DLR Visualization 2 Library - Real-Time Graphical Environments for Virtual Commissioning

Sebastian Kümper[1] Matthias Hellerer[1] Tobias Bellmann[1]

[1]Institute of System Dynamics and Control, German Aerospace Center (DLR),
{sebastian.kuemper, matthias.hellerer, tobias.bellmann}@dlr.de

Abstract

In this paper, the next generation of model-based visualization is introduced, the DLR Visualization 2 Library. This new real-time graphics environment for Modelica is equipped with a state of the art engine for physics based lighting calculation and high-definition render quality, simultaneous visualization of parallel running simulation models, new features like a modern streaming interface and a new, cleaner library structure. It enables the user to create graphical real-time environments for virtual commissioning of complex systems of systems and imaging based sensors. Some applications, as for example depth-camera data generation or rendering of point clouds or vectorized flow visualization are demonstrated in the use cases section of this paper.

Keywords: Visualization, Virtual Commissioning, Systems of Systems, Multi-Body

1 Introduction

The Modelica multi-body library allows for highly detailed simulations of mechanical structures. However, the integrated visualization for multi-body components available in all major Modelica tools contains only some simple geometrical forms, surfaces or static CAD models (Otter, Elmqvist, and Mattsson 2003). While this is sufficient for many engineering tasks, where only the structure and behavior of a system is of interest for the engineer, for virtual commissioning, a realistic graphical environment is sometimes necessary, especially if optical sensors should be used as part of an image processing pipeline (e. g. for homing a robot using camera data). Furthermore, more and more simulations, particularly those of "Systems of Systems", have to run in parallel to increase performance, creating the problem of asynchronous creation of model animation data to be processed by the render engine of the visualization tool.

To overcome these shortcomings, DLR developed the new DLR Visualization 2 library, an evolution and successor to the long available and continuously refined DLR Visualization Library from 2009 (Bellmann 2009; Hellerer, Bellmann, and Schlegel 2014).

Figure 1. Recumbent bike visualized in the DLR Visualization 2 Library. Material properties of the CAD models as metalness or roughness are rendered in a physically meaningful way.

1.1 Modelica Visualization - State of the Art

A comprehensive review of alternative visualization methods for Modelica can be found in Hellerer, Bellmann, and Schlegel (2014). Since then some new work has been published, especially from Waurich and Weber (2017). Here, FMUs of the model are generated and integrated in the Unity engine within a newly developed Unity plugin. The FMU serves as data source for the visualizer elements (e. g. for their positioning) to enable high quality rendering with a state of the art engine. In Fuchs, Streblow, and Müller (2015), Python is used for the visualization of mass flows of thermal-fluid networks.

1.2 Virtual Commissioning

Within the life cycle of a product or plant, commissioning refers to the phase, where a new technical system is activated and used productively for the first time as a whole. However, in complex multi-component systems, this step oftentimes ends with problems, error messages and incompatibilities between the different systems and the desired task cannot be carried out as expected because of unforeseen differences between specification and reality. In order to avoid such problems and potential high costs caused by subsequent error solutions and eventual hardware/software redesigns, virtual commis-

sioning strives to commission the single system components before the completion/production of all other hardware components, by coupling them with a virtual environment, which simulates the yet unavailable components (VDI/VDE-Fachbereich Engineering und Betrieb 2016). Modelica is a useful tool here, as it enables the user to simulate yet to be produced or to be designed hardware components. One example would be the real-time connection of simulated hardware with the real world controller software/hardware of the system. However, if the controller relies on optical sensory input, for example as part of an image processing tool-chain, a real-time visualization with virtual sensors and camera streams becomes a necessity.

2 New Features of the DLR Visualization 2 Library

The DLR Visualization 2 Library is a completely new development. We integrated most of the features from the previous version and we also improved the interface and added new features. This improves the quality of created images and allows the library to be used for more use cases which are also useful for virtual commissioning. The structure of the new library can be seen in Figure 2.

2.1 Improved Rendering Quality

The DLR Visualization 2 Library uses a completely new and modern rendering backend which is based on the real-time 3D engine Unigine (Unigine 2021). This allows for a greatly increased rendering quality. An example image can be seen in Figure 1.

One core change is the material rendering. Previously, the materials were parameterized by abstract values that were not based on real world properties. In the new version the parameters are based on the physical appearance of real world objects and are consequently more intuitive to the user (Greenberg et al. 1997). Every object has three properties: color, metalness and roughness. The color defines the dominant color of the object. Metalness is a boolean value which defines whether the object is made of metal or not. Roughness is a value between 0 and 1 which defines how rough the rendered object should be. A value of 0 indicates that the object is perfectly polished and light gets perfectly reflected. A value of 1 indicates that the object is very rough in a microscopic sense. An example for varying material properties can be seen in Figure 3. These properties can either be set for simple objects, overridden for file objects or directly imported from a gltf file (Khronos 2021). Other CAD-File types are supported but the resulting material may not be the intended material.

Another big improvement is the scene lighting. The light sources have a more realistic look and they produce dynamic high quality shadows. The shadows can also be disabled per light source or per object to give the user more control over the result. The brightness is set via a physical value in lux. The lighting also includes ambient occlusion which enhances cavities by simulating small distance shadows.

