CosmoScout VR: A Modular 3D Solar System Based on SPICE

Simon Schneegans
German Aerospace Center (DLR)
Lilienthalplatz 7
Braunschweig, 38108
simon.schneegans@dlr.de

Moritz Zeumer
German Aerospace Center (DLR)
Lilienthalplatz 7
Braunschweig, 38108
moritz.zeumer@dlr.de

Jonas Gilg
German Aerospace Center (DLR)
Lilienthalplatz 7
Braunschweig, 38108
jonas.gilg@dlr.de

Andreas Gerndt
German Aerospace Center / University of Bremen
Lilienthalplatz 7
Braunschweig, 38108
andreas.gerndt@dlr.de

Abstract—**We present CosmoScout VR - a modular 3D Solar System for interactive exploration and presentation of large space mission datasets. This paper describes the overall architecture as well as several core components of the framework. To foster the application in various scientific domains, CosmoScout VR employs a plugin-based architecture. This not only reduces development times but also allows scientists to create their own data visualization plugins without having to modify the core source code of CosmoScout VR. One of the most important plugins — level-of-detail terrain rendering — is described in greater detail in this paper. Another key feature of CosmoScout VR is the scene graph which is tightly coupled with NASA's SPICE library to allow for high-precision positioning of celestial objects, such as planets, moons, and spacecrafts. SPICE is also used for the seamless navigation throughout the Solar System in which the user automatically follows the closest body. During navigation, the virtual scene is scaled in such a way, that the closest celestial body is always within arm's reach. This allows for simultaneous exploration of multiple datasets in their spatial context at diverse scales. However, the navigation uses all six degrees of freedom which can induce motion sickness. In this paper, we present some counter measures as well as evaluate their effectiveness in a user study. CosmoScout VR is open source, cross-platform, and while it can run on conventional desktop PCs, it also supports stereoscopic multi-screen systems, such as display walls, DOMEs or CAVEs.**

Table of Contents

1. Introduction

Within the last few decades, European and international Earth observation and space exploration missions have produced a wide variety of datasets, and new missions with highly capable instrument suites are about to be launched. This data may contain answers to fundamental questions for science and humanity: How did the Solar System evolve? Is there life on other planets? How will climate change on Earth? Many answers have only been found by combining various datasets, showing phenomena that otherwise remain uncovered.

Especially the visualization of data not only from multiple sources but also at diverse scales offers a significant potential for scientific discoveries. Usually, this requires using several visualization tools simultaneously which hinders a greater understanding as the data cannot easily be contextualized. Thus, an in-depth exploitation of the available data requires a holistic visualization framework to establish a common spatio-temporal reference frame.

Furthermore, visualizing data at interactive frame rates can foster explorative approaches which are particularly promising due to the potential for serendipitous findings. If this is combined with immersive stereoscopic rendering to enhance perception of complex 3D objects such as observation geometries or trajectories of spacecrafts, such a tool could not only be used for data analysis but also for space mission planning. However, this poses not only grand challenges due to the ever-growing size of the available data, but also due to the complexity of a human-computer interface which involves navigation in a dynamic multi-scale environment: our Solar System. Free navigation in such a virtual environment can easily induce a significant amount of cybersickness which hinders the adoption by scientists.

With these opportunities and challenges in mind, we are developing CosmoScout VR [1], a modular 3D Solar System for interactive and immersive data exploration. In this paper, we present the architecture, the core components, and several plugins for the software. After a state-of-the-art discussion in Section 2, we present the software architecture in Section 3. Thereafter, in Section 4, we describe CosmoScout's scene graph which provides high precision and seamless navigation throughout the entire Solar System based on SPICE [2]. We also provide details on the flexible HTML-based user interface, and our HDR-renderer which is key to produce realistic images. In Section 5, we present several plugins of CosmoScout VR with corresponding use cases from various scientific domains. We describe details of the terrain rendering plugin as well as a plugin to reduce the amount of induced cybersickness. Ultimately, we evaluate these aspects in Section 6 by presenting a cybersickness user study and a performance evaluation.

978-1-6654-3760-8/22/$31.00 ©2022 IEEE

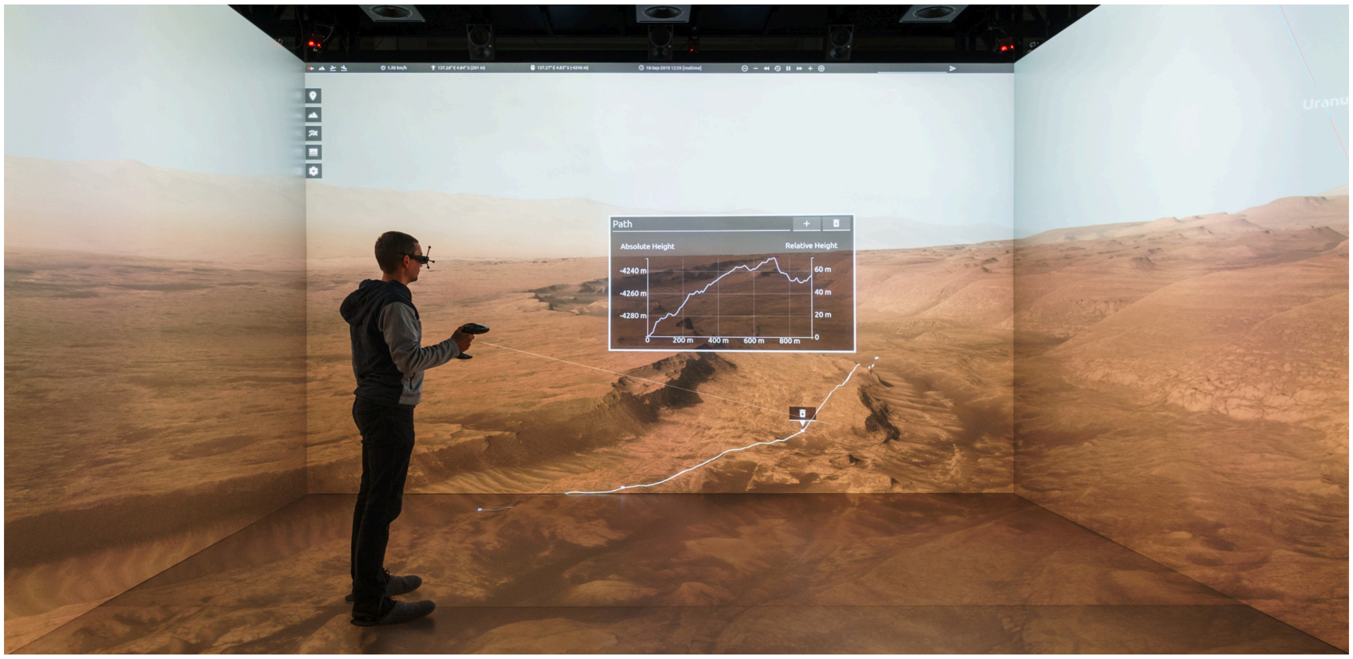

Figure 1. **Exploring Gale Crater on Mars: Interactive visualization of HiRISE data from the Mars Reconnaissance Orbiter using 28 projectors of the aixCAVE at RWTH Aachen.**

2. Related Work

Over the last decades, several open source visualizations of detailed planets or even the entire known Solar System have been created. Not only human fascination for space but also the added value through interactive data visualization have been driving factors in the development of these applications. Some are targeted at desktop-usage, such as the SPICE-enhanced Cosmographia [3] which is based on Celestia[2]. A similar project based on the Unity Game Engine is NASA's *Eyes on the Solar System* [4] with a web version currently in development. These tools are primarily aimed at educating the general public about the mechanics of the Solar System and past, ongoing, and future exploration efforts. Using web browsers as a target platform provides better accessibility and thus promises a more widespread application. CesiumJS [5] is a JavaScript framework for creating web-based virtual globes which has been used in many projects such as NASA's *Trek* pages[3]. Similarly, NASA's *World Wind* [6] can be used to explore data for a single planet (primarily Earth) in a web-based environment. Another promising platform is virtual reality (VR) as it offers higher engagement through increased immersion and promises insights into complex data due to stereo vision. All projects mentioned above are not specifically designed for virtual reality. A software which is primarily targeted at planetarium-like setups but also supports virtual reality is OpenSpace [7]. In fact, OpenSpace is a very similar project to the one presented in this paper with only a slightly shifted application focus. However, all features described in this paper are unique to CosmoScout VR and thus complement the state-of-the-art.

Regardless of the target platform, there are a number of challenges in this application area. Due to the extreme spatial extent, precision handling using floating point arithmetic gets challenging. Furthermore, 3D-navigation introduces a set of challenges which are unique to these sparse multi-scale scenes. In order to deal with the massive amounts of heterogeneous data, sophisticated level-of-detail rendering algorithms, client-server processing, and out-of-core data management is necessary.

