sun, so that an absolute magnitude of ~4.85 produces roughly the radius of the Sun, ~3.5e4 km. Otherwise, the size can be specified directly. Since single stars and star catalogs use different render paths, the relatve brightness of single stars with respect to stars in catalogs may not be entirely accurate.

### Planets

Now, let's have a look at the planets. They are very similar to the stars in that they also need orbit objects. They differ in the object type and attributes tough. *Exonia e* uses JPG images as textures, so at this point you should get the Exonia data pack, which contains these textures. The rest of the objects use *procedurally generated* data.

Here is the definition of the planets:

```
{
    "name" : "Exonia c orbit",
    "color" : [0.3, 0.2, 0.9, 0.7],
    "componentType": [ "Orbits", "Planets" ],

    "parent" : "Exonia Center",
    "archetype" : "Orbit",
    "provider" : "gaiasky.data.orbit.OrbitalParametersProvider",

    "transformFunction" : "galacticToEquatorial",
    "newmethod": true,
    "orbit" : {
        "period" : 1325.85,
        "epoch" : 2455400.5,
        "semimajoraxis" : 353350171.0,
        "eccentricity" : 0.08862,
        "inclination" : 7.134,
        "ascendingnode" : 103.91,
        "argofpericenter" : 149.84,
        "meananomaly" : 307.80
    }
},
{
```

(continues on next page)

(continued from previous page)

```
    "name" : "Exonia c",
    "color" : [0.5, 0.6, 1.0, 1.0],
    "size" : 2410.3,
    "componentType" : "Planets",

    "absmag" : 0.5,

    "parent" : "Exonia Center",
    "archetype" : "Planet",

    "coordinates" : {
        "impl" : "gaiasky.util.coord.OrbitLintCoordinates",
        "orbitname" : "Exonia c orbit"
    },

    "rotation"              : {
        "period" : 400.536,
        "axialtilt" : 0.0,
        "inclination" : 0.281,
        "meridianangle" : 200.39
    },
    "model"                 : {
        "args" : [true],
        "type" : "sphere",
        "params" : {
            "quality" : 400,
            "diameter" : 1.0,
            "flip" : false
        },
        "material" : {
            "height" : "generate",
            "diffuse" : "generate",
            "normal" : "generate",
            "specular" : "generate",
            "biomelut" : "data/tex/base/biome-smooth-lut.png",
            "biomehueshift" : 80.0,
            "noise" : {
                "seed" : 5229243,
                "scale" : 0.2,
                "type" : "simplex",
                "fractaltype" : "ridgemulti",
                "frequency" : 4.34,
                "octaves" : 10,
                "range" : [-1.8, 1.0],
                "power" : 4.0
            },
            "heightScale" : 3.0
        }
    },
    "cloud"         : {
        "size" : 2430.0,
        "cloud" : "generate",
```

(continues on next page)

(continued from previous page)

```
        "noise" : {
            "seed" : 1234,
            "scale" : [1.0, 1.0, 0.4],
            "type" : "simplex",
            "fractaltype" : "ridgemulti",
            "frequency" : 4.34,
            "octaves" : 6,
            "range" : [-1.5, 0.4],
            "power" : 10.0
        },

        "params" : {
            "quality" : 200,
            "diameter" : 2.0,
            "flip" : false
        }
    },
    "atmosphere"    : {
        "size" : 2580.0,
        "wavelengths" : [0.7, 0.8, 0.9],
        "m_Kr" : 0.0025,
        "m_Km" : 0.0015,
        "m_eSun" : 1.0,
        "fogdensity" : 2.5,
        "fogcolor" : [1.0, 0.7, 0.6],

        "params" : {
            "quality" : 600,
            "diameter" : 2.0,
            "flip" : true
        }
    }
},
{
    "name" : "Exonia c1 orbit",
    "color" : [0.8, 0.4, 0.4, 0.7],
    "componentType": [ "Orbits", "Moons" ],

    "parent" : "Exonia c",
    "archetype" : "Orbit",
    "provider" : "gaiasky.data.orbit.OrbitalParametersProvider",

    "newmethod": true,
    "orbit"                 : {
        "mu" : 4.2e13,
        "period" : 1.2624407,
        "epoch" : 2455198.0,
        "semimajoraxis" : 23463.2,
        "eccentricity" : 0.00033,
        "inclination" : 1.791,
        "ascendingnode" : 0.370,
        "argofpericenter" : 0.233,
```

(continues on next page)

(continued from previous page)

```
        "meananomaly" : 0.554
    }
},
{
    "name" : "Exonia c1",
    "color" : [0.5, 0.6, 1.0, 1.0],
    "size" : 410.3,
    "componentType" : "Moons",

    "absmag" : 0.5,

    "parent" : "Exonia c",
    "archetype" : "Planet",

    "coordinates" : {
        "impl" : "gaiasky.util.coord.OrbitLintCoordinates",
        "orbitname" : "Exonia c1 orbit"
    },

    "rotation"              : {
        "period" : 40.536,
        "axialtilt" : 0.0,
        "inclination" : 0.281,
        "meridianangle" : 200.39
    },
    "model"                 : {
        "args" : [true],
        "type" : "sphere",
        "params" : {
            "quality" : 400,
            "diameter" : 1.0,
            "flip" : false
        },
        "material" : {
            "height" : "generate",
            "diffuse" : "generate",
            "normal" : "generate",
            "specular" : "generate",
            "biomelut" : "data/tex/base/rock-smooth-lut.png",
            "biomehueshift" : 289.0,
            "noise" : {
                "seed" : 963249243,
                "scale" : 0.12,
                "type" : "simplex",
                "fractaltype" : "ridgemulti",
                "frequency" : 3.0,
                "octaves" : 8,
                "range" : [0.0, 1.0],
                "power" : 1.0
            },
            "heightScale" : 20.0
        }
```

(continues on next page)

(continued from previous page)

```
    }
},
{
    "name" : "Exonia d orbit",
    "color" : [1.0, 0.7, 0.5, 0.7],
    "componentType": [ "Orbits", "Planets" ],

    "parent" : "Exonia Center",
    "archetype" : "Orbit",
    "provider" : "gaiasky.data.orbit.OrbitalParametersProvider",

    "transformFunction" : "galacticToEquatorial",
    "newmethod" : true,
    "orbit"                 : {
        "period" : 3851.7655287755297,
        "epoch" : 2455198.0,
        "semimajoraxis" : 719622085.246326,
        "eccentricity" : 0.1,
        "inclination" : 0.93415027853740,
        "ascendingnode" : 130.74959784132,
        "argofpericenter" : 180.31984971,
        "meananomaly" : 0.40983227102735
    }

},
{
    "name" : "Exonia d",
    "color" : [0.71, 0.32, 0.08, 1.0],
    "size" : 7439.7,
    "componentType" : "Planets",

    "absmag" : -2.67,
    "appmag" : 5.73,

    "parent" : "Exonia Center",
    "archetype" : "Planet",
    "refplane" : "equatorial",

    "coordinates" : {
        "impl" : "gaiasky.util.coord.OrbitLintCoordinates",
        "orbitname" : "Exonia d orbit"
    },

    "rotation"              : {
        "period" : 1407.509405,
        "axialtilt" : 2.1833,
        "inclination" : 7.005,
        "meridianangle" : 329.548
    },

    "model"                 : {
        "args" : [true],
```

(continues on next page)

(continued from previous page)

```
        "type" : "sphere",
        "params" : {
            "quality" : 400,
            "diameter" : 1.0,
            "flip" : false
        },
        "material" : {
            "height" : "generate",
            "diffuse" : "generate",
            "normal" : "generate",
            "specular" : "generate",
            "biomelut" : "data/tex/base/biome-smooth-lut.png",
            "biomehueshift" : -15.0,
            "noise" : {
                "seed" : 993390,
                "scale" : 0.1,
                "type" : "simplex",
                "fractaltype" : "ridgemulti",
                "frequency" : 5.34,
                "octaves" : 10,
                "range" : [-1.4, 1.0],
                "power" : 7.5
            },
            "heightScale" : 14.0
        }
    },
    "cloud"          : {
        "size" : 7475.0,
        "cloud" : "generate",
        "noise" : {
            "seed" : 1983,
            "scale": [1.8, 1.8, 1.0],
            "type" : "simplex",
            "fractaltype" : "ridgemulti",
            "frequency" : 2.34,
            "octaves" : 4,
            "range" : [-1.5, 0.8],
            "power" : 7.0
        },

        "params" : {
            "quality" : 200,
            "diameter" : 2.0,
            "flip" : false
        }
    },
    "atmosphere"     : {
        "size" : 7730.0,
        "wavelengths" : [0.6, 0.56, 0.475],
        "m_Kr" : 0.0025,
        "m_Km" : 0.0015,
        "m_eSun" : 3.0,
```

(continues on next page)

(continued from previous page)

```
        "fogdensity" : 4.5,
        "fogcolor" : [0.8, 0.9, 1.0],

        "params" : {
            "quality" : 600,
            "diameter" : 2.0,
            "flip" : true
        }
    }
},
{
    "name" : "Exonia e orbit",
    "color" : [0.2, 1.0, 0.5, 0.7],
    "componentType": [ "Orbits", "Planets" ],

    "parent" : "Exonia Center",
    "archetype" : "Orbit",
    "provider" : "gaiasky.data.orbit.OrbitalParametersProvider",

    "transformFunction" : "galacticToEquatorial",
    "newmethod": true,
    "orbit" : {
        "period" : 9905.85,
        "epoch" : 2455400.5,
        "semimajoraxis" : 1353350171.0,
        "eccentricity" : 0.08862,
        "inclination" : 9.134,
        "ascendingnode" : 83.91,
        "argofpericenter" : 149.84,
        "meananomaly" : 209.80
    }
},
{
    "name" : "Exonia e",
    "color" : [0.71, 0.32, 0.08, 1.0],
    "size" : 3389.7,
    "componentType" : "Planets",

    "absmag" : -2.67,
    "appmag" : 5.73,

    "parent" : "Exonia Center",
    "archetype" : "Planet",
    "refplane" : "equatorial",

    "coordinates" : {
        "impl" : "gaiasky.util.coord.OrbitLintCoordinates",
        "orbitname" : "Exonia e orbit"
    },

    "rotation"              : {
        "period" : 1407.509405,
```

(continues on next page)