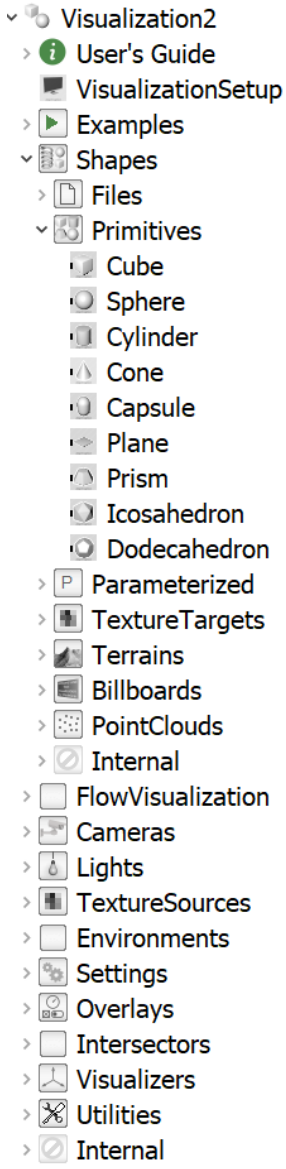

Figure 2. The new Visualization 2 library structure with separate primitives for better visibility on modeling level

2.2 Depth Rendering and Point Clouds

In the field of automation and robotics, sensor systems that create depth information or spatial data are common. Stereo cameras may create depth images, i. e. images where each pixel represents the distance to the viewed object. Laser scanners or sonars may create a list of points that approximate the scanned objects. We added several options to visualize this data and also added the virtual camera `DepthCamera`, to create depth image data in a virtual environment, so that these types of sensors can be simulated.

It is possible to directly visualize the depth field created by the `DepthCamera` within the visualization window. The distance of the camera to the viewed object is then encoded in the color of the camera image pixels. For an example, see the use case in section subsection 3.1. For a better visualization of small distance

DOI 10.3384/ecp21181197

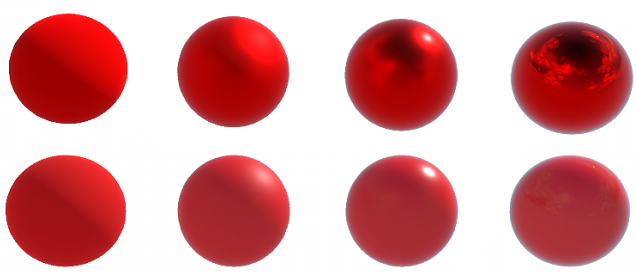

Figure 3. Example of material properties of a sphere. Top row: metalness set to true, Bottom row: metalness set to false. Decreasing roughness from left to right.

differences, the user can enable Eye-Dome-Lighting (Ribes and Boucheny 2011) by applying a dummy shading technique. The data can also be sent to other applications as described in subsection 2.3.

Depth image data can also be visualized in 3D with `DepthMesh`. This is a plane with a given resolution that is deformed according to the depth image. Together with the position and orientation of the camera, the object viewed by the camera can be accurately reconstructed in 3D.

A more versatile solution are point clouds. Here individual points are visualized via small squares. A point cloud can also be generated from a `DepthCamera` with the `CameraPointCloud`. This delivers nearly the same result as the `DepthMesh`, but each pixel of the depth image will correspond to an individual point instead of a closed mesh.

In addition to `CameraPointCloud`, there exists `StreamPointCloud` where the points are received from a network source, `RawPointCloud` where the points can be set directly through Modelica and `FilePointCloud` where the points are loaded from a .xyz-file. The data of multiple time points of point clouds can be combined into a grid, so that the complete scene can be viewed instead of a single time point. The user also has the possibility to filter the data by the quality data from the sensor and color the point clouds accordingly.

An example application can be seen in Figure 4. Here a remote operated underwater vehicle (ROV) is simulated. It has a panning depth camera attached to its front to view the surroundings. On the top left image, the ROV with the landscape is shown. The current view of the depth camera is shown with a `DepthMesh` in red. On the top right is the colored depth image of the camera with enabled eye-dome-lighting. On the bottom left is a `StreamPointCloud` which receives the points from the camera via network. On the bottom right is another `StreamPointCloud`, which combines the information from previous time steps to visualize the complete scanned path. It is also possible to replace the `DepthCamera` source with a real world sensor so that real sensor data is visualized instead.

2.3 Improved Streaming Support

Some simulations, particularly in virtual commissioning scenarios, are controlled by external applications that rely on data from processed camera images. In order to provide such synthetic camera data for external image processing pipelines, we added support to stream virtual camera images to arbitrary targets. An example process can be seen in Figure 5 where the simulation is controlled from the output of an image processing software. The visualization of the simulation creates a depth image, which is sent to an external image processing pipeline to create control