Handling of Large Scenes — One cause for the first challenge is that nowadays GPUs are optimized for 32 bit single precision floating point numbers which can not sufficiently represent entire planets in the range of centimeters. A common solution to this problem is representing the scene at a higher resolution on the CPU and converting it to user-centered coordinates with a floating origin for rendering [8]. To represent the data on the CPU, various approaches have been proposed over the years. One of the most popular approaches is using power-scaled coordinates (PSC) [9] which add a scaling exponent to all 3D coordinates. While this drastically increases the available coordinate range, it still suffers from a decreasing precision with increasing distance from the origin of the scene. As a solution, Axelsson et al. recently proposed the Dynamic Scene Graph (DSG) [10] which allows for high precision at arbitrary positions in space. This is primarily achieved by modifying the standard depth-first scene graph traversal so it starts at a scene graph node in the vicinity of the virtual camera and thus avoids large (and hence error-prone) translation values for objects close to the camera.

In CosmoScout VR, we opted for an approach similar to the DSG which is based on the hierarchical double precision coordinate systems of SPICE [2]. While our approach (see Section 4) provides a similar precision, it is more easy to integrate in an existing single precision scene graph as no modification of the traversal scheme is required.

3D-Navigation — Another remarkable challenge of virtual reality applications is 3D-navigation as it can easily lead to cybersickness and disorientation. This is especially true for space simulations: Most of the time, the navigation must support all six degrees of freedom as there is no clear ubiquitous reference frame and the concept of up and down

[2]https://celestia.space/
[3]https://trek.nasa.gov/

changes frequently. At the same time, the Solar System is a dynamic multi-scale environment in which celestial bodies move with several thousands of kilometers per hour relative to each other. A comprehensive overview of existing multi-scale navigation approaches is given by Argelaguet et al., differentiating between discrete and continuous navigation [11]. In the discrete case, users may choose between a predefined set of fixed scales to work in. While this is suitable for various applications, a continuously changing scale seams more adequate for explorative navigation in the Solar System. In the context of astronomical visualizations, Fu et al. proposed an approach based on power-scaled coordinates where users can manually adjust the scale of the universe to fit the task at hand [12]. They also point out that in this context automatic select-and-go navigation models are very efficient but may result in a loss of spatial context. McCrae et al. proposed an approach where the scale of the scene is automatically adjusted so the closest object is at a fixed draw-distance [13]. The closest object is determined by rendering the scene to a low-resolution depth-cubemap in every frame which comes at a significant performance cost.

For CosmoScout VR, we also implemented an approach which dynamically adjusts the scene scale according to the closest object. In addition, the observer automatically follows the movements of the closest object. Furthermore, we avoid the additional rendering step by explicitly computing the distance to all potential objects. This is feasible due to the sparseness of the Solar System. The details of the implementation are given in Section 4.

Cybersickness Reduction — Between 30% and 80% of users encounter cybersickness (or VR-sickness) in some form or severity during their exposure to virtual environments [14]. To reduce the symptoms while using CosmoScout VR, we implemented and evaluated popular cybersickness mitigation techniques based on common theories about the source of motion- and cybersickness. The postural stability theory assumes that motion sickness is precedented by periods of postural instability, where small uncontrolled movements and changes in the subject's center of gravity occur [15]. To combat this, Chang et al. [16] and Duh et al. [17] projected a grid aligned with the real-world floor into the virtual environment to assist the user's orientation and postural stability. Another, theory about the origin of motion- and cybersickness is the sensory conflict theory. It assumes cybersickness symptoms are caused by the mismatch between the perceived environment and the subject's expectations, as hypothesized by Barret and Thornton in [18]. Vection, the illusion of self motion, is a commonly experienced phenomenon of this conflict as analysed by Keshavarz et al. in [19]. It is the result of the visual system receiving optical flow patterns, while the vestibular system does not perceive these changes in motion. This source of potential cybersickness is addressed through the implementation of a vignette, reducing vection by limiting the field of view during periods of high visual flow in the peripheral field as suggested by Fernandes and Feiner in [20]. The details of the implementations are given in Section 5.

The polysymptomatic and polygenic nature of cybersickness increases the difficulty to develop standardized and effective methods to reduce cybersickness symptoms, as well as measure their effectiveness across a diverse group of users. Historically, questionnaires have been a popular method to measure cybersickness due to the internal and subjective symptoms, and the large individual differences in symptom profiles and susceptibility [21]. The Fast Motion Sickness Scale (FMS) [22] provides a single item questionnaire focussing on

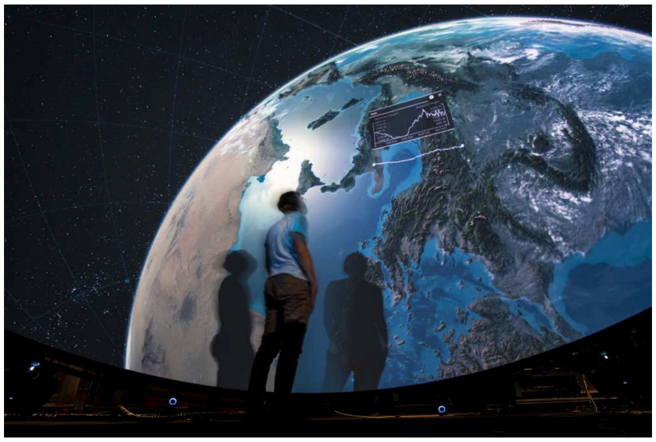
Figure 2. Thanks to software like Vioso's *Anyblend*, CosmoScout VR can be used for planetarium-like projections. The image shows CosmoScout VR running in the "ARENA" science dome of the GEOMAR Helmholtz Centre for Ocean Research in Kiel, Germany.

nausea and general discomfort, to rate the overall experience of motion sickness symptoms. This questionnaire can be used during the exposure to identify sections of increased risk to cybersickness, while traditional post-exposure questionnaires like the Simulator Sickness Questionnaire (SSQ) [23] only identify the most prevalent symptom category. To complement the subjective data with objective measurements, Chardonnet et al. analyze sway signals of the center of gravity in [24] to gather insight into events of motion sickness in virtual reality environments. Using an approach similar to Lim et al. in [25], we measure the center of gravity and postural sway via the VR-device's tracking data in the conducted pilot user study in Section 6.

Level-of-detail (LoD) Terrain Rendering — All Solar System simulations which attempt to display highly detailed planets, moons, and other celestial bodies have at some point to deal with level-of-detail rendering. This is a very mature field of research — a good overview of traditional approaches is given by Pajarola et al. in [26]. While data size and graphics hardware has changed significantly since then, the underlying strategies for level-of-detail management are still valid nowadays. A solid overview of all aspects of planet-scale terrain rendering software is presented by Cozzi and Ring in [8]. While discussing several alternatives, they ultimately propose to use the equirectangular projection for mapping data onto the globe. For Earth this is reasonable, especially since there are many datasets available in this format. However, it suffers from singularities at the poles which lead to triangle degeneration and performance issues. Yet the poles are of specific scientific interest on several extraterrestrial bodies, such as the ice caps on Mars, or the permanently shadowed craters on our Moon. As a solution, Kooima et al. propose a globe decomposition based on icosahedrons [27]. This is however limited by the 32 bit floating point arithmetics on the GPU which lead to artifacts for details smaller than 2.39 m on an Earth-sized globe.

Therefore, CosmoScout VR builds upon the work of Westerteiger et al. [28], using the HEALPix projection [29] which provides a singularity-free subdivision and a uniform data density across the entire globe. Novel aspects of our implementation when compared to the original implementation by Westerteiger et al. are described in Section 5.

3. Architecture

The development of CosmoScout VR started about ten years ago and the software has evolved from a level-of-detail planet-scale renderer [28] to a multi-scale visualization of the entire Solar System and beyond. The software has been made open source and received, for example, contributions in the field of large-data visualization using high-performance computing. In this chapter we first describe potential user groups of CosmoScout VR and deduce requirements and design decisions therefrom. Then we provide a high-level architecture overview and outline several aspects which are then described in greater detail in Section 4.

Target User Groups — The primary user group are *scientists working with large spatial datasets*. Use cases of the software include the interactive analysis of large remote sensing products, in-situ sensor data, and simulation data in its spatial context, for instance for landing site analysis. It can also be used to support space mission planning by immersive visualization of observation geometries and simulation of time-dependent lighting of the planetary environment. A second important user group are *researchers in the fields of visualization, computer graphics, and virtual reality*. Hence, CosmoScout VR is also supposed to be a prototyping-platform which can be used to quickly experiment with new visualization and interaction techniques. Finally, the *general public* benefits from the development of CosmoScout VR, as it can be used for science communication, personal education, and interactive presentations.

Requirements and Design Decisions — Primarily, CosmoScout VR is supposed to be a real-time rendering system putting strong emphasis on the applicability for virtual reality. Consequently, physical correctness will usually be traded for rendering performance. However — whenever possible — parameters will be identified which can be used to adjust this trade-off in one or the other direction. Nevertheless, a constantly high frame-rate is considered more important than physical correctness. Furthermore, users cannot be expected to have much experience with virtual reality. Hence, reducing the impact of cybersickness is an important goal.

On the development-side, CosmoScout VR should be easily extensible to allow scientists to implement visualization modules for specific datasets. This requires a sophisticated plugin-concept and a modular user interface. In addition, plugins should be re-loadable at runtime to allow for fast prototyping. To foster collaboration on this project, CosmoScout VR is designed to be cross-platform, open source, and, whenever possible, established standards should be used to ease the integration of new datasets.

On the hardware side, CosmoScout VR should support both, traditional desktop PCs and a wide variety of virtual reality hardware such as tracking cameras and stereoscopic output devices ranging from HMDs to multi-pipe rendering clusters. Figure 1 shows the application running on a rendering cluster, Figure 2 shows an immersive visualization in a planetarium-like dome environment.