(continued from previous page)

```
        "axialtilt" : 2.1833,
        "inclination" : 7.005,
        "meridianangle" : 329.548
    },

    "model"                 : {
        "args" : [true],
        "type" : "sphere",
        "params" : {
            "quality" : 400,
            "diameter" : 1.0,
            "flip" : false
        },
        "material" : {
            "diffuse" : "data/tex/base/exoniae-diffuse.jpg",
            "emissive" : "data/tex/base/exoniae-emissive.jpg",
            "normal" : "data/tex/base/exoniae-normal.jpg",
            "metallic" : "data/tex/base/exoniae-metallic.jpg",
            "height" : "data/tex/base/exoniae-height.jpg",
            "heightScale" : 25.9848
        }
    },
    "atmosphere"    : {
        "size" : 3500.0,
        "wavelengths" : [0.55, 0.5, 0.45],
        "m_Kr" : 0.0025,
        "m_Km" : 0.0015,
        "m_eSun" : 3.0,
        "fogdensity" : 4.5,
        "fogcolor" : [0.8, 0.9, 1.0],

        "params" : {
            "quality" : 600,
            "diameter" : 2.0,
            "flip" : true
        }
    }
},
{
    "name" : "Exonia f orbit",
    "color" : [1.0, 1.0, 0.4, 0.7],
    "componentType": [ "Orbits", "Planets" ],

    "parent" : "Exonia Center",
    "archetype" : "Orbit",
    "provider" : "gaiasky.data.orbit.OrbitalParametersProvider",

    "transformFunction" : "galacticToEquatorial",
    "newmethod": true,
    "orbit" : {
        "period" : 11095.85,
        "epoch" : 2455400.5,
```

(continues on next page)

(continued from previous page)

```
        "semimajoraxis" : 1453350171.0,
        "eccentricity" : 0.08862,
        "inclination" : 9.134,
        "ascendingnode" : 83.91,
        "argofpericenter" : 149.84,
        "meananomaly" : 209.80
    }
},
{
    "name" : "Exonia f",
    "color" : [0.71, 0.72, 0.78, 1.0],
    "size" : 3389.7,
    "componentType" : "Planets",

    "absmag" : -2.67,
    "appmag" : 5.73,

    "parent" : "Exonia Center",
    "archetype" : "Planet",
    "refplane" : "equatorial",

    "coordinates" : {
        "impl" : "gaiasky.util.coord.OrbitLintCoordinates",
        "orbitname" : "Exonia f orbit"
    },

    "rotation"              : {
        "period" : 1407.509405,
        "axialtilt" : 2.1833,
        "inclination" : 7.005,
        "meridianangle" : 329.548
    },
    "seed" : [9858457687, 11448],
    "randomize" : [ "model", "atmosphere" ],
    "cloud"         : {
        "size" : 3400.0,
        "cloud" : "generate",
        "noise" : {
            "seed" : 1234,
            "scale": [0.05, 0.05, 0.4],
            "type" : "gradval",
            "fractaltype" : "decarpenterswiss",
            "frequency" : 5.34,
            "octaves" : 9,
            "range" : [-0.1, 1.0],
            "power" : 2.0
        },

        "params" : {
            "quality" : 200,
            "diameter" : 2.0,
            "flip" : false
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

The orbits are essentially the same as in the case of stars. The objects are now using the archetype `Planet`, and they define a rotation and a model. The rotation specifies the parameters of the orientation and rotation of the planet like the period, the axial tilt and the inclination. The model defines the 3D model object properties. In this case we use spheres. Within the model is the material, which defines the textures to use for each of the material attributes and optionally the procedural generation parameters. We also have clouds and atmospheres, but these are covered in the *procedural generation section*.

Here is a short clip of the system once loaded into Gaia Sky:

### 1.5.14 Cubemaps

Gaia Sky supports cubemaps, in addition to regular equirectangular (spherically projected) images, to texture planets, moons and other spherical or semi-spherical objects.

The use of cubemaps instead of plain textures helps eliminate the artifacts happening at the poles with UV sphere models. Other possible solutions are using icospheres or octahedronsphers, but in these seams may appear due to the uneven texture coordinates. The image below illustrates this iusse.

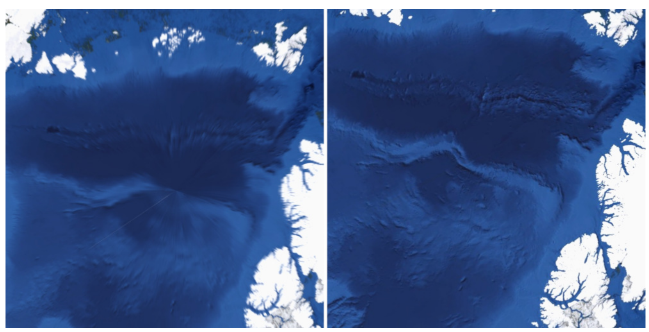


Fig. 86: Detail of the north pole region of the Earth, using a regular texture (left) and using a cubemap (right). Note the artifacts on the left image.

Cubemaps are supported for the diffuse, specular, normal, emissive, metallic, roughness and height channels. All these can be applied to regular models. Additionally, the diffuse cubemap can be applied to the cloud layer. The keys are the following:

- `"diffuseCubemap"`
- `"specularCubemap"`
- `"normalCubemap"`
- `"emissiveCubemap"`
- `"metallicCubemap"`
- `"roughnessCubemap"`
- `"heightCubemap"`

Cubemaps are composed of six individual textures for the up, down, right, left, front and back directions. They can be easily generated from equirectangular images by using this python converter. To generate a cubemap from an equirectangular texture, clone the repository and use the *equitocubemap* script:

```
equitocubemap IMAGE CUBEMAP_SIDE_RES OUTPUT_LOCATION
```

The six cubemap image will be saved in `OUTPUT_LOCATION` with the prefixes `_ft.jpg`, `_bk.jpg`, `_up.jpg`, `_dn.jpg`, `_rt.jpg` and `_lf.jpg`. PNG is also supported.

In Gaia Sky, you need to point the `"diffuseCubemap"` property to the location of these six cubemap sides. Gaia Sky will take the appropriate image for each cubemap side using the file name suffixes. The following suffixes are recognized by Gaia Sky:

Table 38: File name suffixes for each cubemap side.

| Side | Suffixes |
|---|---|
| back | `bk`, `back`, `b` |
| front | `ft`, `front`, `f` |
| up | `up`, `top`, `u`, `t` |
| down | `dn`, `bottom`, `d` |
| right | `rt`, `right`, `r` |
| left | `lf`, `left`, `l` |

### 1.5.15 Virtual Textures

Gaia Sky supports Sparse Virtual Textures (SVT), which enable ultra-high resolution partially resident textures to be used to map planets and other objects. From the user's perspective, virtual textures are *transparent*, meaning that the user does not even need to be aware they are being used.

**Contents**


- *Virtual Textures*
  - *Overview*
  - *Creating Virtual Texture Datasets*
  - *Preparing the tiles*
  - *Tools*
  - *Limitations*


**Hint:** The implementation of Sparse Virtual Textures in Gaia Sky is thoroughly explained in this external article.

#### Overview

**Virtual Textures** (VT), also known as **Sparse Virtual Textures** (SVT), **MegaTextures**, and **Partially Resident Textures** (PRT), have at their core the idea of splitting large textures into several tiles and only streaming the necessary ones (i.e. the ones required to render the current view) to graphics memory in order to optimize memory usage and enable the display of textures so large that they can't be handled effectively by the graphics hardware. In this article we use VT and SVT interchangeably to refer to virtual textures.

This technique aims at drastically increasing the size of usable textures in real time rendering applications by splitting them up in tiles and streaming only the necessary ones to graphics memory. It was initially described in a primitive form by Chris Hall in 19993 and has subsequently been improved upon. My understanding is that most modern implementations are based on Sean Barret's GDC 2008 talk on the topic.

Committed texture pages are kept in a texture, called cache, which is unique for all virtual textures. The size of the cache (in tiles) can be adjusted in the Graphics Settings, *virtual textures section*.

### Creating Virtual Texture Datasets

An SVT is essentially a quadtree which contains a downsized version of the whole texture in the root node. Each level contains 4 times the amount of tiles of the level above, and each tile covers 4 times less area. The pixel count and resolution of all tiles in all levels is always the same.

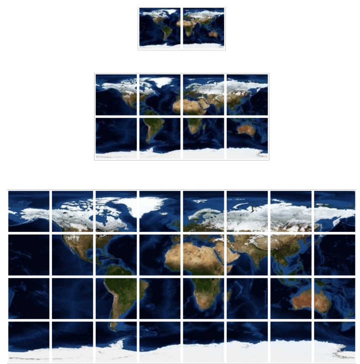


Fig. 87: An example of a virtual texture with 3 levels (0 to 2) for the Earth laid out as a quadtree. Note that the root (level 0, top), covers the whole area, while successive levels have equally-sized tiles that cover less and less area each. This VT has an aspect ratio of 2:1, so it has two root nodes at the top.

In Gaia Sky, SVTs can be packed into a dataset. To do so, we create a new directory for the dataset, preferably using the naming convention *vt-[object]-[channel]-[source]*. For example, *vt-earth-diffuse-nasa* is a good name for a VT for the Earth's surface generated from a NASA dataset. Virtual Textures, like regular textures and cubemaps, can be applied to several material properties:

- Diffuse – the color of the surface of a planet or moon, for shading.
- Specular – the specular map, for shading.
- Normal – the normal map, for shading.
- Height – the elevation map, to be used by the tessellation shader or by the parallax mapping process, depending on the height representation chosen.
- Metallic – the metallic map, for PBR shading.
- Roughness – the roughness map, for PBR shading.
- Clouds – the cloud layer.

Typically, we create a virtual texture dataset for a pre-existing object, like the Earth, the Moon or Mars. The Gaia Sky JSON format incorporates some syntax to update already loaded objects. For instance, we can add a diffuse virtual texture to the Earth with the following JSON descriptor in the file `vt-earth-diffuse-nasa.json`:

```
{"updates" : [
  {
    "name" : "Earth",
    "model": {
      "material" : {
```

(continues on next page)

(continued from previous page)