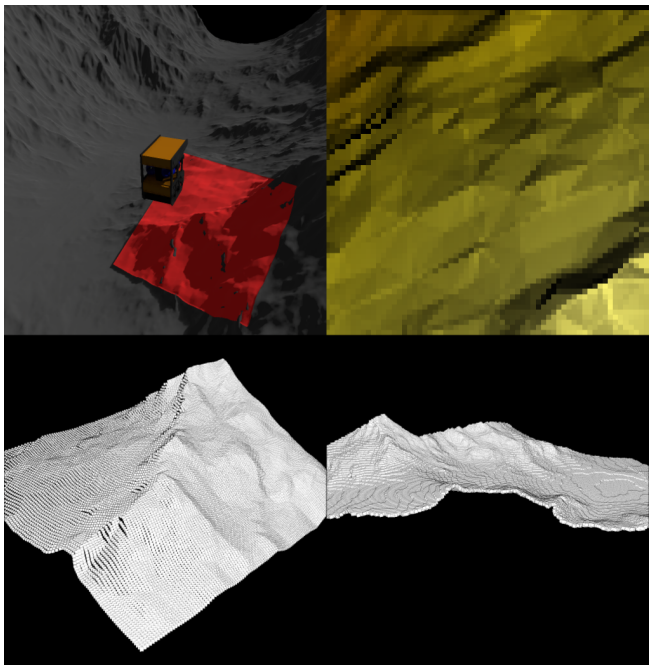

Figure 4. Example application for the use of depth information. Top left: ROV with depth camera floats above terrain, depth mesh in red is viewed area. Top right: current view with wrong colors and shading. Bottom left: current reconstructed landscape based on depth image. Bottom right: point cloud of combined time steps.

inputs for the simulation.

Every virtual camera has the possibility to stream its current view to a network target. The resolution of the stream is defined by a parameter that is independent from the resolution of the screen. The user can also set the target frame-rate of the stream to either reduce the workload of the visualization or to simulate real world limitations. The supported network protocols are UDP, TCP and RTSP.

The new `DepthCamera` can also stream their depth information to network targets, to simulate the results of a stereo vision pipeline. The depth information can be streamed using different options: Either the depth information is streamed via an encoded video (Pece, Kautz, and Weyrich 2011) or the data depth information of each pixel is packed into a FlatBuffers (Google 2014) package (either as 2D image or as 3D data) and subsequently streamed.

2.4 Testing Abilities

During testing of models, engineers want to change multiple parameters and compare the results of several simulations. In Modelica, it is feasible to vary parameters automatically (e. g. using Monte Carlo methods of the Optimization Library (Joos et al. 2002)), so the automated creation of comparison images should also be possible. This is why we added support to control the creation of screenshots and videos from Modelica.

To create screenshots automatically, the user specifies the simulation time points at which a screenshot should be taken. During simulation, the visualization will halt for a brief moment at the requested time points and create a screenshot. This also works for extremely fast simulations. The simulation time is appended to the given path so that the user can make several screenshots during one simulation. In Figure 6 (Pignède and

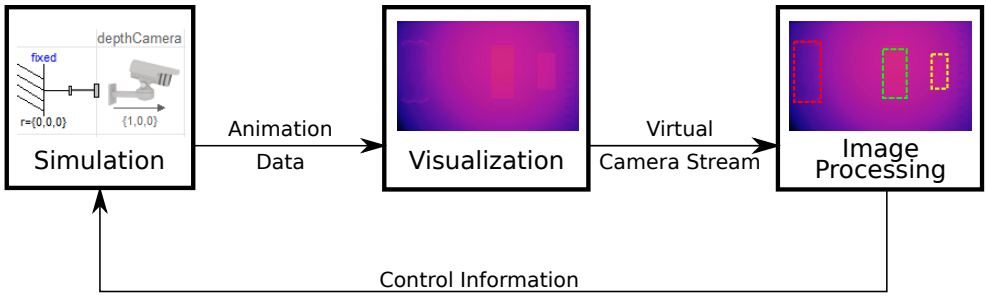

Figure 5. The process for a simulation controlled by cameras. The simulation sends animation data to the visualization, which renders a depth image and streams it to an external application. This application processes the depth image, calculates control information and sends it back to the simulation.

Lichtenheldt 2022) several screenshots are automatically created at different time steps for a short overview of the simulation.

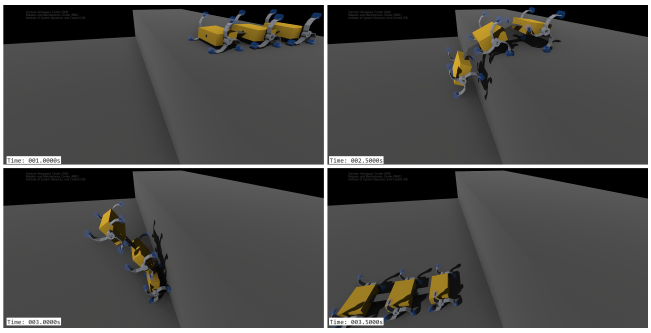

Figure 6. Screenshots automatically created at different time steps

Normally, a screenshot of all cameras that are displayed in the main window is created. A screenshot from a single camera can be triggered inside the individual cameras. Here, the screenshots can be taken at pre-defined times or when certain simulation events are happening. This is not only useful for checking results, but can also be used to mimic the functionality of a real world camera that is only able to create still images instead of videos.

Another option to compare results are auto-generated videos. When this option is enabled, the visualization will automatically create a video with the specified settings and save it to the specified path. This allows the user to run a multitude of simulations with varying parameters and simply compare the resulting videos.