Software Architecture — Figure 3 gives a high-level overview of the involved components: CosmoScout VR is based on external libraries, such as the ViSTA framework [30] for I/O and cluster synchronization, and SPICE [2] for positioning of celestial bodies. SPICE was chosen as it is standard in the space industry. ViSTA was not only chosen for historical reasons, but also because it is completely open source, and thus introduces no license or royalty issues. In contrast to

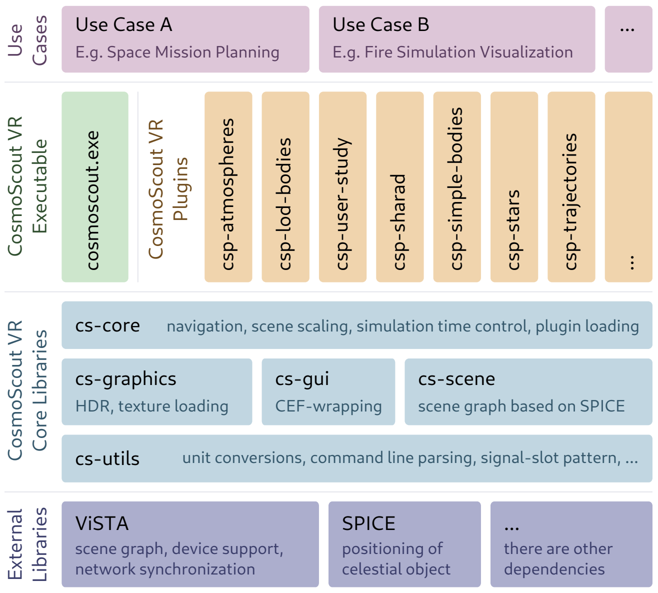
Use Cases
Use Case A
E.g. Space Mission Planning
Use Case B
E.g. Fire Simulation Visualization
...
CosmoScout VR Executable
cosmoscout.exe
CosmoScout VR Plugins
csp-atmospheres
csp-lod-bodies
csp-user-study
csp-sharad
csp-simple-bodies
csp-stars
csp-trajectories
...
CosmoScout VR Core Libraries
cs-core
navigation, scene scaling, simulation time control, plugin loading
cs-graphics
HDR, texture loading
cs-gui
CEF-wrapping
cs-scene
scene graph based on SPICE
cs-utils
unit conversions, command line parsing, signal-slot pattern, ...
External Libraries
ViSTA
scene graph, device support, network synchronization
SPICE
positioning of celestial object
...
there are other dependencies

Figure 3. CosmoScout VR consists of five core libraries which are based on several third-party libraries, such as ViSTA [30] and SPICE [2]. The visualization functionalities are implemented on top in form of plugins.

most other available engines, it is designed from the ground to support rendering on CAVEs, tiled displays, and other clustered setups. Furthermore, during the development of CosmoScout VR, there are oftentimes long-lived parallel development branches for specific research projects. From our experience, using engines such as *Unreal* or *Unity* makes it difficult to maintain multiple versions of the software in parallel using a version control system.

The core functionality of CosmoScout VR is split into five core libraries which are built on top of these external dependencies: `cs-utils` contains frequently used functionality such as conversions between different units, latitudes, or time formats. The SPICE-wrapping for positioning of celestial objects is implemented in `cs-scene`. The classes required to built HTML-based user interface elements are implemented in `cs-gui`. The rendering loop is part of ViSTA, but `cs-graphics` contains specific classes required for rendering 3D objects, shadow computations, and high-dynamic-range (HDR) rendering. Finally, `cs-core` contains high-level functionality using classes of the other libraries, as well as the algorithms for 3D-navigation. Key aspects of these core libraries are described in Section 4.

The aforementioned core libraries do not provide any visualization modules. This is rather implemented in form of plugins which are loaded by the CosmoScout VR executable at runtime. In fact, if CosmoScout VR is started without any plugins, the user will only see a black screen with a minimal user interface. Loading specific visualization functionality from plugins not only reduces development time since it can be reloaded at runtime, but also allows for the integration of novel data visualization methods without affecting core components of CosmoScout VR. Plugins can bring in their own third-party dependencies, use all functionality of the core libraries, and modify the user interface. By design, plugins cannot communicate with each other. This has not been required yet and simplifies the implementation significantly as plugins do not depend upon each other. Some important plugins of CosmoScout VR are described in Section 5.

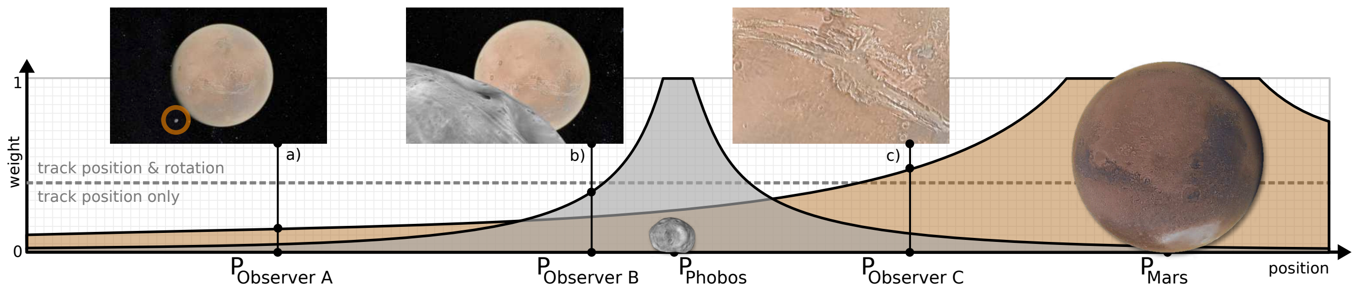

Figure 4. In CosmoScout VR, each frame a weight is calculated for each celestial body. The weight depends on the body's radius and the real-world distance to the user. The body with the highest weight is considered to be the *active* body. Any user movement is relative to the SPICE center of the currently active body. If the weight exceeds a given threshold (marked by the dashed line above), the user will enter the rotation-fixed frame of the active body. This system allows for automatic and seamless transitions between various navigation scenarios: When being far away from a planet, a user wants to follow its revolution around the Sun, but not its rotation (a). Any small objects passing between the planet and the user should not affect the user's movement (such as Phobos, marked with a circle in image a). If the user now moves closer to Phobos, its weight will at some point exceed the weight of Mars (b). Hence, the user will now follow Phobos on its orbit around Mars. Finally, when the user gets very close to the surface of Mars, the assigned weight will exceed the threshold and the user will start following the surface motion (c). This makes it possible to land on the surface of any celestial body while it is actually moving quickly through outer space using free navigation only.

4. Core Components

SPICE-Based Scene Graph (`cs-scene`)

When rendering scenes as large as the Solar System but contain objects in the range of a few centimeters, precision issues arise quickly. In computer graphics, the geometric relations between 3D-objects are typically stored in a scene graph. This is usually a tree in which transformation nodes define a coordinate system relative to their parent node. For example, the poses of individual Lunar rover components may be specified with respect to the rover's local coordinate system which in turn may be expressed relative to the Moon. The Moon's pose may be calculated relative to Earth which is positioned relative to the Sun. During rendering, this tree is traversed top-to-bottom to compute the absolute transformations of all nodes. This results in large translation values which cannot be stored with sufficient precision on today's GPUs which are usually limited to 32 bit operations. The solution described in [10] modifies the usual traversal scheme to start not at the root node but at a transformation node close to the current user position. This results in small user-centered transformation values for close-by objects. However, this required modification is not always possible if an existing 3D engine is used.

Therefore, we propose an alternative approach to position objects with high precision in space which is based on the kernel hierarchies of SPICE [2]. SPICE models geometric relationships of celestial objects with time-dependent, hierarchical *reference frames* in double precision. There are different reference frame types such as inertial reference frames which are fixed with respect to the stars, or body-fixed reference frames which follow the rotation of a celestial body. In CosmoScout VR, special transformation nodes (so-called *CelestialAnchorNodes*) as well as the observer (which corresponds to the user's head) are derived from a *CelestialAnchor* class which stores, in addition to a SPICE frame, a double-precision position, rotation, and scale relative to that frame. For example, the individual components of a Lunar rover can be expressed with single-precision transformation nodes attached to a *CelestialAnchorNode* representing the rover, while the observer is currently positioned relative to the body-fixed reference frame of the Moon using the additional position and rotation properties.

To allow for standard scene graph traversal, *CelestialAnchorNodes* are always children of the root node of the scene graph. Before the traversal, the current relative transformation between each *CelestialAnchorNode* and the observer is computed via SPICE and stored in the single-precision transformation of each *CelestialAnchorNode*. This results in observer-centered coordinates for each scene graph node during rendering. However, special attention has to be paid to large objects such as planets. These need to be decomposed into smaller parts which are transformed to observer-centered coordinates individually.

Due to the additional double-precision position, rotation, and scale, it is also possible to transition a *CelestialAnchor* from one SPICE frame to another without affecting its absolute position in space. This allows for seamless reference frame transitions during navigation.