```
        "diffuseSVT" : {
          "location" : "$data/virtualtex-earth-diffuse/tex",
          "tileSize" : 1024
        }
      }
    }
  }
]}
```

The "updates" object name at the top marks the objects in the list as *updates*. Then, we define the name of the object and the properties we need to update, with the same structure as in the original description file. For instance, if `diffuseSVT` is a property of `material`, which is inside `model`, the same structure must be maintained in the update file.

The following are the objects and attributes that can be updated:

- `material` – material and all its sub-attributes. In particular all, regular textures, cubemaps and virtual textures: - `diffuse`, `diffuseCubemap`, `diffuseSVT`. - `specular`, `specularCubemap`, `specularSVT`. - `normal`, `normalCubemap`, `normalSVT`. - `height`, `heightCubemap`, `heightSVT`. - `emissive`, `emissiveCubemap`, `emissiveSVT`. - `metallic`, `metallicCubemap`, `metallicSVT`. - `roughness`, `roughnessCubemap`, `roughnessSVT`.
- `cloud` – describes the cloud layer. Can also have a virtual texture. - `diffuse`, `diffuseCubemap`, `diffuseSVT`.
- `atmosphere` – all its direct attributes.
- `rotation` – all its direct attributes.

Any SVT needs to specify a `location` and a `tileSize`. The location is the directory where the tiles for the different levels are located. The tile size is just the resolution of the tiles of this SVT.

Gaia Sky can work with multiple SVTs, but they all need to have the same tile size. Additionally, the tile size needs to be a power of two in [4, 1024].

### Preparing the tiles

The dataset directory must contain a dataset descriptor file named `dataset.json`, and the actual data descriptor seen in the previous section (`vt-earth-diffuse-nasa.json`).

A VT dataset directory looks like this:

```
tex/
dataset.json
vt-earth-diffuse-nasa.json
```

The tiles are located in the `tex` directory within the dataset directory. Tile files are separated by levels using directories. Every level has the name `level[level]`. For example, the tiles for level 3 are all inside the `tex/level3` directory.

The `tex/` directory looks like the following, for a dataset with 7 levels, from 0 to 6:

```
level0/
level1/
level2/
level3/
level4/
```

(continues on next page)

(continued from previous page)

```
level5/
level6/
```

Each level directory contains the tiles for that level. The first level contains either or two tiles (depending on the aspect ratio of the virtual texture), the second level contains 4 times that number, and so on (each tile is subdivided into 4 sub-tiles in the next level). Tile files are named `tx_[col]_[row].ext`, where `col` is the column, and `row` is the row. Supported formats are JPG and PNG.

The `level1/` directory looks like this:

```
tx_0_0.jpg
tx_0_1.jpg
tx_1_0.jpg
tx_1_1.jpg
tx_2_0.jpg
tx_2_1.jpg
tx_3_0.jpg
tx_3_1.jpg
```

When in doubt, look at the existing VT datasets.

It is important to know that **levels (except level 0) do not need to be complete**. Missing tiles will be queried at higher levels automatically.

### Tools

We did not find any open-source tools to our liking to create virtual texture tiles from high-resolution texture data, so we created our own. You can find them in the virtual texture tools repository. This repository contains two scripts:

- `split-tiles` — can split a texture into square tiles of a given resolution, and names the tiles in the format expected by Gaia Sky (and also Celestia), which is `tx_[col]_[row].ext`. The output format, quality and starting column and row are configurable via arguments.
- `generate-lod` — given a bunch of tiles and a level number, this script generates all the upper levels by stitching and resizing tiles. It lays them out in directories with the format `levelN`, where N is the zer-based level. The input tiles are also expected in a directory. The output format and quality are configurable.

### Limitations

The limitations of our implementation are the following:

- Due to the fact that all SVTs in the scene share the same cache, right now we can't have SVTs with different tile sizes in the same scene.
- Similarly, only square tiles are supported. Actually, I can't think of a single good use case for non-square tiles.
- Supported virtual texture aspect ratios are n:1, with $n \geq 1$. This is due to the fact that VT quadtrees are square by definition (1:1), and we have an array of root quadtree nodes that stack horizontally in the tree object. It is currently not possible to have a VT with a greater height than width.
- Performance is not very good, especially with many SVTs running at once. This may be due to the shader mimpmap level lookups. This produces $depth$ texture lookups (mip levels) in the worst-case scenario when only the root node is available in the cache. A workaround would be to fill lower levels, additionally to the tile level, in the indirection buffer whenever a tile enters the cache. This would also have a (CPU) overhead. Might be faster.

- All SVTs in the scene share the same tile detection pass. This means that there is only one render operation in that pass. This might be good or bad, I'm not quite sure yet.

### 1.5.16 Mesh warping

It is possible to apply an arbitrary warping mesh to distort the final image using a **PFM** (portable float map) file.

The file format is rather simple, and is described here. The file contains an array of NxM 3-component pixels in RGB (grayscale not supported). Each position contains the mapped resulting location in UV, in the R and G components respectively, in $[0, 1]$. The geometry warp format is the same as in the MPCDI v2.0 specification, section 3.6.2 (see here).

This file is read and converted into a mesh by Gaia Sky. The mesh is used to distort the final image at the end of the rendering pipeline.

In order to specify a PFM warping mesh file, you need to edit the *configuration file* of Gaia Sky and add a few lines in the `postprocess` section:

```
postprocess:
  [...]
  warpingMesh:
    pfmFile: /path/to/your/warping-mesh.pfm
```

A few warping mesh examples (big endian) are provided below:

- **Identity** – `warp-identity.pfm` – identity function, $x' = x, y' = y$.
- **Flip X** – `warp-invert-x.pfm` – flips the X coordinate, $x' = 1 - x, y' = y$.
- **Flip Y** – `warp-invert-y.pfm` – flips the Y coordinate, $x' = x, y' = 1 - y$.
- **Flip XY** – `warp-invert-xy.pfm` – flips the X and Y coordinates, $x' = 1 - x, y' = 1 - y$.
- **X^2** – `warp-x2.pfm` – applies a square function to X, $x' = x^2, y' = y$.
- **X^2, Y^2** – `warp-x2y2.pfm` – applies a square function to X and Y, $x' = x^2, y' = y^2$.

## 1.6 Gaia Sky VR

**Note:** Gaia Sky VR is **beta** software. It works reasonably well, but you may encounter hiccups here and there..

Gaia Sky VR is the VR version of Gaia Sky. It runs on multiple headsets and operating systems using the OpenXR API.

**Contents**


- *Gaia Sky VR*
  - *System requirements*
  - *Set-up*
    - *Windows*
    - *Linux*

- *Downloading datasets*
- *Controls*
- *Caveats*
- *Common problems*


Our tests have been carried out with the **Oculus Rift CV1** headset on Windows and the **Valve Index** on Windows and Linux. We also successfully tested it with the **HP Reverb G2**. Due to the system-agnostic nature of OpenXR, other VR HMD systems and controllers supporting OpenXR should also work fine.

**Note:** Gaia level-of-detail star catalogs don't work very well in VR and may cause performance issues. We recommend using static star catalogs like DR3-tiny, DR3-weeny, Hipparcos or GCNS5.

Currently, the regular installation of Gaia Sky also includes the VR version.

### 1.6.1 System requirements

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

| | |
|---|---|
| **VR System** | OpenXR-compatible VR system (HMD, VR controllers, trackers) |
| **Operating system** | Windows 10+ / Linux |
| **CPU** | Intel Core i5 4rd Generation or similar (4+ core) |
| **GPU** | VR-capable GPU (GTX 970+ strongly recommended) |
| **Memory** | 8+ GB RAM |
| **Hard drive** | 1 GB of free disk space (depends on datasets) |

### 1.6.2 Set-up

Essentially, you need an OpenXR runtime installed system-wide.

1. **Install runtime** — Follow the provided vendor instructions and install the software PC application for your VR headset. This application provides the OpenXR runtime. This is the Oculus/Meta PC app for Meta headsets, or SteamVR for the HTC Vive/Pro and the Valve Index, for example.
2. **Set active OpenXR runtime** — Set the runtime as the **active OpenXR runtime**. This typically is in the settings dialog of the vendor software. This step enables your particular OpenXR runtime to be discoverable by OpenXR-enabled applications like Gaia Sky VR.
3. **Run Gaia Sky VR** — Launch Gaia Sky VR and it should connect to your active OpenXR runtime automatically. Refer to the following sub-sections to learn how to launch Gaia Sky VR for your system.

#### Windows

The easiest way to get it running in Windows is to install the latest version of Gaia Sky and directly run the executable **`gaiaskyvr.exe`** file. You should also have a start menu entry called 'Gaia Sky VR', if you chose to create it during the installation.

#### Linux

Download and install Gaia Sky, and then run:

```
$  gaiasky -vr
```

### 1.6.3 Downloading datasets

See the *Dataset manager* section.

### 1.6.4 Controls

OpenXR defines a system-agnostic input scheme where the application defines actions which can be bound to different input device hardware. We offer a set of comprehensive bindings for the Oculus Rift CV1, the HTC Vive, the Valve Index and some others. However, if your headset is not supported you can bind the actions to your controller input yourself in your runtime. Please consult the documentation of your OpenXR runtime to learn how to do so.

The default controls are the following:

### 1.6.5 Caveats

Gaia Sky VR has been tested with a very small sample of VR systems. Only the Oculus Rift CV1 and the Valve Index are currently well tested. Please, do not expect everything to work flawlessly with other systems and/or headsets.

### 1.6.6 Common problems

- Make sure your runtime is set as the active OpenXR runtime.
- If you experience low frame rates try using a small and static star catalog like DR3-weeny or Hipparcos instead of a Gaia DR3 LOD one.
- If you are using an Nvidia 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.

## 1.7 Additional resources

This page gathers a list of learning resources about Gaia Sky. Find them below int the video tutorials and workshops sections.

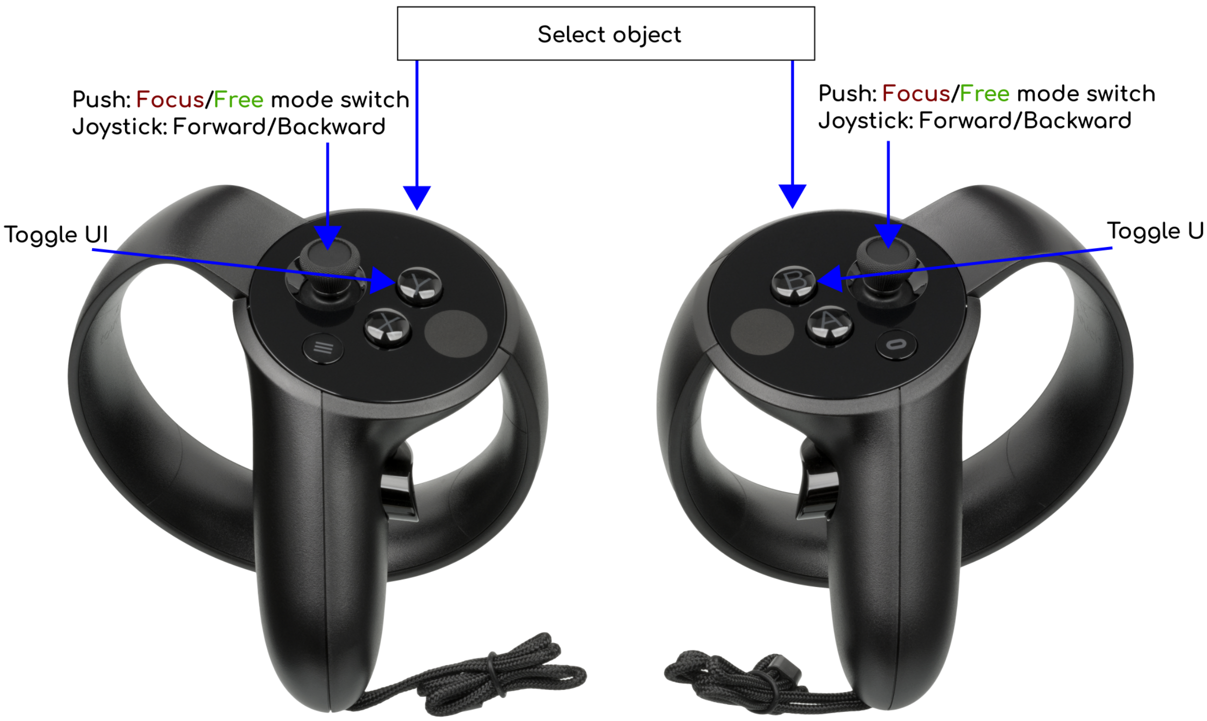



Fig. 88: The default controls for Gaia Sky VR with the Oculus Touch controllers

### 1.7.1 Video tutorials

- Gaia Sky video tutorials
- Gaia Sky videos channel

### 1.7.2 Presentations

- Gaia Sky VR – AR/VR for Space Programmes, ESA/ESTEC, Noordwijk, The Netherlands. 2023.

### 1.7.3 Workshop notes

#### Outreach tutorial (MWGaiaDN 2024)

**Hint:** This tutorial is designed to be followed with Gaia Sky 3.5.8!

This page contains the tutorial on the general usage of Gaia Sky given at the MWGaiaDN Induction School in PLNT Leiden (February 2, 2024).

The main aim of this tutorial is to provide a general understanding of the scripting system in Gaia Sky, and to train the participants in the creation of outreach videos.

**Presentation (slides):** Find the accompanying presentation for this tutorial here.

The topics covered in this tutorial are the following:

- Gaia Sky introduction:
  - Dataset manager.
  - Controls, movement, selection.
  - User interface.
  - Camera, type visibility.
  - Render modes (3D, planetarium, 360, reprojection).
  - Visual settings.
  - External datasets (loading, filters, SAMP).
- Scripting:
  - The API (basic functions, etc.).
  - Writing Python scripts for Gaia Sky.
  - Running scripts on a Gaia Sky instance.
  - Advanced topics: camera and scene parking runnables.
- Camera paths:
  - Recording and playback.
  - Keyframe system.
  - Still frame output system.
- Still frame output mode.

*Estimated duration:* 3 hours

### Before starting...

In order to follow the course it is strongly recommended to have a local installation of Gaia Sky 3.5.8 so that you can explore and try out the teachings for yourself. In order to install Gaia Sky, follow the instructions for your operating system in *the installation section* of the Gaia Sky documentation.

### Welcome window

When your start up Gaia Sky 3.5.8, you will be greeted with this view:

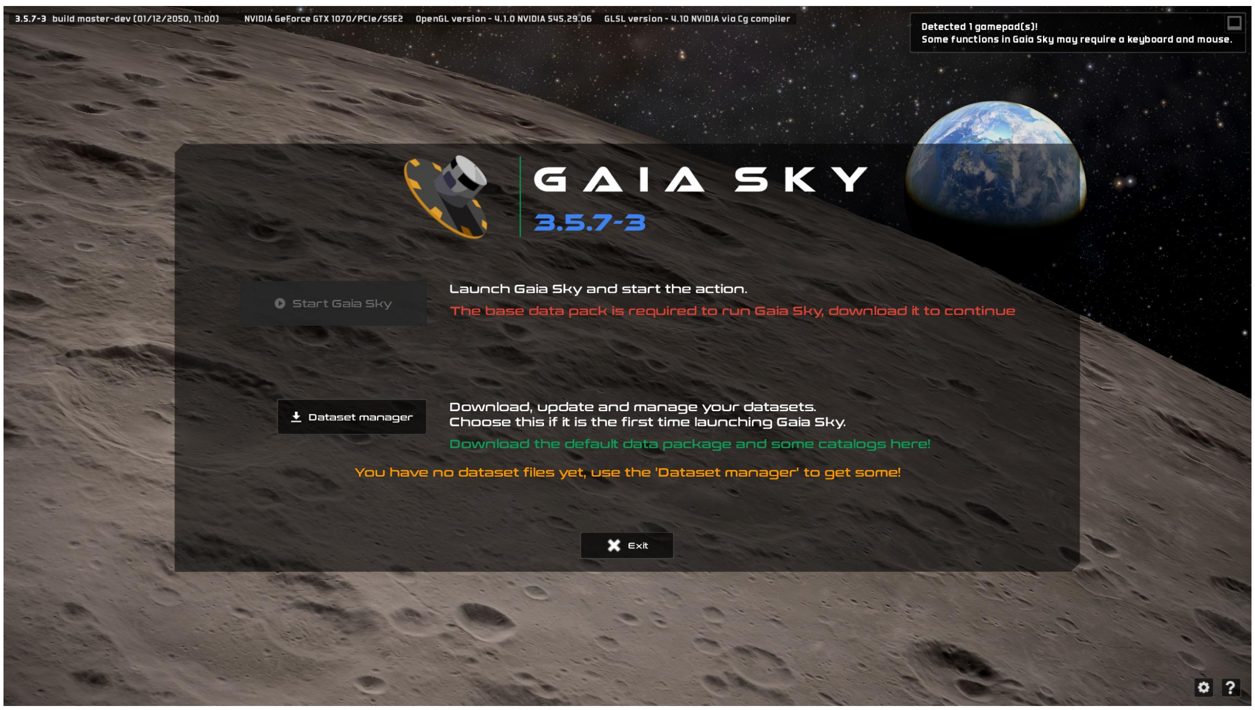



Fig. 89: Gaia Sky welcome UI

From here you can access the global preferences (cog wheel to the bottom right), fire up the **dataset manager**, or start Gaia Sky.

### Dataset manager

The dataset manager is used to download, update and manage datasets. It consists of two tabs:

- **Available for download** — contains datasets that are available to be downloaded and installed.
- **Installed** — contains the datasets currently installed locally.

The first time your start Gaia Sky you need to download at least the **Base data pack** (key `default-data`) to even be able to start the program. The base data pack contains essential data like most of the Solar System, the Milky Way, grids, constellations and other objects.

You can explore the available datasets freely.

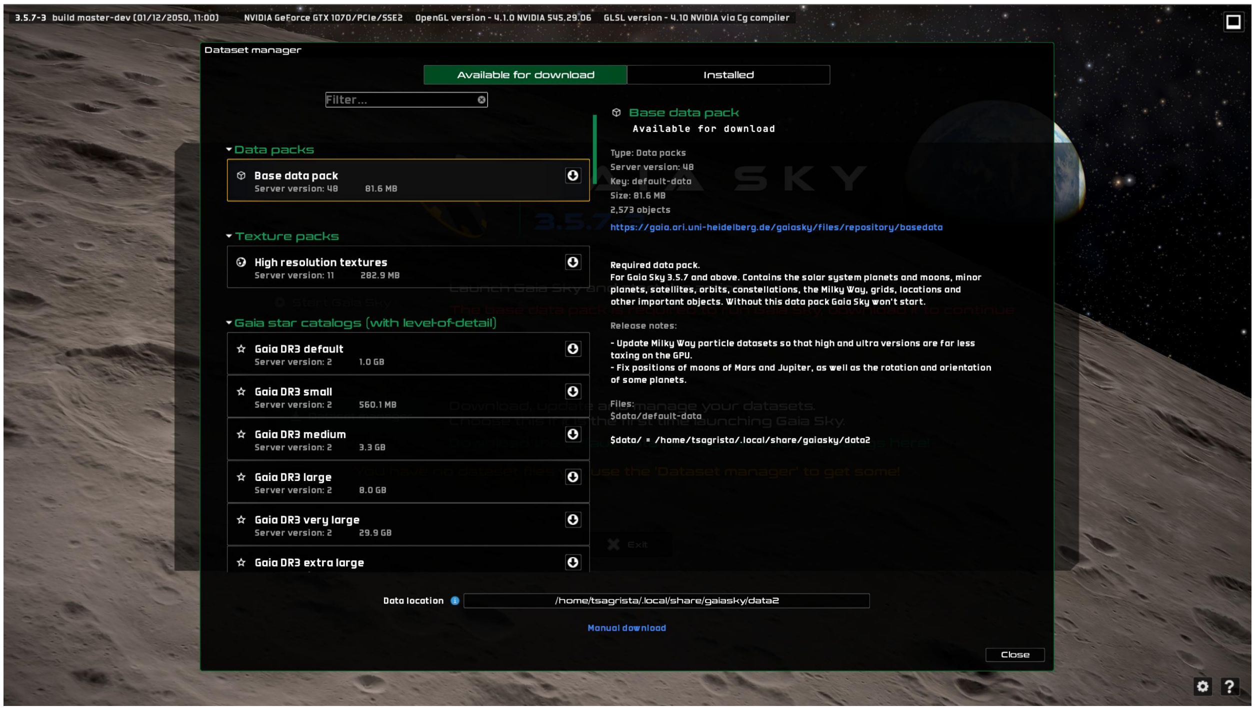



Fig. 90: The dataset manager

### Basic controls

When Gaia Sky is ready to go, you will be presented with this screen:

In it you can see a few things already. To the bottom right the *focus panel* tells you that you are in focus mode, meaning that all our movement is relative to the focus object. The default focus of Gaia Sky is the Earth. You can also see in the **quick info bar** at the top that our focus is the Earth, and that the closest object to our location is also the Earth. Additionally you see that your home object is again the Earth. Finally the **control panes** are accessible via the buttons anchored to the top left. If you click on one of these buttons, the respective pane opens. We will use them later.

### Movement

But right now let's try some movement. In **focus mode** the camera will by default orbit around the focus object. **Try clicking and dragging with your left mouse button**. The camera should orbit around the Earth showing parts of the surface which were previously hidden. You will notice that the whole scene rotates. Now **try scrolling with your mouse wheel**. The camera will move either farther away from (scroll down) or closer up to (scroll up) the Earth. You can always press and hold *z* to speed up the camera considerably. Now, if you **click and drag with your right mouse button**, you can offset the focus object from the center, but your movement will still be relative to it.

You can also use your keyboard arrows ← ↑ → ↓ to orbit left or right around the focus object, or move closer to or away from it.

You can use *shift* with a **mouse drag** in order to roll the camera.