2.5 Flow Visualization

In the previous version of the visualization library it was already possible to visualize the flow of media inside a pipe. In the new version the path is defined by a series of points. At each point, the user can define the desired position, the speed and the color of the visualized flow elements. The path in between the points is either interpolated cubically or linearly. The flow elements themselves can either be visualized with cones, rings, arrows or even imported CAD-Files. When using CAD-Files, the user can control the up-vector so that rotations along the flow axis are possible. This can also be used to visualize objects that consist of multiple small moving objects. In Figure 7 this is used to visualize the chain of a bicycle.

Another new addition is the visualization of flows inside of a three dimensional field. This can be used to visualize the

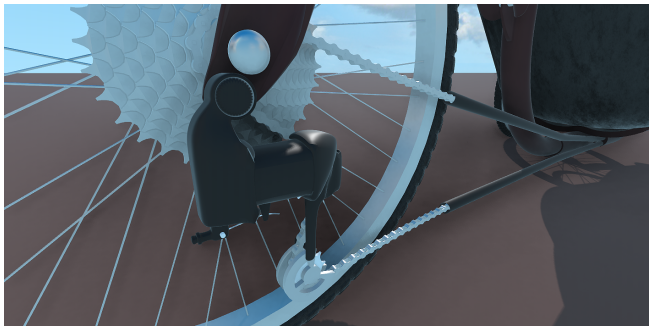

Figure 7. A moving chain is visualized with the `PathVectorFlow`

airflow around objects, like cars or wings. The vector field consists of a three dimensional array of points. At each point the user can specify a vector and its color. There exist two methods to visualize vector fields, `GridVectorField` and `GridVectorFlow`. `GridVectorField` visualizes the individual vectors as arrows which have the specified direction, length and color. `GridVectorFlow` visualizes the flow inside of this field. In order to do this, the user defines seeding points. At these points particles are inserted that follow the flow inside the field. The flow can also be visualized by lines that can either follow a virtual particle over time through the field or they can be simulated in a single time step. The lines are colored with the colors defined in the grid. An example flow around a sphere can be seen in Figure 8.

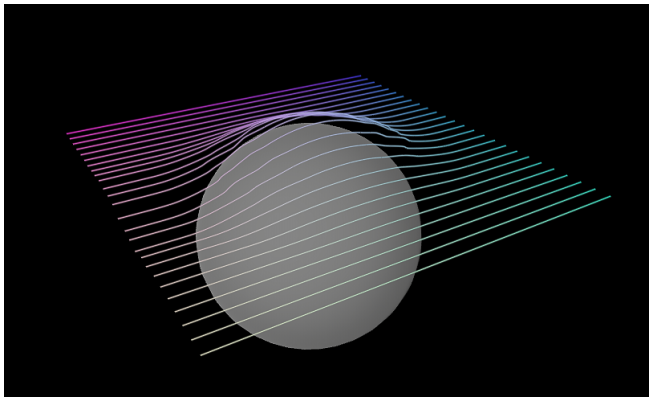

Figure 8. Visualized air flow around a sphere with the new `GridVectorFlow`

DOI 10.3384/ecp21181197

2.6 CAD Array

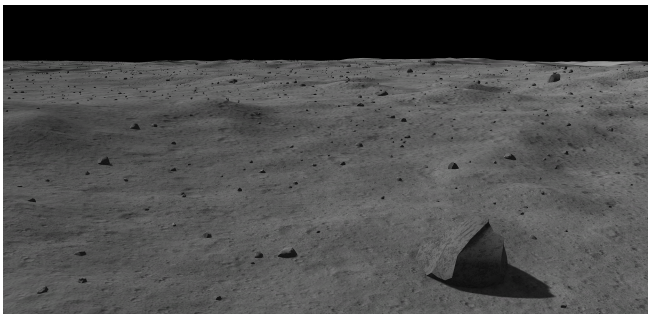

Figure 9. Use of the CAD array for the visualization of a distribution of 1.7 million rocks on a simulated moon landscape

There are situations where one would need to visualize large amounts of similar, static objects, e. g. rocks on a surface. Doing this with the normal `CADFile` leads to many equations in Modelica and therefore slows down the initialization and simulation significantly.

To counter this, `CADArray` allows the user to visualize a large number of instances of the same CAD file. For each instance the position, rotation and scale can be specified. This can either be done directly in Modelica via arrays or a file can be loaded that holds all of the information.

An example application can be seen in Figure 9. A lunar landscape had to be enhanced with rocks to provide additional detail and obstacles for a lunar landing simulation. The visualized moon is fairly large, so in order to achieve the desired obstacle density around 1.7 million rocks have to be placed and visualized on the surface. This would neither be possible to simulate nor to visualize without `CADArray`.

2.7 Level of Detail

Visualizing large amounts of objects with many details can be very challenging to render, so we introduced the possibility to specify multiple levels of detail (LoD).

A version of the object with many details is chosen as the main file. Additionally multiple LoDs can be added. Each of these LoDs is defined by a CAD file and a minimum visibility distance. At this distance the object will be visualized by the LoD's CAD file and the CAD file of the lower levels will be hidden.

