D5.3.3 – Remote hybrid rendering: first prototype tool

WP5: User tools

Project Acronym	CRESTA
Project Title	Collaborative Research Into Exascale Systemware, Tools and Applications
Project Number	287703
Instrument	Collaborative project
Thematic Priority	ICT-2011.9.13 Exascale computing, software and simulation

Due date:	M18
Submission date:	31/03/2013
Project start date:	01/10/2011
Project duration:	36 months
Deliverable lead organisation	USTUTT
Version:	1.0
Status	Final
Author(s):	Martin Aumüller (USTUTT)
Reviewer(s)	Achim Basermann (DLR), Alan Gray (UEDIN), Stefano Markidis (KTH)

Dissemination level	
PU	*PU - Public*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	28/02/2013	First version of the deliverable	Martin Aumüller (USTUTT)
0.2	06/03/2013	Describe installation and configuration, add PDF version of documentation extracted from source code	Martin Aumüller (USTUTT), Achim Basermann (DLR)
0.3	20/03/2013	Added usage example and explanation of remote hybrid rendering	Martin Aumüller (USTUTT), Alan Gray (UEDIN), S. Markidis (KTH)
1.0	20/03/2012	Final version of the deliverable	Martin Aumüller (USTUTT),

Table of Contents

1 Executive Summary

This document accompanies the software delivered as the first prototype of remote hybrid rendering.

Remote hybrid rendering is used to access remote exascale simulations from immersive projection environments over the Internet. The display system may range from a desktop computer to an immersive virtual environment such as a CAVE. The display system forwards user input to the visualisation cluster, which uses highly scalable methods to render images of the post-processed simulation data and returns them to the display system. The display system enriches these with context information before they are shown.

Together with the documentation extracted from the source code in the appendix, this document describes the first prototype for remote hybrid rendering. It has been implemented as plug-ins to the virtual reality renderer OpenCOVER [5] of the visualization system COVISE [6]. The source code of these plug-ins is open and can be retrieved from the CRESTA project subversion repository.

While implementing the prototype, some changes to the protocol draft for remote hybrid rendering became necessary.

Future versions of the tool will be improved regarding bandwidth requirements and scalability.

2 Remote Hybrid Rendering

As transferring the results of a large-scale simulation to a local system for rendering is not viable [2], one often takes recourse to remote rendering: instead of post-processed data, rendered images are transmitted to the display. The very much lowered bandwidth and processing requirements of remote rendering allow for making efficient use of remote compute resources by a much larger user base.

Head-tracked immersive virtual environments, where the rendering is constantly updated according to the user's current head position, require high frame rates and low reaction latencies to achieve a high sensation of presence and to avoid motion sickness [3]. These immersive visualisation environments provide more intuitive ways for specifying the location of regions of interest, cutting planes, seed points for particle traces, or reference points for iso surface extraction than desktop-based systems. We aim to enable users to experience exascale simulations in such immersive environments over the Internet.

To improve frame rate and reaction times, we will try to decouple interaction from network latencies as far as possible, but still without requiring to transfer huge data to the client. Only extracted features from simulation results will be rendered either directly on the simulation host or on a remote visualisation cluster employing scalable methods. But "context information" such as essentially static geometry, as e. g. turbine shapes, interaction cues for the parameters controlling the visualisation algorithms applied on the visualisation cluster and menus will be rendered locally, at a rate independent of the remote rendering. As both remotely and locally rendered images are composited for the final display, we call this technique "remote hybrid rendering". This compositing usually takes pixel depth into account, but it might also use opacity information.

Please refer to section 4.3 for an illustration of this process by a concrete example.

3 First Prototype Implementation of Remote Hybrid Rendering

3.1 Implementation Details

Both, the client (local) as well as the server (remote) side of the prototype for remote hybrid rendering are implemented based on the same software: OpenCOVER [5], the virtual reality renderer of the visualisation system COVISE [6].

The server has to load the VncServer plug-in, while the client needs the plug-in VncClient. There are no other differences between server and client.