More information on the controls is available in the *controls section* of the Gaia Sky user manual.

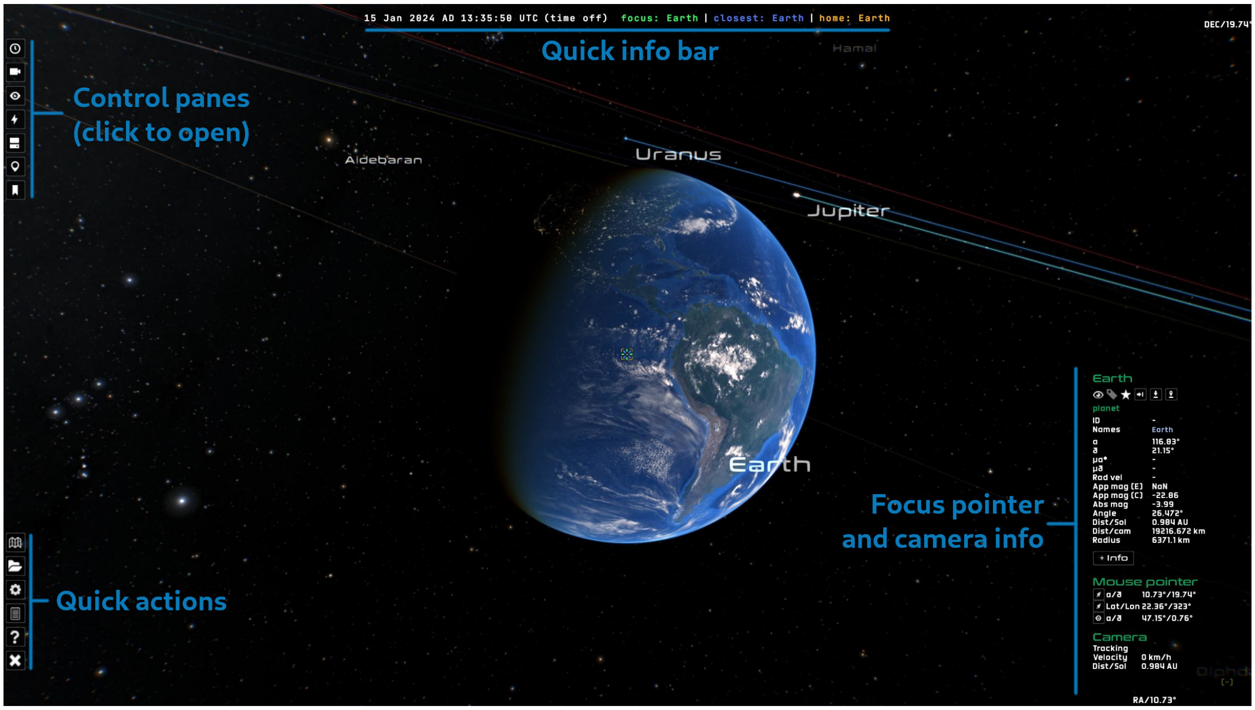
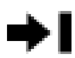



Fig. 91: Gaia Sky default scene

### Selection

You can change the focus by simply **double clicking** on any object on the scene. You can also press *f* to bring up the **search** dialog where you can look up objects by name. Try it now. Press *f* and type in "mars", without the quotes, and hit *esc*. You should see that the camera now points in the direction of Mars. To actually go to Mars simply scroll up until you reach it, or click on the  icon next to the name in the focus info panel. If you do so, Gaia Sky takes control of the camera and brings you to Mars.

If you want to move instantly to your current focus object, hit *ctrl* + *g*.

At any time you can use the *home* key in your keyboard to return back to Earth or whatever home object you have defined in the *configuration file*.

### The User Interface

The user interface of Gaia Sky consists of basically two components: keyboard shortcuts and a graphical user interface in the form of a few panes, buttons and windows. The most important of those are the **control panes**, accessible via a series of buttons anchored to the left.

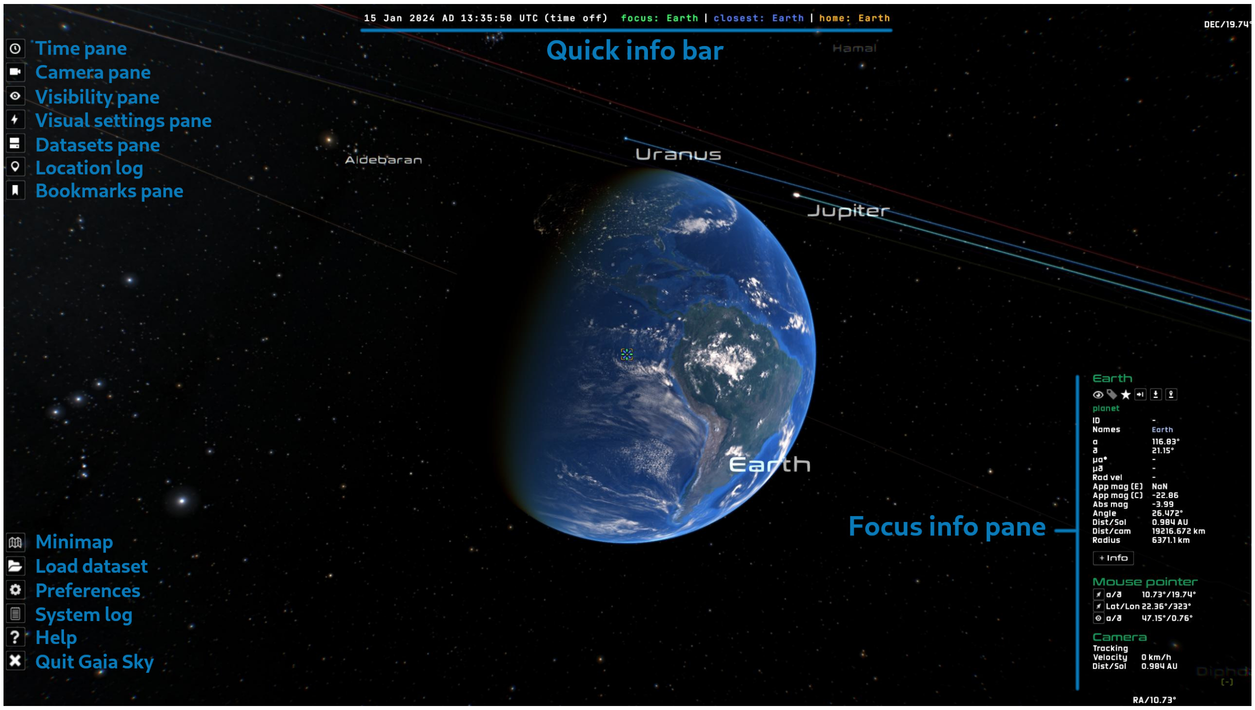


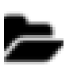
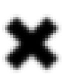

Fig. 92: Gaia Sky user interface with the most useful functions

### Control panes

**Hint:** The control panes are described in detail in *its own section* of the Gaia Sky user manual.

The control panes (previously called *control panel* in the old UI—it can still be used but is off by default) are made up of seven different panes, accessed using the buttons anchored to the top-left: *Time*, *Camera*, *Type visibility*, *Visual settings*, *Datasets*, *Location log* and *Bookmarks*. Each pane can be **expanded** and **collapsed** by clicking on the button or by using the respective keyboard shortcut (listed in the button tooltip).

Anchored to the bottom-left of the screen we can find six buttons to perform a few special actions:

-  Toggle the mini-map
-  Load a dataset
-  Open the preferences window
-  Show the session log
-  Show the help dialog
-  Exit Gaia Sky

### Quick info bar

To the top of the screen you can see the **quick info bar** which provides information on the current time, the current focus object (if any), the current closest object to our location and the current home object. The colors of these objects (green, blue, orange) correspond to the colors of the crosshairs. The crosshairs can be enabled or disabled from the interface tab in the preferences window (use *p* to bring it up).

### Debug panel

Gaia Sky has a built-in debug information panel that provides system information and is hidden by default. You can bring it up with *ctrl* + *d*, or by ticking the "*Show debug info*" check box in the system tab of the preferences window. By default, the debug panel is collapsed.


Fig. 93: Collapsed debug panel

You can expand it with the + symbol to get additional information.

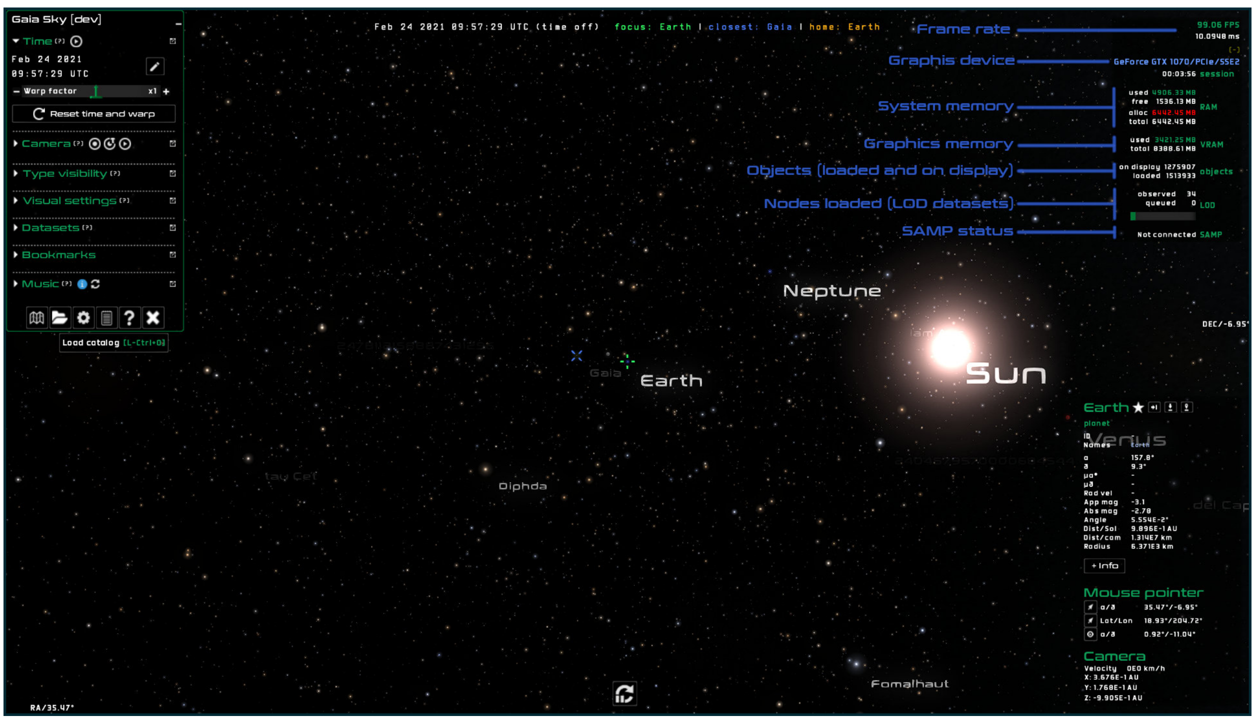



Fig. 94: Expanded debug panel

As you can see, the debug panel shows information on the current graphics device, system and graphics memory, the amount of objects loaded and on display, the octree (if a LOD dataset is in use) or the SAMP status.

Additional debug information can be obtained in the system tab of the help dialog (*?* or *h*).

### Time controls

Gaia Sky can simulate time. Play and pause the simulation using the / `Play/Pause` buttons in the time pane, or toggle using *Space*. You can also change time warp, which is expressed as a scaling factor, using the provided **Warp factor** slider. Use , or and . or to divide by 2 and double the value of the time warp respectively. If you keep either of those pressed, the warp factor will increase or decrease steadily.

Use the *Reset time and warp* button to reset the time warp to x1, and set the time to the current real world time (UTC).


Fig. 95: The time pane in the controls window of Gaia Sky.

Now, go ahead and press *home*. This will bring us back to Earth. Now, start the time with or *space* and **drag the slider slightly to the right** to increase its speed. You will see that the Earth rotates faster and faster as you move the slider to the right. Now, **drag it to the left** until time is reversed and the Earth starts rotating in the opposite direction. Now time is going backwards!

If you set the time warp high enough you will notice that as the bodies in the Solar System start going crazy, the stars start to slightly move. That's right: Gaia Sky also simulates proper motions.

### Camera modes

We have already talked about the **focus camera mode**, but Gaia Sky provides some more *Camera modes*:

- *0* - **Free mode**: the camera is not locked to a focus object and can roam freely. The movement is achieved with the scroll wheel of your mouse, and the view is controlled by clicking and draggin the left and right mouse buttons
- *1* - **Focus mode**: the camera is locked to a focus object and its movement depends on it
- *2* - **Game mode**: similar to free mode but the camera is moved with *wasd* and the view (pitch and yaw) is controlled with the mouse. This control system is commonly found in FPS (First-Person Shooter) games on PC
- *3* - **Spacecraft mode**: take control of a spacecraft (outside the scope of this tutorial)

The most interesting mode is **free mode** which lets us roam freely. Go ahead and press *0* to try it out. The controls are a little different from those of **focus mode**, but they should not be to hard to get used too. Basically, use your **left mouse button** to yaw and pitch the view, use *shift* to roll, and use the **right mouse button** to pan.