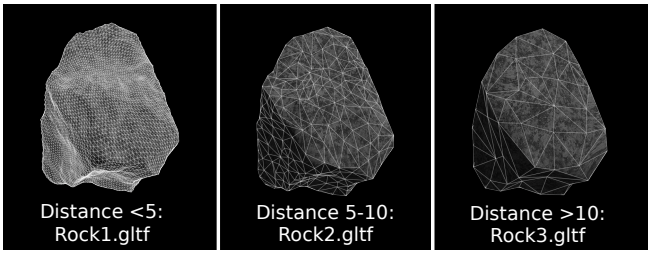

Figure 10. A rock displayed with varying polygon count at different distances. The user can specify what CAD files should be shown at which distance to reduce the stress on the GPU, while keeping visual quality high.

An example of a rock with multiple LoDs can be seen in Figure 10. When the viewer is less then five units away, the very detailed Rock1.gltf is shown. When the viewer is farther away at a distance of between 5 and 10 units, the less detailed Rock2.gltf is shown. At a distance over 10, the even less detailed Rock3.gltf is shown. This reduces the load on the GPU, while keeping visual quality high.

2.8 Additional Improvements

Interface

We completely reworked the Modelica interface in the DLR Visualization 2 Library to improve the overall usability of the library. We separated the primitives into individual objects so that at a glance, the user can see what kind of objects are displayed. Additionally, primitives and other objects with a main color are displayed in that color in the Modelica interface. A comparison can be seen in Figure 11.

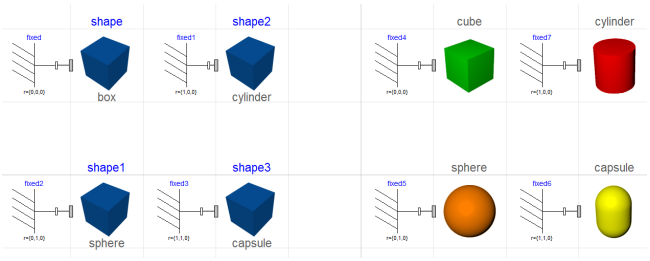

Figure 11. Comparison of the Modelica blocks for Primitives from the old library (left) to the new library (right). In the new library the primitives are split into different objects and the color is shown in the model

Rigged CAD File

We added the possibility to manipulate objects with bones that are defined in the CAD file. Manipulating objects with virtual bones is a standard in computer graphics. The most prominent use case is the creation of animated humans or animals, as it allows for the deformation of the skin. However, bones can also be used for mechanical structures like robots or rovers. Here, the advantage over individual subobjects is that the relative position of the objects is given in the CAD file and the user does not have to manually position them in the simulation model. For easier use, we added a Modelica function which extracts the bone information from the specified CAD file to create a model where the bones can be identified by their given names.

VR camera

We improved the usage of VR-Headsets. Now, every headset that uses the OpenVR standard is supported. Optionally, overlay items displayed for the VR user on a plane, which floats in front of the user, are now possible.

Another feature is the addition of green screen support (chroma keying). Here, a camera is mounted on the VR headset. The camera image will be displayed in the VR image and combined with the virtual image, which will be displayed at the masked green areas. This can be used to merge a real environment with a virtual environment, e. g. a real cockpit with a virtual landscape.

Window setup from within Modelica

The visualization window arrangement can now be controlled from within Modelica. This may be used for the automatic setup of a multi screen simulation. This includes the position, size and style. The position and size is specified relative to the screen

size, so that the appearance is independent from the screen resolution. The window can be a normal window with title bar and border, without any borders and title or fullscreen. It is also possible to create additional windows with the same properties.

Camera Position Back-Channel

The user can retrieve the position and orientation of every camera in Modelica. The cameras have an optional output for the position vector and the orientation matrix. These can be used to construct a frame and thereby objects can be attached to a user controlled camera. This also works for the `VRCamera`. This means that objects can be placed relative in the view of the VR user to, e. g. to create some additional interface.

Textures

In the previous version of the Visualization Library, textures (i. e. images or videos) were treated differently depending on the source. In the DLR Visualization 2 Library there is a common interface: the texture buffer. All individual sources (virtual camera, network stream, webcam, file) are treated the same and are interchangeable. This allows the user to iteratively test the simulation by simply exchanging the source of a texture and nothing else.

Overlay Positioning

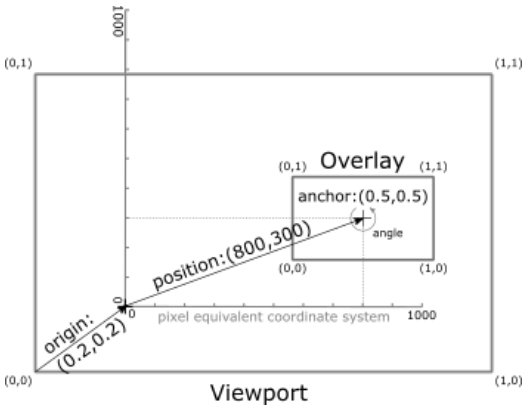

Figure 12. Possibilities of the overlay positioning mechanism. The origin is given relative to the viewport. The overlay position is relative to the origin and is given in pixel equivalents to allow for exact positioning of multiple interacting items. Finally the anchor can be used to change the handle position of the item.