Seamless Navigation in the Solar System (`cs-core`)

CosmoScout VR has several characteristics making free navigation an especially challenging topic and thus requiring special attention. First, in contrast to most other virtual reality applications, the navigation must fully support all six degrees of freedom as the concept of up and down changes frequently. Second, since the scene is dynamic and celestial bodies move all the time, there is a need for a system which manages the transitions between reference frames automatically. Finally, the Solar System is a multi-scale environment covering many magnitudes of scale. This requires a system which automatically adjusts the users' movement speed so that they can cover the vast distances in outer space as well as maneuver between rock formations on the surface of a celestial body. Such a navigation with support for all six degrees of freedom can easily lead to cybersickness. To mitigate this, we developed a dedicated plugin which is described in more detail in Section 5 and evaluated in Section 6.

When used in virtual reality, CosmoScout VR primarily employs a control scheme similar to the one-handed flying described by [31]. When the user initiates a movement by pulling the hair trigger of the controller, the current pose of the controller is saved. As long as the trigger stays pulled, the observer's movement direction corresponds to the vector from the initial position to the current position. The vector's

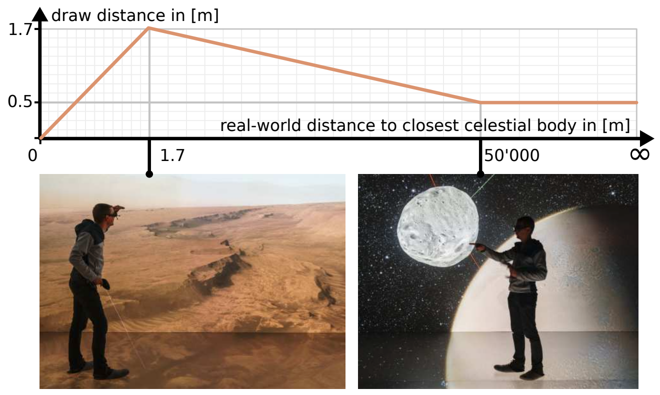

Figure 5. The virtual scene is scaled towards the user's head so that the closest body is drawn at a specific visual distance. This is computed according to the graph above (which can be adjusted in the settings). In this example, the scene is drawn at a 1:1-scale if the user is on the surface. In outer space, the scene is scaled down to draw the celestial body within arm's reach.

magnitude is mapped to the velocity of the observer. At the same time, the observer's rotational movement is determined by the angular difference between the initial and the current pose of the controller. As this is a rate-controlled navigation scheme where the differences are mapped to velocities, the user does not need to twist the controller in uncomfortable positions but can gradually navigate the simulation with precise movements.

In order to seamlessly navigate from the surface of one celestial body to the surface of another, while both are moving through space, the reference frame of the observer has to change at some point. This is done by computing a weight w_{body} for each celestial body at each simulation-time step. This weight depends on the bodies' radius r_{body} and its distance to the observer d.

$$w_{body} = \frac{r_{body}}{r_{body} + max(0, d)} \tag{1}$$

This function yields a weight of one at the surface and a decreasing weight with increasing observer distance. The body with the highest weight is considered to be the *active* body. Whenever the active body changes, the observer will change to an inertial reference frame centered at this body and thus follow its movement automatically. If the weight of the active body exceeds a configured threshold, the observer will change to a body-fixed frame to also follow the bodies' rotation. Note that, because the active body is not necessarily the closest body, this system allows smaller objects such as moons or spacecrafts pass between the observer and the active body without unexpected reference frame changes. An example of this system is provided in Figure 4.

A well-proven approach for navigating huge multi-scale environments is scaling either the user or the scene in such a way, that the closest surface is drawn at a comfortable distance [13]. This allows inspecting details of small objects as well as covering large distances between objects quickly. In CosmoScout VR, this visual distance to the surface of the closest celestial body is computed based on the real-world distance of the observer to that body (Figure 5). If the observer is only a few meters above the surface of a planet, the surface is drawn below the observer's feet to prevent penetration of the virtual content. If the observer is in the orbit of a celestial body, the surface can be drawn closer to allow for inspection and efficient interaction. Consequently, celestial bodies are drawn closer to the observer if they are farther away in real-world. While this seems counter-intuitive at first, it allows for realistic 1:1-rendering when standing on a celestial body's surface, and simultaneously making efficient use of the parallax space when navigating in outer space. This is implemented by scaling the observer so that the closest celestial body is drawn at an appropriate distance. Note that this scaling happens with respect to the *closest* body which is not necessarily the same as the *active* body introduced before. This ensures that the user never penetrates the virtual content.

With this approach, ground following is implicitly implemented. As the observer is automatically scaled to touch the planetary surface, users will always perceive themselves as standing on the surface of the planet. When moving closer to the surface, the surface will not appear to be moving towards the user but will be scaled up instead. As the scaling center is the cyclops position between the user's eyes, the difference is only noticeable due to the not-changing parallax of the planetary surface. See Figure 6 for an example how this special type of ground following works.

HTML-Based User Interface (`cs-gui`)

The *cs-gui* core library contains classes required to build a graphical user interface (GUI) based on web technology such as HTML, CSS, and JavaScript using the Chromium Embedded Framework (CEF[4]). There are several advantages and disadvantages of using such technology in a virtual environment. The key advantage is the rich ecosystem of open source libraries for creating all kinds of GUIs allowing developers to easily create complex, interactive GUI elements such as graphs, timelines, or 2D-maps. Furthermore, the development cycle of the user interface is comparably fast, as changes can be previewed in a web browser without any recompilation of the software. In fact, it is also possible to connect to a running instance of CosmoScout VR and inspect, modify, and debug the GUI elements remotely via web sockets. On the other hand, a lot of complexity is added to the code as the GUI is rendered in a separate process and thus special attention has to be paid to the communication

[4]https://bitbucket.org/chromiumembedded/cef/src

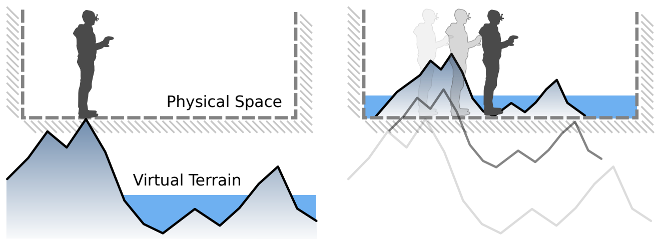

Figure 6. In the vicinity of a celestial body, the scene is scaled so the bodies surface is right below the user's feet (left image). The scaling center is the user's head and the scaling factor is updated each frame. Hence, when the user steps into a valley, the scene will be scaled towards the head until the feet are on the ground again (right image). This scaling is imperceptible with a non-stereoscopic output device. With a stereoscopic display, only the parallax of the terrain changes, not its projected position on the screen.

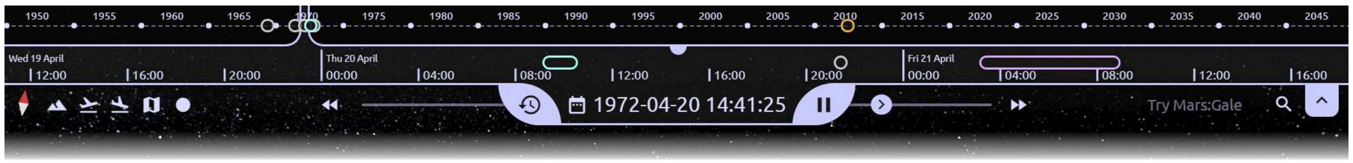

Figure 7. **A central element of CosmoScout's user interface is an interactive timeline. The upper overview-timeline is usually hidden and can be expanded by clicking the arrow-button on the right-hand side. This overview can be dragged and zoomed independently of the main timeline at the bottom. Temporal events are shown with colored circles or bars. The simulation speed can be adjusted with the slider below the timeline.**

with the main thread of CosmoScout VR. Additionally, there is the need to transfer the rendered frames of the user interface to the GPU which causes a bottleneck for large screens.

GUI elements can either be placed in screen space or can be attached to scene graph nodes. The former is used when CosmoScout VR is started in desktop mode, the latter is useful in virtual reality where the GUI is positioned in 3D space relative to the user. Each GUI element runs in a separate process and the CosmoScout VR executable and the plugins communicate with it via inter-process communication. This happens bidirectionally — messages are sent to execute JavaScript code in the context of the GUI element, and there is a set of functions available on the JavaScript side which trigger callbacks in the main thread of CosmoScout VR. A theoretical problem of this system arises in a clustered setup: There is no way to synchronize the state of the GUI elements which means all cluster nodes have their own state of the user interface which is not necessarily equal. Also, the callbacks emitted in the main thread are not guaranteed to arrive in the same frame. As all input sent to the GUI elements is synchronized however, this has never been a problem in practice. Nevertheless, any non-deterministic JavaScript code, such as using random numbers or doing web requests, should be avoided.

The GUI of CosmoScout VR consists of several core components which can be extended by plugins. An expandable sidebar contains most of the configuration possibilities, and at the bottom left, an integrated JavaScript console allows for interactive scripting. Another central component is an interactive timeline, which allows the user to adjust the simulation speed, jump to specific time points, or adjust the current simulation time continuously by dragging the timeline (see Figure 7). Other examples of the GUI can be seen in Figure 1 and Figure 8.