### Special render modes

There are three special render modes: *3D mode*, *planetarium mode*, *panorama mode* and *orthosphere view*. You can access these modes using the buttons at the bottom of the camera pane or the following shortcuts:

- or *ctrl* + *s* - 3D mode
- or *ctrl* + *p* - Planetarium mode
- or *ctrl* + *k* - Panorama mode
- or *ctrl* + *j* - Orthosphere view

### Component visibility

The visibility of most graphical elements can be switched off and on using the buttons in the **type visibility pane** in the control panel. For example you can hide the stars by clicking on the `stars` button. The object types available are the following:

- – Stars
- – Planets
- – Moons
- – Satellites
- – Asteroids
- – Star clusters
- – Milky Way
- – Galaxies
- – Nebulae
- – Meshes
- – Equatorial grid
- – Ecliptic grid

- – Galactic grid
- – Labels
- – Titles
- – Orbits
- – Locations
- – Cosmic locations
- – Countries
- – Constellations
- – Constellation boundaries
- – Rulers
- – Particle effects
- – Atmospheres
- – Clouds
- – Axes
- – Velocity vectors
- – Others

### Velocity vectors

One of the elements, the **velocity vectors**, enable a few properties when selected. See the *velocity vectors section* in the Gaia Sky user manual for more information on that.

- **Number factor** – control how many velocity vectors are rendered. The stars are sorted by magnitude (ascending) so the brightest stars will get velocity vectors first
- **Length factor** – length factor to scale the velocity vectors
- **Color mode** – choose the color scheme for the velocity vectors
- **Show arrowheads** – Whether to show the vectors with arrow caps or not

**Hint:** Control the width of the velocity vectors with the **line width** slider in the **visual settings** pane.

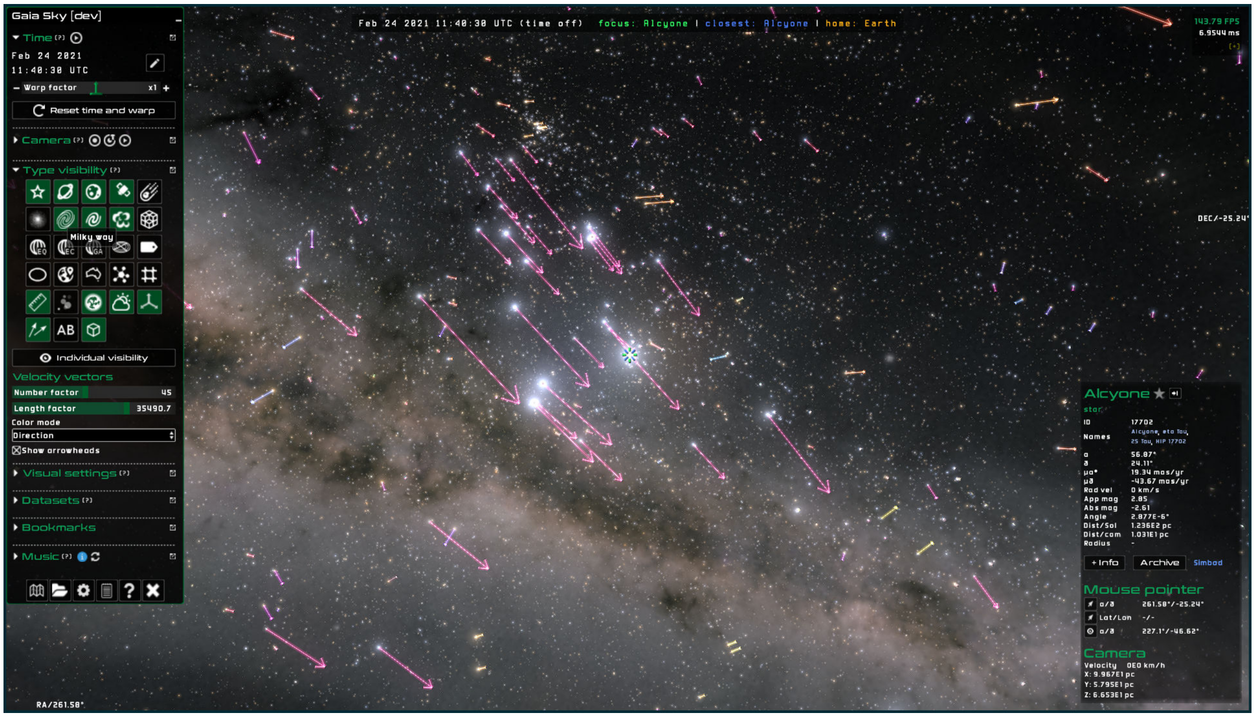



Fig. 96: Velocity vectors in Gaia Sky

## Visual settings

The **visual settings** pane contains a few options to control the shading of stars and other elements:

- **Star brightness** – control the brightness of stars
- **Magnitude multiplier** – exponent of power function that controls the brightness of stars. Controls the brightness difference between bright and faint stars
- **Star glow factor** – close-by star size
- **Point size** – size of point-like stars and other objects
- **Base star level** – the minimum brightness level for all stars
- **Ambient light** – control the amount of ambient light. This only affects the models such as the planets or satellites
- **Line width** – control the width of all lines in Gaia Sky (orbits, velocity vectors, etc.)
- **Label size** – control the size of the labels
- **Elevation multiplier** – scale the height representation for planets with elevation maps

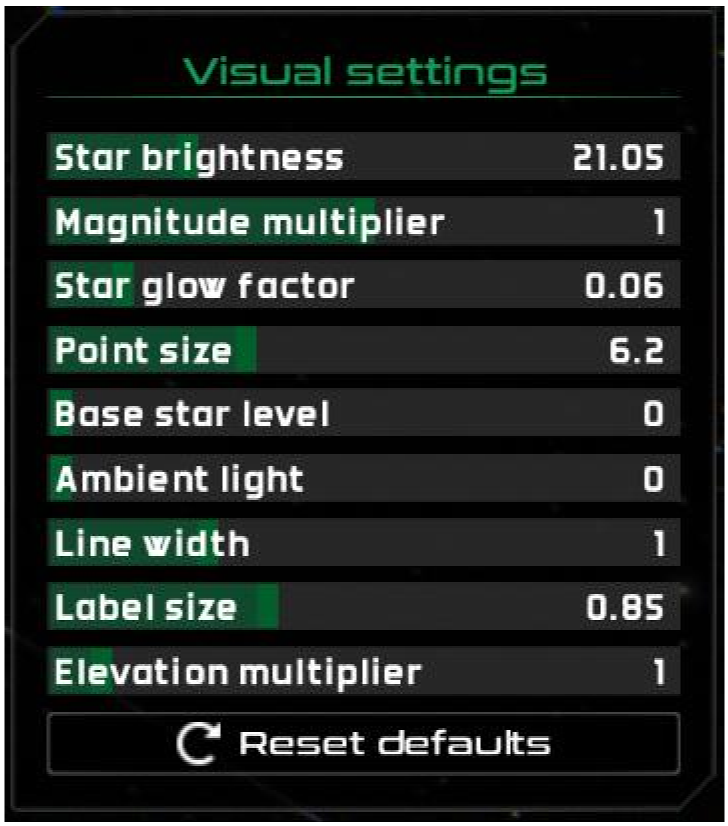



Fig. 97: The visual settings pane.

### External datasets

Gaia Sky supports the loading of external datasets at runtime. Right now, `VOTable`, `csv` and `FITS` formats are supported. Gaia Sky needs some metadata in the form of UCDs or column names in order to parse the dataset columns correctly. Refer to the *STIL data loader* section of the Gaia Sky user manual 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 control panel.

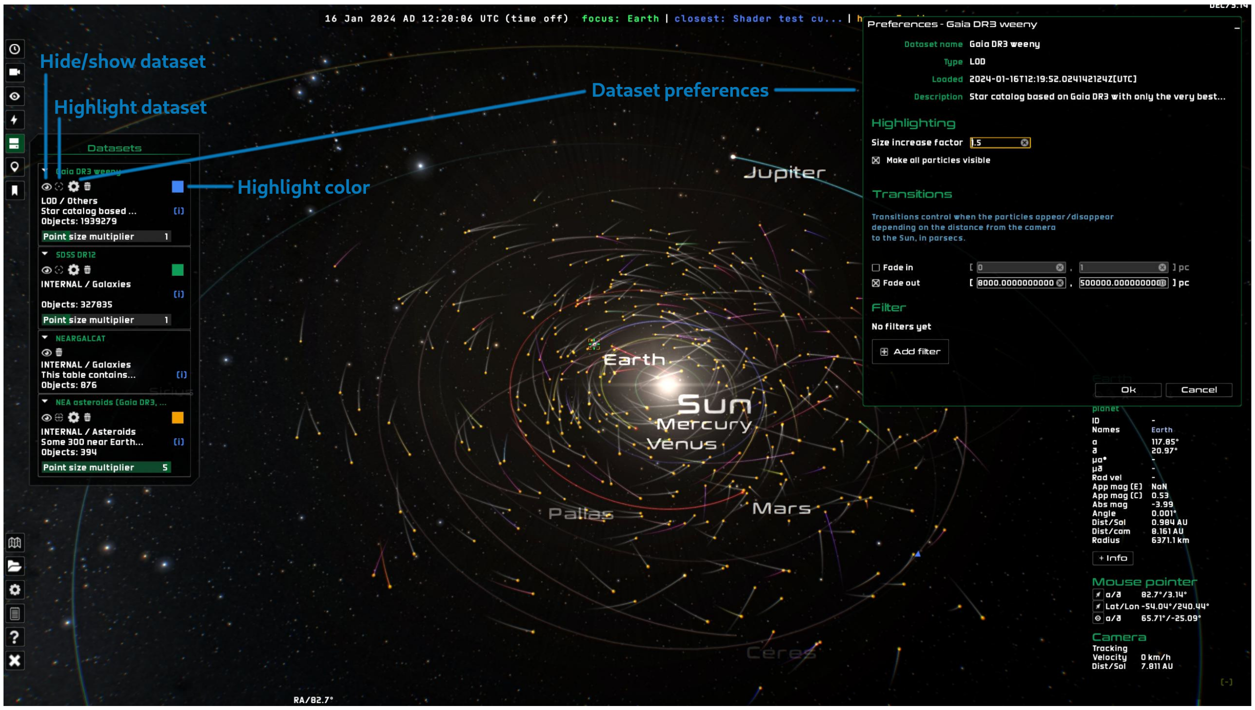



Fig. 98: Datasets pane of Gaia Sky.

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

- Directly from the UI, using the button or pressing *ctrl + o*
- Through SAMP, via a connection to another astronomy software package such as Topcat or Aladin
- Via a script (addressed later on in the workshop if time allows)
- Creating a descriptor file, saving it, along with the dataset, in the data directory, and selecting it in the dataset manager (avanced!)

**Loading a dataset from the UI** – Go ahead and remove the current star catalog by clicking on the icon in the datasets pane. Now, download a raw Hipparcos dataset VOTable, click on the icon (or press *ctrl + o*) and select the file. In the next dialog just click *Ok* to start loading the catalog. 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 on 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 (Odysee mirror, YouTube mirror):

**Loading a dataset via scripting** – Wait for the scripting section of this course.

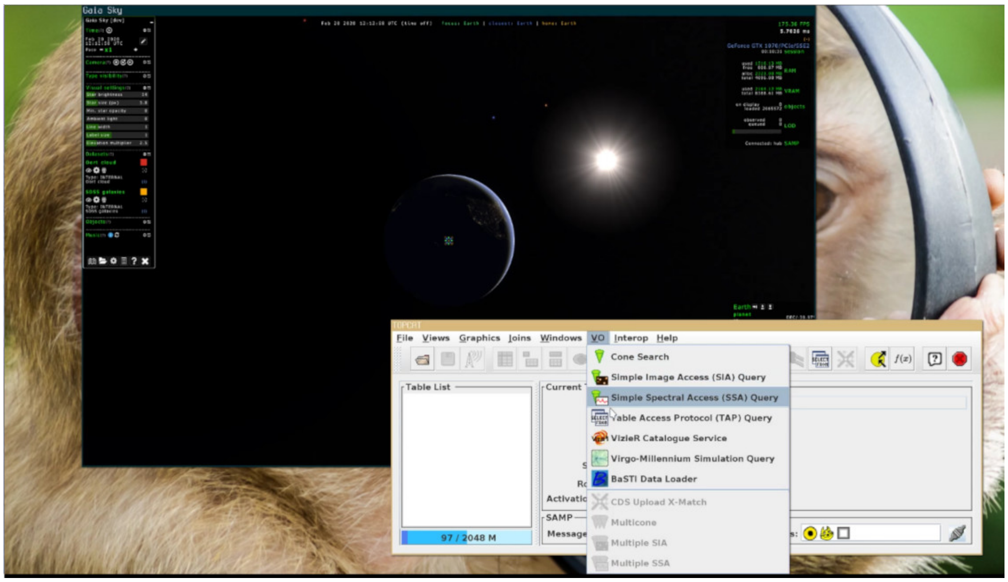



Fig. 99: Loading a dataset from Topcat through SAMP (click for video)

**Preparing a descriptor file** – Not addressed in this tutorial. See *the catalog formats section* for more information.

## Working with datasets

All datasets loaded are displayed in the datasets pane in the control panel. A few useful tips for working with datasets:

- The visibility of individual datasets can be switched on and off by clicking on the button
- Remove datasets with the button
- You can **highlight a dataset** by clicking on the button. The highlight color is defined by the color selector right on top of it. Additionally, we can map an attribute to the highlight color using a color map. Let's try it out:
  1. Click on the color box in the Hipparcos dataset we have just loaded from Topcat via SAMP
  2. Select the radio button "Color map"
  3. Select the *rainbow* color map
  4. Choose your attriubte. In this case, we will use the number of transits, *ntr*
  5. Click *Ok*
  6. Click on the highlight dataset icon to apply the color map
- You can **define basic filters** on the objects of the dataset using their attributes from the dataset preferences window. For example, we can filter out all stars with $\delta > 50°$:
  1. Click on the dataset preferences button
  2. Click on *Add filter*
  3. Select your attribute (declination $\delta$)