We added more possibilities for the positioning of overlay items (formerly HUDs). In the previous version, the positioning was rather limited. The origin was always on the bottom left of the viewport. In Figure 12 the new possibilities can be seen. This is controlled by three positioning parameters:

- origin: relative position in the viewport
- position: relative to origin in pixel equivalents (equals 1/1000th of a defined side of the viewport)
- anchor: handle position of item relative to item size

This allows the user to position the overlay items to be aligned to any part of the viewport and also position it by any part of the item while still keeping the possibility for items to be positioned exactly for them to interact with each other.

Environments

Environments define the overall look of the simulation. They provide options for the sun light, the ambient light and the background. For the background, either a skybox can be defined or the sky color is calculated based on the position of the sun. Haze can be added to provide a more realistic visualization of large scale earth-based simulations.

3 Use Cases

While still under development, the DLR Visualization 2 Library is already used for many internal projects. A few practical applications will be introduced here.

3.1 Virtual Commissioning of a Robot Cell

Visualizing the overall assembly of a system early in the development process and during virtual commissioning has many benefits. Engineers can, for example, immediately asses the size of complex work-spaces with many moving parts to avoid collisions. Further, it is often times necessary to present a machine or plant to stakeholders long before it is physically built. However, if real components rely on optical sensor values, e. g. an imaging pipeline to control a pick and place task, this can be addressed in the simulation visualization as well.

Such an application is shown in Figure 13: Two interacting robots assemble housings for network components (Bellmann, Seefried, and Thiele 2020; Reiser 2021). For this task they are equipped with depth cameras on the side of their tools to detect the exact position of a workpiece. In this complex application the visualization is part of virtual commissioning. It creates depth images as shown in Figure 14 and streams them to an image processing algorithm. Its results then provide positioning input to the robot controllers and thereby close the control loop over visualization and image processing.

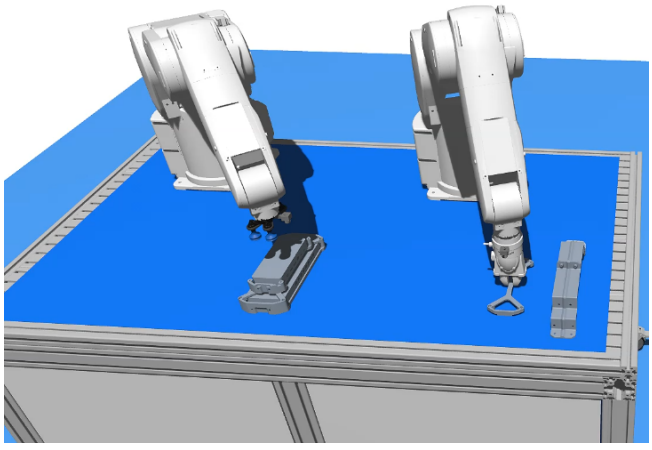

Figure 13. Example for virtual commissioning of a robot cell. Two cooperative robots assemble housings for network components.

3.2 Visualization of Parallel Running Simulations

The Helmholtz Future Project ARCHES develops teams of heterogeneous robotic agents for deep-sea and extraterrestrial exploration with the goal of improving efficiency and robustness

DOI 10.3384/ecp21181197

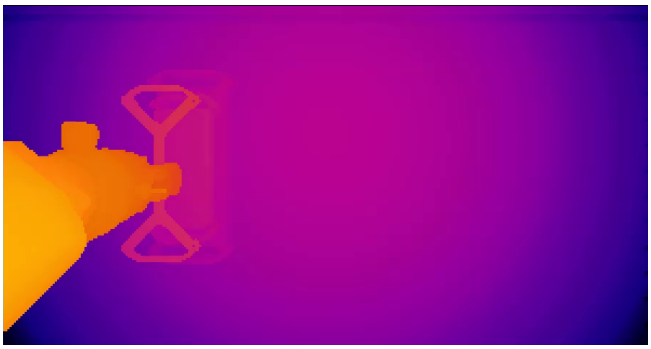

Figure 14. Depth image produced by a stereo camera, mounted on one of the robots tool. Color denotes distance from camera.

under extreme conditions (Schuster et al. 2020). During the development of such a team, only a limited number of robots is actually available and using those for regular software tests or during development is complex and laborious. So simulations play an important role during development of robot teams. Yet the simulation of a large number of agents is again difficult, especially in an efficient manner. Almost all available Modelica implementations are very limited when it comes to parallelization. Typically the whole simulation is run in one monolithic block on a single CPU core (Gebremedhin 2019). So to simulate a whole team of robots, each agent is split into a single simulation. The simulations are then only loosely coupled, but they all have to be visible in one common visualization. The DLR Visualization 2 library supports the rendering of multiple simulations in one visualization container.