Rendering with Photometric Quantities (`cs-graphics`)

Many use cases of CosmoScout VR — such as training for on-orbit servicing or image synthesis for testing optical navigation algorithms — require the simulation of lighting conditions as realistic as possible. For this, CosmoScout VR includes an optional high-dynamic-range (HDR) rendering pipeline. In order to achieve coherent lighting in a virtual universe, all light sources must be modelled with their correct radiation ratios which vary in our Solar System by several orders of magnitude. Therefore, CosmoScout VR uses photometric units such as *Lumen*, *Lux* and *Candela* throughout the optional high-dynamic-range rendering pipeline.

If HDR rendering is enabled, plugins can query the direction towards the Sun and the Sun-based illuminance for any point in space. With this information, plugins can compute appropriate luminance values for the rendered geometry. The actual shading computation differs from plugin to plugin.

When all scene geometry has been rendered, the average luminance of the current frame is computed by a series of parallel reductions using compute shaders. As CosmoScout VR may run on a cluster of rendering nodes, the local values need to be collected by the cluster master node. The global average luminance values are then distributed to the clients which finally use them to compute an auto-exposure value. This exposure is adapted gradually using the algorithm from [32] to mimic the adaption of human eyes to changes in illumination. Ultimately, a tonemapping operator is applied to reduce the dynamic range before showing the image to the user. Different tonemapping operators can be used, per default CosmoScout VR uses filmic tonemapping [33].

5. Plugins & Applications

As mentioned before, a central aspect of CosmoScout VR's architecture is outsourcing as much functionality as possible to plugins. Plugins can use all previously described functionality from the core libraries but cannot communicate with each other. This ensures that they only contain well-encapsulated functionality. On the down-side, it is sometimes necessary to move functionality which is required by multiple plugins to a core library in order to prevent code duplication.

While there are numerous plugins available in the source code repository of CosmoScout VR [1], we will focus on two of them and their use cases in this chapter: cybersickness reduction and level-of-detail terrain rendering.

Reducing Cybersickness (`csp-vr-accessibility`)

Many use cases of CosmoScout VR are prone to induce cybersickness as they involve free navigation in the Solar System. The means to reduce cybersickness described in Section 2 are implemented in a plugin, so they can be added to hardware setups where they are needed and left out otherwise. Both the grid and the vignette are designed as customizable as possible, as cybersickness is very subjective, and the studies presented earlier suggest a wide range of user preference.

The floor grid (left image of Figure 9) provides a stable frame of reference coinciding with the real world floor. This is done to improve postural stability at the cost of presence and immersion. The grid is implemented as a large quad attached to the origin of the tracking space. This allows the tracked user to move relative to the grid. A vertical offset can be configured so the grid properly matches the real world floor. Additionally, the extent of the grid is customizable to change the size of the grid platform the user is standing on. To allow the user to fully adapt the grid to their needs, the texture of the grid is interchangeable, and its scaling factor can be adapted to determine the coarseness of the grid. Lastly, the grid's color and opacity can be changed as well.

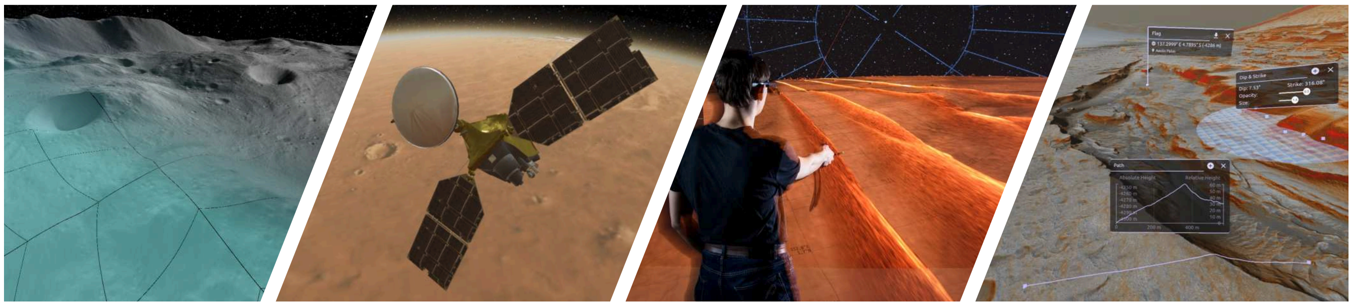

Figure 8. **All visualization capabilities of CosmoScout VR are loaded from plugins, some of which are shown in the images above: The first shows a close-up on the south pole of the dwarf planet Vesta. This can be visualized without problems by the level-of-detail terrain rendering plugin thanks to the HEALPix projection (the terrain tile boundaries are overlaid in turquoise color). Some plugins are rather generic and can be used for various use cases such as rendering of 3D satellites using the GLTF standard (second image). Others are dedicated to specific use cases visualizing special sensor types such as subsurface radar data (third image). The final image shows several terrain-measurement tools which are also implemented as a plugin.**

The field-of-view vignette (right image of Figure 9) reduces peripheral visual flow when the observer is moving. It is realized as a post-processing shader drawing a 2D vignette effect over the rendered scene. The inner and outer radii of the vignette are adjustable. Additionally, there are configurable upper and lower velocity thresholds, setting the limits within which the vignette is active. It is not shown below the lower threshold, and it is fully closed to the inner radius above the upper threshold. The vignette can be toggled between a static and a dynamic mode. In the dynamic mode, the radius of the vignette is adjusted gradually according to the observer speed. The static vignette uses an opening and closing animation when the user passes the velocity thresholds to automatically open or close the vignette. Lastly, the color of the vignette is customizable, and an additional toggle exists to switch between a circular vignette and a vertical vignette. The vertical vignette mimics the natural human vision by not limiting the horizontal field of view. In addition, the vertical limitation is asymmetrical as humans have a larger field of view towards the floor [25].

In Section 6, a pilot user study is presented to evaluate the implemented means for cybersickness mitigation. The main goals were to assess their effectiveness, to find optimal default settings, and to guide future development.

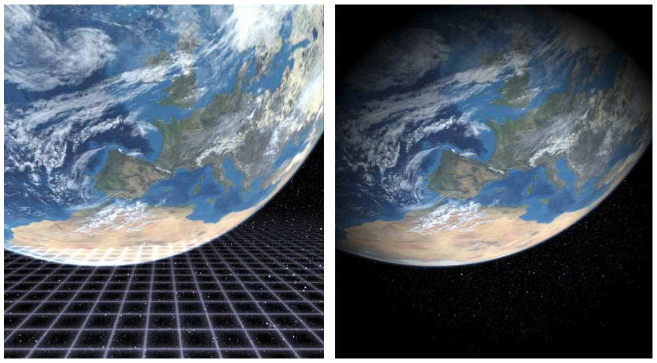

Figure 9. **In order to reduce cybersickness, CosmoScout VR includes a dedicated VR-accessibility plugin. This offers a transparent floor grid to provide a stable reference frame with respect to the real world (left image) and a configurable vignette to limit one's field of view during rapid movements (right image).**

Level-of-Detail Terrain Rendering (`csp-lod-bodies`)

This plugin contains a key feature of CosmoScout VR: Precise rendering of planet-scale terrain datasets (elevation and imagery) without singularities at the poles. This is interesting for extraterrestrial bodies like Mars or the Moon where the poles are of high scientific interest. In the first image of Figure 8, the south pole of the asteroid Vesta is rendered.

The approach is based on previous work by Westerteiger et al. [28]. Planets and moons are subdivided into twelve base tiles according to the HEALPix [29] projection (Figure 10). They represent the root nodes of twelve quad-trees in which each node covers the same spatial extend as its child nodes but with half the resolution. During rendering, nodes are recursively subdivided until they meet a specific split criterion. The current implementation uses an upper limit for the projected screen space size of the tile's bounding box as split criterion. This limit is adjusted automatically to ensure that the frame rate stays above the screen's refresh rate. While this is a very basic criterion, we found that it works well in practice and requires no preprocessing such as computing geometric differences between quad-tree levels. This is important, as it allows us to extend the original implementation by Westerteiger et al. to use the Web Map Service (WMS) protocol standardized by the Open Geospatial Consortium (OGC) for retrieval of elevation and imagery data.

For each tile, a 256x256 pixel image is requested via HTTP from a map server following the WMS standard. This request includes information on the required datasets, the spatial extent, and the target image format (e.g. 24 bit RGB for imagery and 32 bit grey scale for elevation data). The raw data is then loaded by the map server and reprojected to HEALPix on-the-fly. To add support for this projection to the map server, some changes to the HEALPix projection of the underlying PROJ library[5] were necessary and are included therein since version 6.3.0. The changes include the possibility to rotate the HEALPix space by 45°: This way, all tiles become axis-aligned and can therefore retrieved as square-shaped images (right image of Figure 10). Finally, the images are streamed to the application and cached locally. This approach requires nearly no preprocessing of the raw datasets (apart from standard optimizations to enable fast data loading by the map server), and thus new datasets can be integrated easily.

[5]https://proj.org/

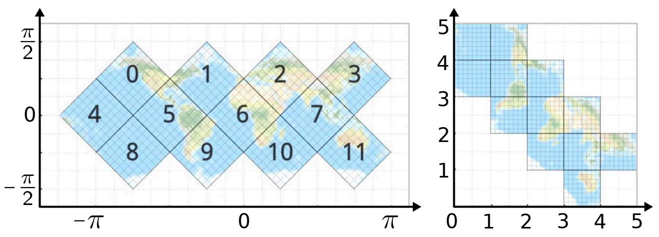

Figure 10. The original HEALPix implementation projects geographical data to the twelve base tiles shown in the left chart. To store the tiles as images, the projection is modified so that the projected image is rotated by 45°. An additional scaling and offset aligns the base patch bounds with integer values (right chart).