Please refer to the appendix with documentation extracted from the implementation for further details.

3.1.1 VncServer Plug-in

The VncServer plug-in for OpenCOVER provides a full implementation of a VNC server: every VNC client can connect to it and interact with the visualization with keyboard and mouse. For implementing this functionality, the library LibVNCServer [9] has been used.

For remote hybrid rendering, it has been augmented with the following features:

- Transmission of depth data (z-buffer) from server to client for enabling compositing with image contributions rendered on the client
- Reception of 3D viewer and pointer positions sent by client
- Reception of interaction data sent by client

These additional features can only be exploited by specially adapted VNC clients.

Colour image data is compressed using VNC's possibilities, for compressing depth data the snappy compressor library is used [8].

There are two methods for copying the image data from GPU to CPU: one that relies purely on the OpenGL API call *glReadPixels*, and another one that employs CUDA for the transfer from GPU to CPU memory. Especially on gaming class hardware, resorting to CUDA provides better performance [7].

3.1.2 VncClient Plug-in

The VncClient plug-in for OpenCOVER is such a specially adapted VNC client. It retrieves both colour image and depth data from the server and renders these as an additional node in its scenegraph. This achieves compositing of remote and local content. During each frame, the current values of the matrices describing the positions of the user's head and hand are sent to the server. In addition, the results of user interactions, e.g. new seed points for particle traces, are transmitted to the server.

3.2 Obtaining the Software

The two plug-ins are available as open source, while OpenCOVER is not. For compiling and using the plug-ins, at least Subversion revision 24237 of COVISE is required, as implementing remote hybrid rendering necessitated changes to the plug-in interface of OpenCOVER. Pre-compiled versions of COVISE for testing the software can be downloaded from https://fs.hlrs.de/projects/covise/support/download/.

The source code of the software, i.e. the two plug-ins for OpenCOVER, can be accessed using Subversion at https://svn.ecdf.ed.ac.uk/repo/ph/cresta/wp5/remoterendering/trunk. The documentation can be extracted by running doxygen in the directory RHR/html. Additionally, this document contains a PDF version of the extracted documentation in the appendix.

4 Using the Software

4.1 Configuration of OpenCOVER

In order to use remote hybrid rendering, OpenCOVER has to be configured to load the VncServer plug-in on the remote system. This is done by adding the tag *<VncServer />* within *<COVER><Plugin></Plugin></COVER>* to the XML configuration file for COVISE. Additionally, the TCP port, where the server waits for requests, can be changed from its default 5900 by adding the attribute rfbPort=”portnumber”. In addition, the precision of the transmitted depth values can be configured with the attribute depthPrecision. The possible values are 8, 16, 24, and 32 for the corresponding number of bits. The default is 16.

On the client system, OpenCOVER has to load the VncClient plug-in. Analogously, this is achieved by adding the tag <VncClient />. In most cases, the attribute rfbHost has to be given a value, in order to establish a connection to a VncServer plugin running on another system but localhost. The attribute rfbPort can be used to change the TCP port to which the client will try to connect.

More detailed information on configuring OpenCOVER in general is available in [1].