  4. Select your comparator ($<$)

5. Enter your value, in this case 50
6. Click *Ok*
7. The stars with a declination greater than 50 degrees should be filtered out

Multiple filters can be combined with the **AND** and **OR** operators

## External information

Gaia Sky offers three ways to display external information on the current focus object: **Wikipedia**, **Gaia archive** and **Simbad**.

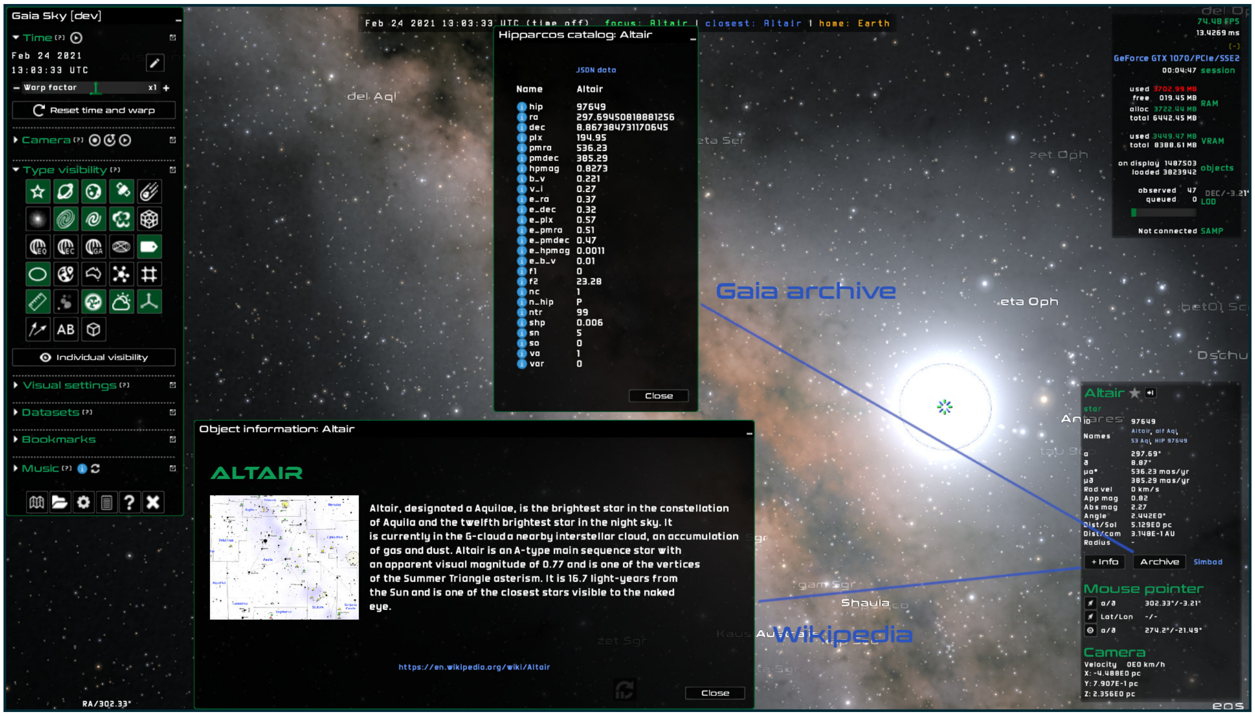



Fig. 100: Wikipedia, Gaia archive and Simbad connections

- When the *+Info* button appears in the focus info pane, it means that there is a Wikipedia article on this object ready to be pulled and displayed in Gaia Sky
- When the *Archive* button appears in the focus info pane, it means that the full table information of selected star can be pulled from the Gaia archive
- When the `Simbad` link appears in the focus info pane, it means that the objects has been found on Simbad, and you can click the link to open it in your web browser

### Scripting

Gaia Sky exposes an API that is accessible through Python (via Py4j) or through HTTP over a network (using the *REST API HTTP server*). The full documentation on the scripting system can be found in the *scripting section* of the Gaia Sky user manual.

In this tutorial, we focus on the writing of Python scripts that tap into the Gaia Sky API.

- Scripting **API specification**:
  - API Gaia Sky master (development branch)
  - API Gaia Sky 3.5.8
- Interesting **showcase scripts** can be found here.
- Basic **testing scripts** can be found here.

This section includes a **hands-on session** where we work on some scripts (full file listing) and write new ones to later run them on Gaia Sky. The scripts are:

- Locating_the_Hyades_tidal_tails.py — a simple sequential script which exemplifies some of the most common API calls, and can be used to capture a video. The script requires the following data and subtitles files to run (save them in the same directory as the script):
  - Aldebaran.vot
  - Hyades_stars.csv
  - Hyades_subtitles.srt
  - distSDR3_N.csv
- line-objects-update.py — a script showcasing the feature to run scripting code within the Gaia Sky main loop, so that it runs every frame. This is used to run update operations every single frame. In our test script, we create a line between the Earth and the Moon, start the time simulation, and update the position of the line every frame so that it stays in sync with the scene.

### Camera paths

Gaia Sky includes a feature to record and play back camera paths. This comes in handy if you want to showcase a certain itinerary through a dataset, for example.

**Recording a camera path** — The system 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 control panel. 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 the *folders* section in the Gaia Sky documentation).

**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 camcorder 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.

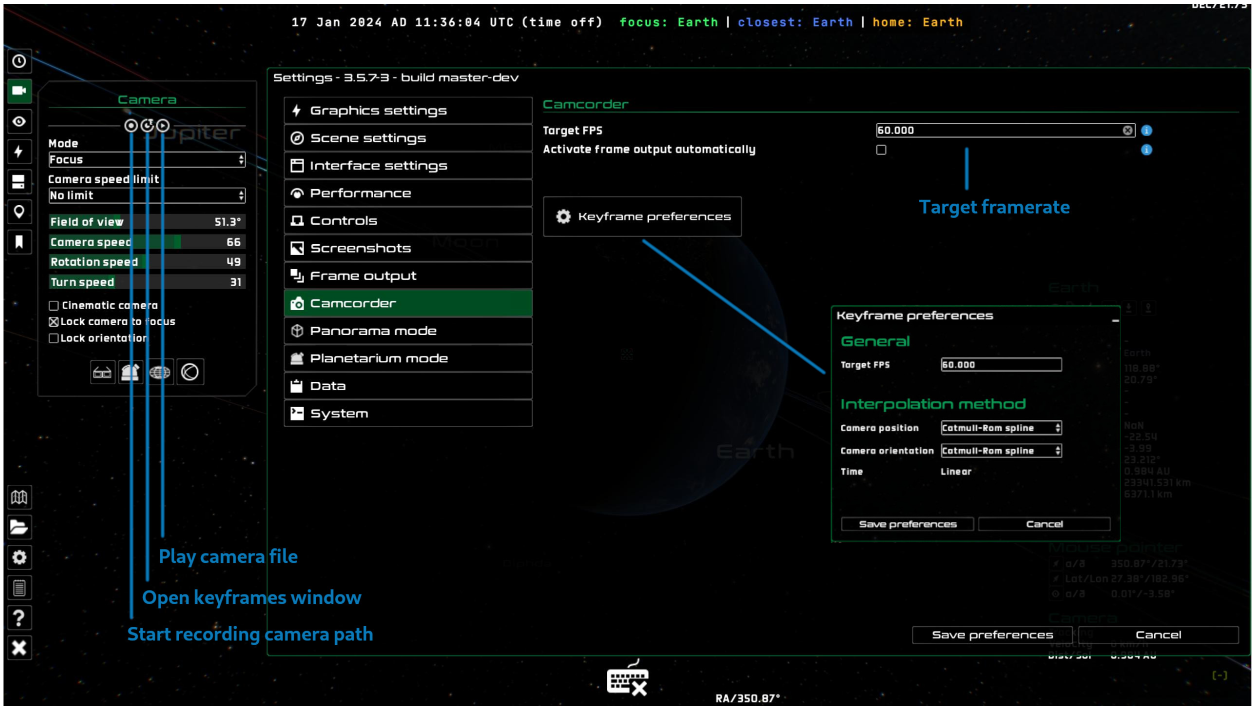



Fig. 101: Location of the controls of the camcorder in Gaia Sky

More information on camera paths in Gaia Sky can be found in *the camera paths section* of the Gaia Sky user manual.

### Keyframe system

The camera path system 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 outside 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 Gaia Sky user manual.

### Frame output mode

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.

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* of the Gaia Sky user manual.

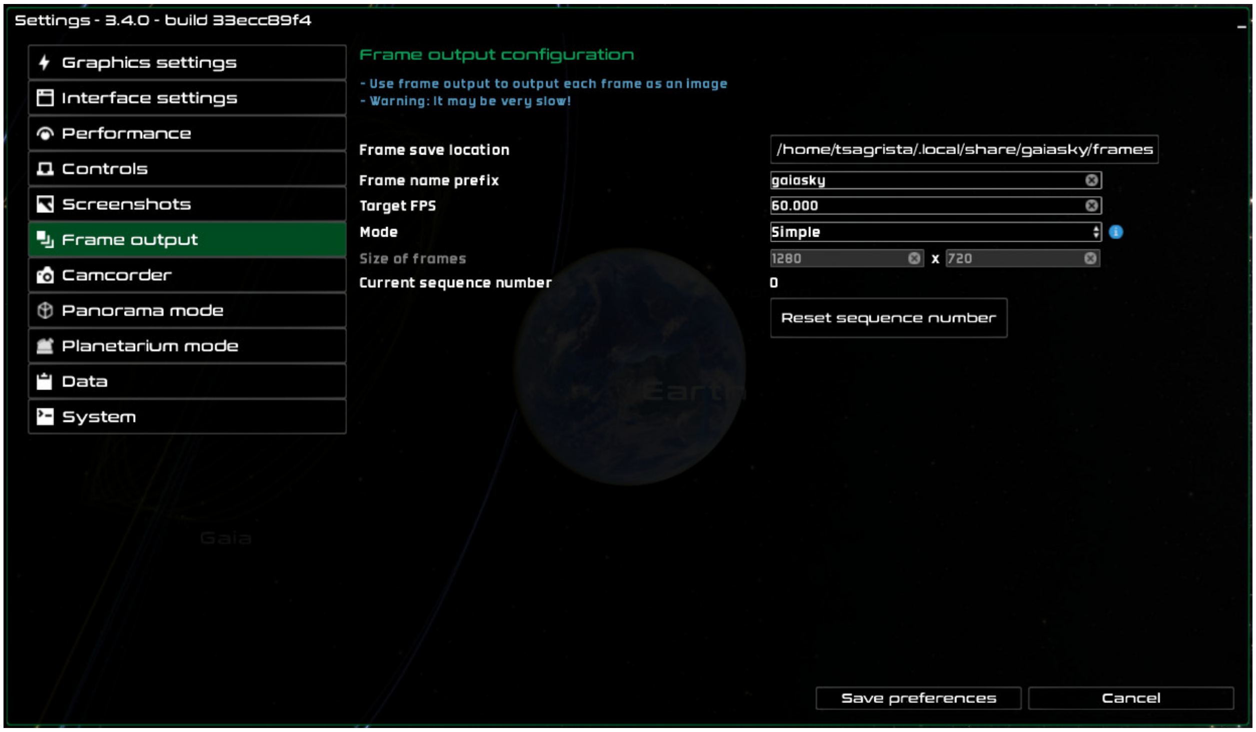



Fig. 102: The configuration screen for the frame output system

### Conclusion

Congratulations! You have reached the end of the tutorial. You are now a Gaia Sky master ;)

### Scripting workshop (DPAC 2023)

This page has been retired. However, you can still browse the workshop notes in the link below:

- Scripting workshop notes (DPAC 2023)

### General tutorial (DPAC 2021)

This page has been retired. However, you can still browse the tutorial notes in the link below:

- Tutorial notes (DPAC 2021)

### Video production tutorial (DPAC 2020)

This page has been retired. However, you can still browse the tutorial notes in the link below:

- Video production tutorial notes (DPAC 2020)

## 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 package*.

### 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 Gaia-DR catalogs?

We offer several different catalogs based on the latest Gaia data release. **Only one** should be used at a time, as they are different subsets of the same data, meaning that smaller catalogs are contained in larger catalogs. For example, the stars in `edr3-default` are contained in `edr3-large`. We offer so many to give the opportunity to explore the Gaia data to everyone. Even if you have a low-end PC, or don't have lots of disk space to spare, you can still run Gaia Sky with the smaller subsets, which only contain the 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 at start-up, what to do?

First, make sure that your drivers are up to date and your graphics card supports OpenGL `3.2` and GLSL `3.3`.

Some startup crashes are due to inconsistencies in the data. Usually, removing the data folder (`~/.local/share/gaiasky/data` on Linux, `%userprofile%\.gaiasky\data` on Windows, `~/.gaiasky/data` on macOS) solves the problem. When Gaia Sky starts again, you will need to re-download the base data pack and the datasets.

**Debug mode**

You can activate *debug mode* to force Gaia Sky to print out much more information, which may help in pinpointing what is going wrong. To do so, you need to launch Gaia Sky from the command line (PowerShell or cmd on Windows) using the *-d* flag.