During rendering, the same fixed square grid of vertices is drawn for each tile. This grid consists of 257 vertices on each edge (that is 256 vertices to match the tiles data resolution plus one for stitching with the neighboring patches). All vertices are then vertically displaced along the geodetic normal according to the elevation data and textured with the image data. The required elevation and imagery data for all currently visible tiles is stored in two texture arrays on the GPU. Consequently, each tile has access to all visible data when being drawn. This allows us to extend the original implementation with terrain stitching to prevent visible seams in the terrain due to t-junctions where different tile resolutions meet: For edges between two tiles with different resolution, elevation data is always sampled from the lower resolution neighbor. When the resolution of both tiles is the same, data is always sampled from the western neighbor.

If vertex positions are transformed from geographic to Cartesian coordinates on the GPU, jittering artifacts due to insufficient floating point precision arise when the camera is close to the virtual surface of a large celestial body. This is solved similarly as described in [7]: If the virtual camera is so close to a terrain tile that jittering would be noticeable, the four corners of the tile are transformed to double-precision camera-centric coordinates on the CPU, and linearly interpolated on the GPU using single-precision arithmetics. This means, that the surface of the ellipsoidal globe is at some point approximated by a series of piecewise linear patches. However, the resulting error is small: The center altitude of linearly interpolated 1x1 km tiles differs from the real ellipsoidal surface by about 1 cm for an Earth-sized globe.

An exemplary use case for this plugin is *Landing Site Evaluation*. It was actually the initial goal of CosmoScout VR to support planetary researchers in the evaluation of potential landing sites for future space exploration missions. To this end, significant development happened during the EU-funded project CROSSDRIVE [34] which was about collaborative rover mission planning. During this project, the `csp-lod-bodies` plugin was extended to support advanced terrain visualizations such as slope shading. Dedicated data visualization plugins were added, for example visualizing subsurface radar data (third image of Figure 8). In addition, virtual tools were developed which allow geologists to measure profile lines, dip and strike angles, landing ellipses, surface areas, or volumes (last image of Figure 8). In order to contextualize the data, spacecrafts are positioned based on SPICE data with `csp-satellites` (second image of Figure 8). Their trajectories through space are also drawn by a dedicated plugin: `csp-trajectories` (Figure 11). To complete the virtual environment, stars are loaded from the Tycho, Tycho2 and the Hipparcos catalogues by `csp-stars` and atmospheres around celestial bodies are visualized by `csp-atmospheres` using single Mie- and Rayleigh scattering [35].

A completely different use case is the *simulation of onboard cameras for testing terrain-relative navigation algorithms*. While CosmoScout VR primarily aims at interactive analysis of remote sensing products in conjunction with other large-scale datasets, we are investigating the possibility to use it for emulating a camera on board of a Lunar lander. The goal is to create artificial images of the Lunar surface which can then be used in closed-loop simulations for testing autonomous landing algorithms. Therefore, CosmoScout VR must generate as realistic images as possible in real-time (see Figure 12). For this project, the `csp-lod-bodies` plugin was extended to support the Oren-Nayar [36] and the Hapke [37] model reflectance model for shading the Lunar surface. For integration into the simulation loop, CosmoScout VR requires an interface which on the one hand allows remote-controlling the application state (e.g. changing the camera position and attitude, the simulation date, or the Sun direction) and on the other hand streaming of rendered frames to a third-party application. To enable this, we added a plugin called `csp-web-api`, which allows remote execution of JavaScript code via an HTTP interface. This allows to modifying the application state from a third-party application such as a script or a dedicated control interface. In addition, it allows capturing of color and depth images and streaming them via HTTP to a third-party application.

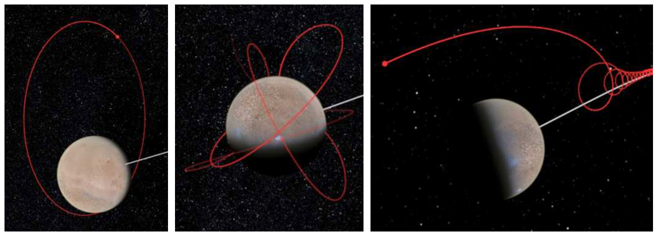

Figure 11. These images show the trajectory of Mars Express at the same simulation time but with different reference frames. The inertial J2000 frame centered around the parent body is useful when visualizing orbits (left). For ground tracks, a body-fixed non-inertial frame centered around the parent body (center) is useful. For interplanetary cruise, an inertial frame centered at the Solar System's barycenter can be used (right).

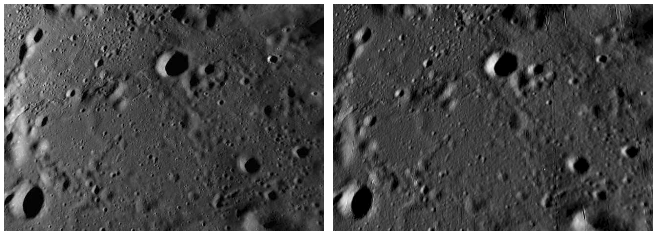

Figure 12. The left image is a cropped and rectified photograph taken by the SpaceIL Beresheet Lander. The camera's pose and intrinsic parameters were obtained from a single image, using crater landmarks detected by [38]. The right image is a corresponding image rendered with CosmoScout VR using LOLA elevation data. While the geometric similarities are apparent, quantitative image comparisons are future work.

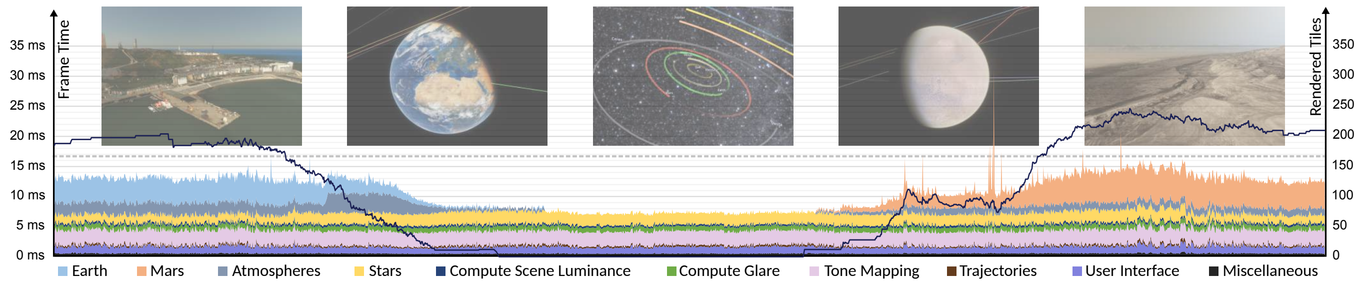

Figure 13. The graph shows exemplary GPU timings for a journey through the Solar System over the course of about 2000 frames. At the beginning, the camera is stationary on the surface of Earth, then it travels backwards to show the entire planet and later an overview of the Solar System. It moves towards Mars and finally lands in Gale Crater. The stacked area chart shows the contribution of various components to the overall frame time. The thin line indicates the amount of terrain tiles rendered by the `csp-lod-bodies` plugin for Earth and Mars respectively. The measurements were performed on a laptop equipped with a NVidia Quadro P3200, an Intel Core i9-8950 HK, and a full HD screen.

6. Evaluation

In this chapter, we evaluate the performance of the aforementioned plugins as well as present a pilot user study to assess the effectiveness of the cybersickness mitigation methods.

Performance

In order to illustrate the rendering performance of CosmoScout VR and its plugins, Figure 13 shows representative GPU timings when run on a contemporary laptop. The graph shows GPU timings only, as CosmoScout VR is GPU-bound on most hardware setups. In addition to the `csp-lod-bodies` plugin, other plugins for rendering stars, atmospheres, and trajectories were enabled. Most of the frames stayed well below the dotted line of 16.67 ms, only a few dropped when Mars came into view quickly. This is expected, as the level of detail of the planets is automatically adjusted to keep the frame time below this value. When approaching the surface of Mars, the number of rendered tiles peaked at about 250 (which corresponds to about 32.8 million triangles) but was automatically reduced to ascertain a stable frame rate of 60 frames per second. The ”Compute Scene Luminance”, ”Compute Glare”, and ”Tone Mapping” steps are only required if the HDR rendering path is enabled. This means, if HDR is disabled, the level of detail will automatically increase using the gained frame time.

A potential performance bottleneck of CosmoScout VR's GUI is the data transfer from main memory to GPU memory. However, updated regions of the user interface are transferred from the GUI processes to GPU memory using persistently mapped buffers of OpenGL 4.4, which proved to be very fast. For the 2000 frames shown in Figure 13, the average CPU time required for communicating with the GUI processes and transferring the image data was $\bar{x}_{CPU} = 1.237\,\text{ms}$ with a standard deviation of $\sigma_{CPU} = 0.165\,\text{ms}$. The same part of the code required only $\bar{x}_{GPU} = 0.779\,\text{ms}$ / $\sigma_{GPU} = 0.138\,\text{ms}$ on the GPU.