In this use case a multi-agent simulation has been created to simulate multiple rovers on a virtual but realistic landscape. It provides a network API which allows the developers of control algorithms for the rovers to send commands to the robots and to receive status data from them. Among this data are video streams of the simulated robots cameras e. g. for a navigation pipeline. Figure 15 shows a common visualization of multiple

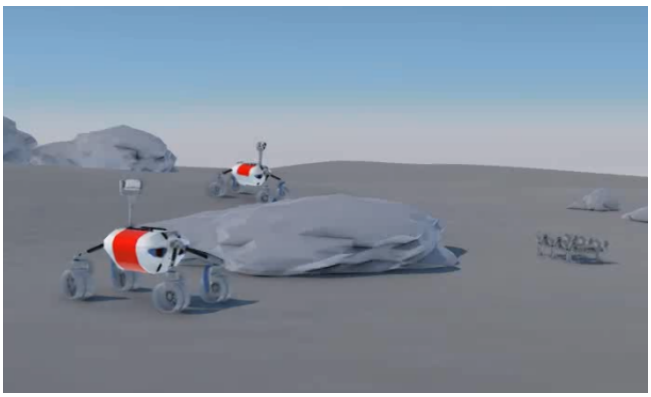

Figure 15. Multiple rovers, each from a separate simulation, rendered in one visualization

largely independent, simulations in one visualization container.

The virtual camera images should be as close to real ones as possible to give operators a realistic impression during development and training. The video streams might even be used as input for image processing algorithms for tests and during development as presented in Wedler et al. (2017).

3.3 Automated Testing of Rover Operations

The Martian Moon eXploration mission (MMX) is a cooperative effort by the Japanese space agency JAXA, the French space agency CNES, and the German space agency DLR. Together they plan to explore the martian moons Phobos and Deimos (Ulamec et al. 2019). An important part of this project is the deployment of a mobile rover on Phobos (Bertrand et al. 2019; Buse et al. 2021). Designing the first wheeled rover for use on Phobos is a highly complex task. Little is known about many environmental factors such as the surface structure and what is known, like the micro gravity, poses a number of new problems that no previous wheeled exploration rover has ever faced (Bertrand et al. 2019; JAXA 2017; Lange 2020).

An environment like this cannot be recreated anywhere on earth, therefore simulations and their visualization play a central role during the development. One of the most challenging tasks is the simulation of the wheel-ground contact under low-g conditions. Without a visualization it is very hard to get a real insight into the complex interaction of multiple contact points in a non-intuitive environment.

Such visualizations are also directly involved in the development of the locomotion planning tool for the rover. A simulated navigation and wheel cameras will be used to determine the feasibility of a planned locomotion trajectory.

Finally, space missions also have to present their work and their results to the general public in an interesting and engaging fashion. For this, nowadays, high quality render images of the mission are generally expected. Figure 16 shows such an image, presented, for example, on JAXAs MMX mission website (JAXA 2020).

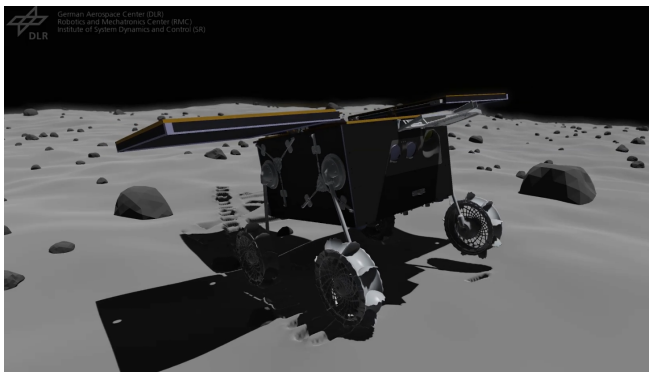

Figure 16. Visualization of the simulated MMX Rover on Phobos. The rover moves on soft soil with large rocks in a micro gravity environment.

4 Conclusion and Outlook

Whilst the possibility to export FMU of Modelica models enables virtual commissioning of the physical system behavior, the usage of visually rendered environments for virtual commissioning is not standardized yet. Especially the integration of real imaging pipelines requires a high-definition real-time rendering engine with video stream capabilities, not available in most Modelica tools. In this work we presented the commercially available DLR Visualization 2 Library and its new features aiming at integrating such pipelines. Another big step in usability is the possibility to visualize the results of multiple, parallel running simulations.

For future developments, it is planned to extend the DLR Visualization 2 Library with a generic C++ API, a Matlab/Simulink interface and a JULIA library. The library, interfaces and the render tool itself will be provided in a free community edition limited to real-time multi-body visualizations and a commercial version containing all features.

Acknowledgments

The authors would like to thank all colleagues at DLR helping during the implementation and testing of the library, especially Robert Reiser, Fabian Buse, Antoine Pignede and Miguel Neves, who also provided some of the examples and use cases. Additionally, the authors would like to thank GEOMAR for the model of the ROV. This work was partially funded by the HVF 68 Project Lighthouse.

References

Bellmann, Tobias (2009). "Interactive simulations and advanced visualization with modelica". In: *Proceedings of the 7th international Modelica conference*. Linköping University Electronic Press.

Bellmann, Tobias, Andreas Seefried, and Bernhard Thiele (2020-10). "The DLR Robots library – Using replaceable packages to simulate various serial robots". In: *Proceedings of Asian Modelica Conference 2020* (Tokyo, Japan). Linköping University Electronic Press. DOI: 10.3384/ecp2020174153.