```
$  gaiasky -d
```

**Configuration file**

Sometimes, the configuration file may get corrupted. To fix this, remove it (`~/.config/gaiasky/config.yaml` on Linux, `%userprofile%\.gaiasky\config.yaml` on Windows, `~/.gaiasky/config.yaml` on macOS) and start Gaia Sky again. The default configuration file will be copied to that location and used.

**Getting a crash log**

For modern Gaia Sky versions (> 2.2.0), you can *find the logs in this location*.

For old Gaia Sky versions (< 2.2.0), you may need to run Gaia Sky from a terminal. In this case, the procedure depends on your Operating System.

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

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:

```
$ cd /Applications/Gaia\ Sky.app/Contents/Resources/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'm running out of memory, what to do?

Don't fret. Check out the *maximum heap space section* to learn how to increase the maximum heap memory allocated to Gaia Sky. If you computer does not have enough physical RAM, try using a smaller dataset.

### 1.8.6 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. Finally, select *Tessellation* in the "Elevation representation" drop-down of the graphics pane in the settings window. See the *elevation (height) section*.

### 1.8.7 Q: What is the internal reference system used in Gaia Sky?

The reference system is described in *Internal reference system*. The internal workings of Gaia Sky are described in this paper.

### 1.8.8 Q: Can I contribute?

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

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

Thanks a lot, but no. You may donate to any other awesome open source project of your choosing instead.

## 1.9 Changelog

- Comprehensive version history
- Detailed changelog
- Full commit history

## 1.10 About

### 1.10.1 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 at tsagrista@ari.uni-heidelberg.de.

Do not forget to visit our Homepage@ARI.

### 1.10.2 Author

**Toni Sagristà Sellés** – tonisagrista.com

### 1.10.3 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

### 1.10.4 Stats

Gaia Sky download numbers (including documentation requests and data packages) can be found here.