User Study

A pilot user study was conducted to measure the effectiveness of the implemented cybersickness-mitigation methods and to identify potential for future improvements. In the within-subject study, we use the Fast Motion Sickness Scale (FMS) instead of the historically popular Simulator Sickness Questionnaire (SSQ), since the single item FMS allows discrete sampling of cybersickness symptom severity during the exposure, instead of a single post-exposure questionnaire like the SSQ. In addition to the subjective FMS scores, center of gravity measurements, similar to the head dispersion measured by Lim et al. [25], are used to collect the subject's postural stability at discrete points during the exposure. Lastly, questionnaires are used to assess user satisfaction on a seven-point Likert scale.

Procedure — The pilot user study starts with an introduction, where data about experience with motion sickness and virtual reality is collected. Afterwards, each subject performs a short tutorial to get used to the virtual environment. In the main part, each subject navigates through three scenarios, each time with either the vignette, the grid, or no feature enabled. The order of the scenarios is counterbalanced to compensate for confounding factors like increasing acclimatization to the virtual environment and competency with the control scheme. In each scenario, the user has to follow the same path through a series of checkpoints, some of which measure the subject's postural stability while others ask for an FMS rating (see Figure 14). Each scenario pass is followed by a five-minute break from the virtual environment and a questionnaire to assess each subject's satisfaction. Finally, a questionnaire for comparison of the vignette feature, the grid feature, and the featureless version is presented.

Participants — The pilot study was conducted with 21 participants (15 male, 6 female), the average age was 32 (± 7) years. All subjects indicated a low incidence or rare cases of motion sickness from other sources (i.e. sea/car sickness), except for motion sickness originating from simulators which was reported more often, but may be caused by more provocative stimuli. On average, the subjects reported low experience with virtual reality (between less than once a month down to less than once a year or never). Out of the 21 subjects, 4 decided to abort the pilot study due to feeling unwell, 3 of them aborted during the first scenario, 1 aborted during the second scenario (2 during the scenario with the grid, and one each during the vignette and featureless scenario).

Subjective Results — We use the Friedman Test [39] to find significant differences between the three scenarios in the answers for the comparative questionnaire. The results are shown in Figure 15. The Friedman Test reports significant differences for cybersickness symptoms (Question 3, $Q = 8.05$, $p = 0.018$), with a medium to large effect size (Cohen's $d = 0.75$) between the featureless scenario (w/o) and the vignette scenario (V). Additionally, there is a significant difference in enjoyment (Question 1, $Q = 8.0$, $p = 0.018$), with a medium effect size (Cohen's $d = 0.51$) between the grid scenario (G) and the featureless scenario (w/o).

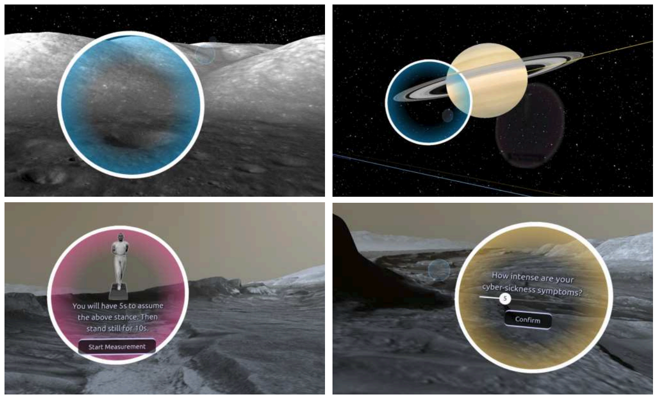
You will have 5s to assume the above stance. Then stand still for 10s
Start Measurement
How intense are your cyber-sickness symptoms?
Confirm

Figure 14. Subjects had to navigate through a series of checkpoints, starting on the Moon (top left), moving through space (top right) and ending on the Martian surface. At about one-minute intervals, checkpoints measured the user's center of gravity (bottom left) and requested an FMS rating (bottom right).

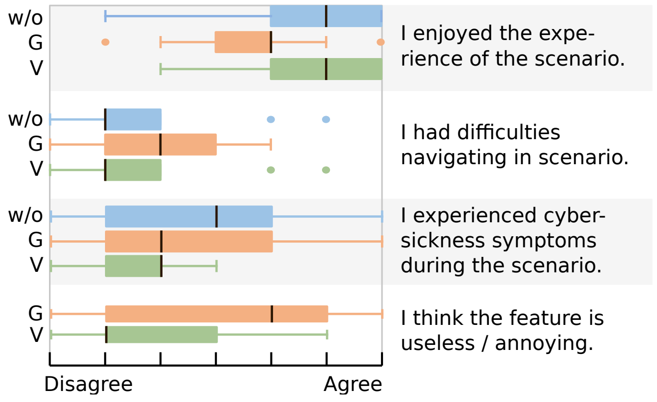

Figure 15. The qualitative feedback showed that users preferred the vignette (V) over the grid (G). Opposed to the grid, the vignette does not seem to influence the user experience negatively when compared to the scenario without any aid (w/o). In addition, users reported a reduced amount of cybersickness symptoms.

Objective Results — The Fast Motion Sickness Scale (FMS) and balance measurements (COG) indicate similar findings, however, the differences between scenarios are not significant. This may be a result of difficulties to precisely measure the center of gravity, as we reduced the 30 seconds balance measurement proposed by Chardonnet et al. in [24] down to 10 seconds to avoid the subject getting impatient during the measurements, as they are done more frequently. The FMS measurements may not show significant differences due to the strong influence of other factors like time spent inside the virtual environment and the 5-minute break being insufficient to regulate the overall accumulation of cybersickness.

Discussion — The outcome shows that the vignette provides best results, reducing cybersickness symptoms noticeably. Furthermore, it was not obstructing or annoying for users less susceptible to cybersickness and thus not in need of the feature. This flexibility suggests the vignette can be turned on in default settings for all users, albeit with a higher inner radius, reducing its notability without sacrificing its effectiveness, in response to feedback from several subjects.

The grid received mixed results, with both positive, but also negative feedback from subjects regarding its effectiveness and obstructiveness. Subjects less susceptible to cybersickness tend to find the grid occluding their view or annoying. On the other side, subjects who experienced cybersickness symptoms seem to be split into two groups: Subjects who felt comfortable with the controls mostly reported the grid to be helpful, providing a stable floor and a reference to help align the virtual to the real world floor. Subjects who appeared less confident with the controls showed difficulties adjusting the observer's orientation, resulting in a slanted virtual horizon which was not aligned with the grid. Those users often reported that the grid felt unaligned with the real world floor and that this perceived misalignment resulted in increased cybersickness symptoms and reduced postural stability. Rolling movements and an angular mismatch between the virtual horizon and the floor grid appear to be the most common cause for cybersickness inside the simulation. Additionally, several users reported problems or negative feedback when the grid is displayed while moving.

In conclusion, we found the vignette to be a robust and flexible method of cybersickness mitigation when 6-degrees-of-freedom navigation cannot be avoided or limited. The grid, however, appears to be highly subjective and useful only in specific situations. Therefore, the grid should be used based on personal preference. We gathered additional feedback and information to further improve and develop the grid, like an option to automatically hide the grid while moving, or a method to automatically level the horizon when it is close to the real-world horizon. Finally, an additional, alternative navigation scheme like a world-in-miniature overview with teleportation navigation may also be beneficial to the user experience and cybersickness.

7. Conclusion & Future Work

In this paper, we presented the architecture of CosmoScout VR, as well as several core components of the framework. We have shown the modularity of the open source software which fosters its application in various scientific domains. It can be used to gain insight into large space mission datasets through contextualization and explorative analysis in an immersive virtual environment. This approach is not only valuable for data analysis, but can also be used to effectively communicate scientific findings to colleagues and the general public.

While the developed framework has already proven to be valuable and reliable in practice, there is much potential for future work. For instance, the user study provided valuable insights on how we can further improve the user experience in virtual reality. The development of additional domain-specific plugins is another promising aspect and we are currently collaborating in various projects to visualize large-scale datasets such as mantle convection data, oceanic flow data, or wildfire simulation data. Furthermore, a quantitative comparison of generated images with corresponding photographs will be conducted in the future. There is also a lot of potential for performance improvements, especially in the HDR rendering path.

Overall, CosmoScout VR has matured significantly over the years and as it became more stable it will serve as a solid basis for a lot of upcoming research regarding novel scientific data visualization techniques, new navigation and interaction methods, and efficient computer graphics algorithms.

Acknowledgments

This work is partially funded and supported by the European Commission grant H2020-FETHPC-2017 ”VESTEC” (ref. 800904).

References

[1] S. Schneegans, M. Flatken, and A. Gerndt, “CosmoScout VR 1.4.0,” March 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4646924

[2] C. H. Acton, “Ancillary data services of NASA’s Navigation and Ancillary Information Facility,” *Planetary and Space Science*, vol. 44, no. 1, pp. 65 – 70, 1996.

[3] B. Semenov, “WebGeocalc and Cosmographia: Modern tools to access OPS SPICE data,” in *2018 SpaceOps Conference*, 2018, p. 2366.