Bertrand, Jean et al. (2019-05). "Roving on Phobos: Challenges of the MMX Rover for Space Robotics". In: *15th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)* (Noordwijk, Netherlands). ESA/ESTEC.

Buse, Fabian et al. (2021-10). "Wheeled locomotion in milligravity: A technology experiment for the MMX Rover". In: *72th International Astronautical Congress* (Dubai, UAE). The International Astronautical Federation.

Fuchs, Marcus, Rita Streblow, and Dirk Müller (2015). "Visualizing simulation results from modelica fluid models using graph drawing in python". In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. 118. Linköping University Electronic Press, pp. 737–745.

Gebremedhin, Mahder (2019-01). *Automatic and Explicit Parallelization Approaches for Equation Based Mathematical Modeling and Simulation*. Linköping University Electronic Press. DOI: 10.3384/diss.diva-152789.

Google (2014). *FlatBuffers*. URL: https://google.github.io/flatbuffers (visited on 2021-04-22).

Greenberg, Donald P. et al. (1997). "A Framework for Realistic Image Synthesis". In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. USA: ACM Press/Addison-Wesley Publishing Co., pp. 477–494. ISBN: 0897918967. DOI: 10.1145/258734.258914. URL: https://doi.org/10.1145/258734.258914.

Hellerer, Matthias, Tobias Bellmann, and Florian Schlegel (2014). "The DLR Visualization Library-recent development and applications". In: *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*. 096. Linköping University Electronic Press, pp. 899–911.

JAXA (2017-08). *Gravity both too strong and too weak: landing on the Martian moons*. URL: https://mmx-news.isas.jaxa.jp/?p=331&lang=en (visited on 2021-04-22).

JAXA (2020-10). *The MMX Rover is undergoing tests for landing*. URL: https://mmx-news.isas.jaxa.jp/?p=1271&lang=en (visited on 2021-04-22).

Joos, Hans-Dieter et al. (2002). "A multi-objective optimisation-based software environment for control systems design". In: *IEEE International Conference on Control Applications and International Symposium on Computer Aided Control Systems Design, 2002-09-18 - 2002-09-20, Glasgow, Scotland (UK)*. IEEE, pp. 7–14.

Khronos (2021). *glTF Specification Webpage*. URL: https://www.khronos.org/gltf (visited on 2021-04-22).

Lange, Michael (2020-09). *First tests for landing the Martian Moons eXploration Rover*. URL: https://www.dlr.de/content/en/articles/news/2020/03/20200930_in-free-fall-to-the-martian-moon-phobos.html (visited on 2021-04-22).

Otter, Martin, Hilding Elmqvist, and Sven Mattsson (2003-11). "The New Modelica MultiBody Library". In: pp. 311–330.

Pece, Fabrizio, Jan Kautz, and Tim Weyrich (2011). "Adapting Standard Video Codecs for Depth Streaming". In: *Joint Virtual Reality Conference of EGVE - EuroVR*. Ed. by Sabine Coquillart, Anthony Steed, and Greg Welch. The Eurographics Association. ISBN: 978-3-905674-33-0. DOI: 10.2312/EGVE/JVRC11/059-066.

Pignède, Antoine and Roy Lichtenheldt (2022). "Modeling, Simulation and Optimization of the DLR Scout Rover to Enable Extraterrestrial Cave Exploration". In: *The 6th Joint International Conference on Multibody System Dynamics (IMSD) and The 10th Asian Conference on Multibody Dynamics (ACMD), New Delhi, India: October 16-20, 2022*. Accepted for publication.

Reiser, Robert (2021). "Object Manipulation and assembly in Modelica". In: *Proceedings of the 14th international Modelica conference*. Linköping University Electronic Press.

Ribes, Alejandro and Christian Boucheny (2011-04). "Eye-Dome Lighting: a non-photorealistic shading technique". In: URL: https://blog.kitware.com/eye-dome-lighting-a-non-photorealistic-shading-technique (visited on 2021-04-22).

Schuster, Martin J. et al. (2020-10). "The ARCHES Space-Analogue Demonstration Mission: Towards Heterogeneous Teams of Autonomous Robots for Collaborative Scientific Sampling in Planetary Exploration". In: *IEEE Robotics and Automation Letters* 5.4, pp. 5315–5322. DOI: 10.1109/lra.2020.3007468.

Ulamec, S. et al. (2019-10). "A rover for the JAXA MMX Mission to Phobos". In: *70th International Astronautical Congress* (Washington DC, USA). The International Astronautical Federation. Chap. A3.

Unigine (2021). URL: https://unigine.com/ (visited on 2021-04-22).

VDI/VDE-Fachbereich Engineering und Betrieb (2016-08). *Virtual commissioning - Model types and glossary*. Tech. rep. VDI/VDE 3693. Verein Deutscher Ingenieure e.V., p. 35.

Waurich, Volker and Jürgen Weber (2017). "Interactive FMU-based visualization for an early design experience". In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*. 132. Linköping University Electronic Press, pp. 879–885.

Wedler, Armin et al. (2017-09). "First Results of the ROBEX Analogue Mission Campaign: Robotic Deployment of Seismic Networks for Future Lunar Missions". In: *68th International Astronautical Congress* (Adelaide, Australia). The International Astronautical Federation.