[4] K. Hussey, “”Eyes On The Solar System” a real-time, 3D-interactive experience for planetaria and beyond,” in *European Planetary Science Congress*, vol. 1, 2010, p. 50.

[5] P. Cozzi and D. Bagnell, “A WebGL globe rendering pipeline,” *GPU Pro*, vol. 4, pp. 39–48, 2013.

[6] F. Pirotti, M. A. Brovelli, G. Prestifilippo, G. Zamboni, C. E. Kilsedar, M. Piragnolo, and P. Hogan, “An open source virtual globe rendering engine for 3D applications: NASA world wind,” *Open Geospatial Data, Software and Standards*, vol. 2, no. 1, pp. 1–14, 2017.

[7] K. Bladin, E. Axelsson, E. Broberg, C. Emmart, P. Ljung, A. Bock, and A. Ynnerman, “Globe browsing: Contextualized spatio-temporal planetary surface visualization,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 802–811, 2017.

[8] P. Cozzi and K. Ring, *3D engine design for virtual globes*. CRC Press, 2011.

[9] C.-W. Fu and A. J. Hanson, “A transparently scalable visualization architecture for exploring the universe,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 1, pp. 108–121, 2006.

[10] E. Axelsson, J. Costa, C. Silva, C. Emmart, A. Bock, and A. Ynnerman, “Dynamic scene graph: Enabling scaling, positioning, and navigation in the universe,” in *Computer Graphics Forum*, vol. 36. Wiley Online Library, 2017, pp. 459–468.

[11] F. Argelaguet and M. Maignant, “GiAnt: stereoscopic-compliant multi-scale navigation in ves,” in *Proceedings of the 22nd acm conference on virtual reality software and technology*, 2016, pp. 269–277.

[12] C.-W. Fu, A. J. Hanson, and E. A. Wernert, “Navigation techniques for large-scale astronomical exploration,” in *Visualization and Data Analysis 2006*, R. F. Erbacher, J. C. Roberts, M. T. Gröhn, and K. Börner, Eds., vol. 6060, International Society for Optics and Photonics. SPIE, 2006, pp. 179 – 188.

[13] J. McCrae, I. Mordatch, M. Glueck, and A. Khan, “Multiscale 3D navigation,” in *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, 2009, pp. 7–14.

[14] L. Rebenitsch and C. Owen, “Review on cybersickness in applications and visual displays,” in *Virtual Reality*, 2016, pp. 101–125.

[15] G. E. Riccio and T. A. Stoffregen, “An ecological theory of motion sickness and postural instability,” *Ecological Psychology*, pp. 195–240, 1991.

[16] E. Chang, H. InJae, H. Jeon, Y. Chun, K. H. Taek, and C. Park, “Effects of rest frames on cybersickness and oscillatory brain activity,” in *2013 International Winter Workshop on Brain-Computer Interface (BCI)*, 2 2013, pp. 62–64.

[17] H. B. L. Duh, D. E. Parker, and T. A. Furness, “An ”independent visual background” reduced balance disturbance evoked by visual scene motion: Implication for alleviating simulator sickness,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 01 2001, pp. 85–89.

[18] G. V. Barrett and C. L. Thornton, “Relationship between perceptual style and simulator sickness.” *Journal of Applied Psychology*, pp. 304–308, 1968.

[19] B. Keshavarz, A. E. Philipp-Muller, W. Hemmerich, B. E. Riecke, and J. L. Campos, “The effect of visual motion stimulus characteristics on vection and visually induced motion sickness,” *Displays*, pp. 71–81, 2019.

[20] A. S. Fernandes and S. K. Feiner, “Combating VR sickness through subtle dynamic field-of-view modification,” in *2016 IEEE Symposium on 3D User Interfaces (3DUI)*, 2016, pp. 201–210.

[21] M. E. McCauley and T. J. Sharkey, “Cybersickness: Perception of self-motion in virtual environments,” in *Presence: Virtual and Augmented Reality*, 1992, pp. 311–318.

[22] B. Keshavarz and H. Hecht, “Validating an efficient method to quantify motion sickness,” *Human Factors*, pp. 415–426, 2011.

[23] R. S. Kennedy, N. E. Lane, K. S. Berbaum, and M. G. Lilienthal, “Simulator sickness questionnaire: An enhanced method for quantifying simulator sickness,” *The International Journal of Aviation Psychology*, pp. 203–220, 1993.

[24] J.-R. Chardonnet, M. A. Mirzaei, and F. Merienne, “Visually induced motion sickness estimation and prediction in virtual reality using frequency components analysis of postural sway signal,” in *International Conference on Artificial Reality and Telexistence Eurographics Symposium on Virtual Environments*, Kyoto, Japan, 10 2015, pp. 9–16.

[25] K. Lim, J. Lee, K. Won, N. Kala, and T. Lee, “A novel method for VR sickness reduction based on dynamic field of view processing,” *Virtual Reality*, 7 2020.

[26] R. Pajarola and E. Gobbetti, “Survey of semi-regular multiresolution models for interactive terrain rendering,” *The Visual Computer*, vol. 23, no. 8, pp. 583–605, 2007.

[27] R. Kooima, J. Leigh, A. Johnson, D. Roberts, M. SubbaRao, and T. A. DeFanti, “Planetary-scale terrain composition,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 5, pp. 719–733, 2009.

[28] R. Westerteiger, A. Gerndt, and B. Hamann, “Spherical terrain rendering using the hierarchical HEALPix grid,” in *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering-Proceedings of IRTG 1131 Workshop 2011*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[29] K. M. Gorski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartel-

mann, “HEALPix: A framework for high-resolution discretization and fast analysis of data distributed on the sphere,” *The Astrophysical Journal*, vol. 622, no. 2, p. 759, 2005.

[30] I. Assenmacher and T. Kuhlen, “The ViSTA virtual reality toolkit,” *Proceedings of the IEEE VR SEARIS*, pp. 23–26, 2008.

[31] A. Drogemuller, A. Cunningham, J. Walsh, M. Cordeil, W. Ross, and B. Thomas, “Evaluating navigation techniques for 3D graph visualizations in virtual reality,” in *2018 International Symposium on Big Data Visual and Immersive Analytics (BDVA)*. IEEE, 2018, pp. 1–10.

[32] S. N. Pattanaik, J. Tumblin, H. Yee, and D. P. Greenberg, “Time-dependent visual adaptation for fast realistic image display,” in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, 2000, p. 47–54.

[33] H.-P. Duiker, “Filmic tonemapping for real-time rendering,” *Proceedings of ACM SIGGRAPH Courses. ACM*, 2010.

[34] A. S. García, T. Fernando, D. J. Roberts, C. Bar, M. Cencetti, W. Engelke, and A. Gerndt, “Collaborative virtual reality platform for visualizing space data and mission planning,” *Multimedia Tools and Applications*, vol. 78, no. 23, pp. 33 191–33 220, 2019.

[35] P. Collienne, R. Wolff, A. Gerndt, and T. Kuhlen, “Physically based rendering of the martian atmosphere,” in *Workshop der GI-Fachgruppe VR/AR*. Shaker Verlag, Aachen, 2013, pp. 97–108.

[36] M. Oren and S. K. Nayar, “Generalization of lambert’s reflectance model,” in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 1994, pp. 239–246.

[37] B. Hapke, *Theory of reflectance and emittance spectroscopy*. Cambridge University Press, 2012.

[38] B. Maass, S. Woicke, W. M. Oliveira, B. Razgus, and H. Krüger, “Crater navigation system for autonomous precision landing on the moon,” *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 8, pp. 1414–1431, 2020.

[39] M. Friedman, “The use of ranks to avoid the assumption of normality implicit in the analysis of variance,” *Journal of the american statistical association*, vol. 32, no. 200, pp. 675–701, 1937.

Biography

Simon Schneegans *received his M.Sc. degree in Computer Science and Media from Bauhaus University Weimar. Since 2016, he is employed as research scientist at the German Aerospace Center in the Institute for Software Technology. His research domain is real-time computer graphics with a focus on natural phenomena like atmospheric scattering or photorealistic terrain rendering.*

Moritz Zeumer *received his M.Sc. degree in applied Computer Science in 2021 from the University of Applied Sciences and Arts Hanover. Since then, he is employed as a research scientist at the German Aerospace Center in the Institute for Software Technology. His research domain is visualization and human-computer-interaction with a focus on cybersickness in vr-applications.*

Jonas Gilg *received his M.Sc. degree in Applied Computer Sciences from the University of Applied Sciences and Arts, Hanover in 2019. Since then, he is employed as a research scientist at the German Aerospace Center in the Institute for Software Technology. His research domain is visual analytics with a focus on georeferenced data.*

Prof. Dr. Andreas Gerndt *received his degree in computer science from Technical University, Darmstadt, Germany in 1993. In the position of a research scientist, he also worked at the Fraunhofer Institute for Computer Graphics (IGD) in Germany. Thereafter, he was a software engineer for several companies with focus on Software Engineering and Computer Graphics. In 1999 he continued his studies in Virtual Reality and Scientific Visualization at RWTH Aachen University, Germany, where he received his doctoral degree in computer science. After two years of interdisciplinary research activities as a post-doctoral fellow at the University of Louisiana, Lafayette, USA, he returned to Germany in 2008 to head a department at the German Aerospace Center (DLR). Since 2019, he is also Professor in High-Performance Visualization at University of Bremen, Germany.*