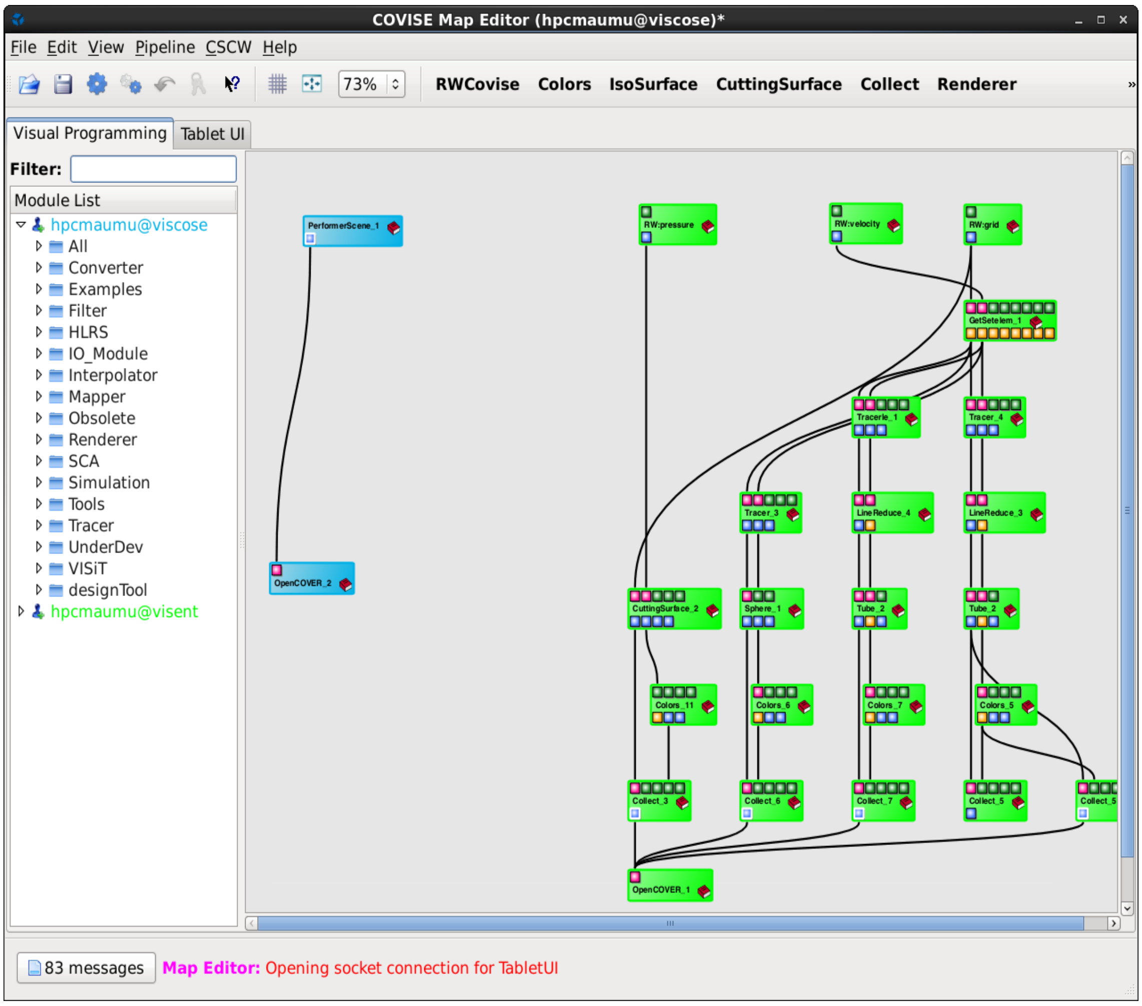
4.2 COVISE

A typical visualization session with remote hybrid rendering will rely on the ability of COVISE to distribute visualization modules across several systems. The compute and data intense tasks will be handled on the powerful remote system, while the local system will only be used to display menus or some static geometry to provide context for the remote visualization results. Hence, the local system will be configured to run only OpenCOVER and perhaps very few simple additional modules, while the remote system will also run OpenCOVER as well as a larger amount of modules for analyzing the data.

The next subsection describes such a work flow.

4.3 Usage Example

During the implementation of the prototype, a visualization of the simulated air flow around an Audi A8 has been used. The data flow network of the post-processing modules has been organized as depicted in Figure 1. Data flows from top, starting with file input modules, to bottom. The modules depicted in light blue are executed on the local (i.e. connected to the screen displaying the final image) system, while the modules coloured in light green are executed on the remote system.



Map Editor: Opening socket connection for TabletUI

Figure 1: data flow network for remote hybrid rendering.

The remote system is used for post-processing the results of the flow simulation and rendering the corresponding visualizations, such as stream lines as well as a plane cutting through the flow field colourized according to air pressure. Figure 2 shows the resulting image rendered by the modules labeled *OpenCOVER_1* in Figure 1.

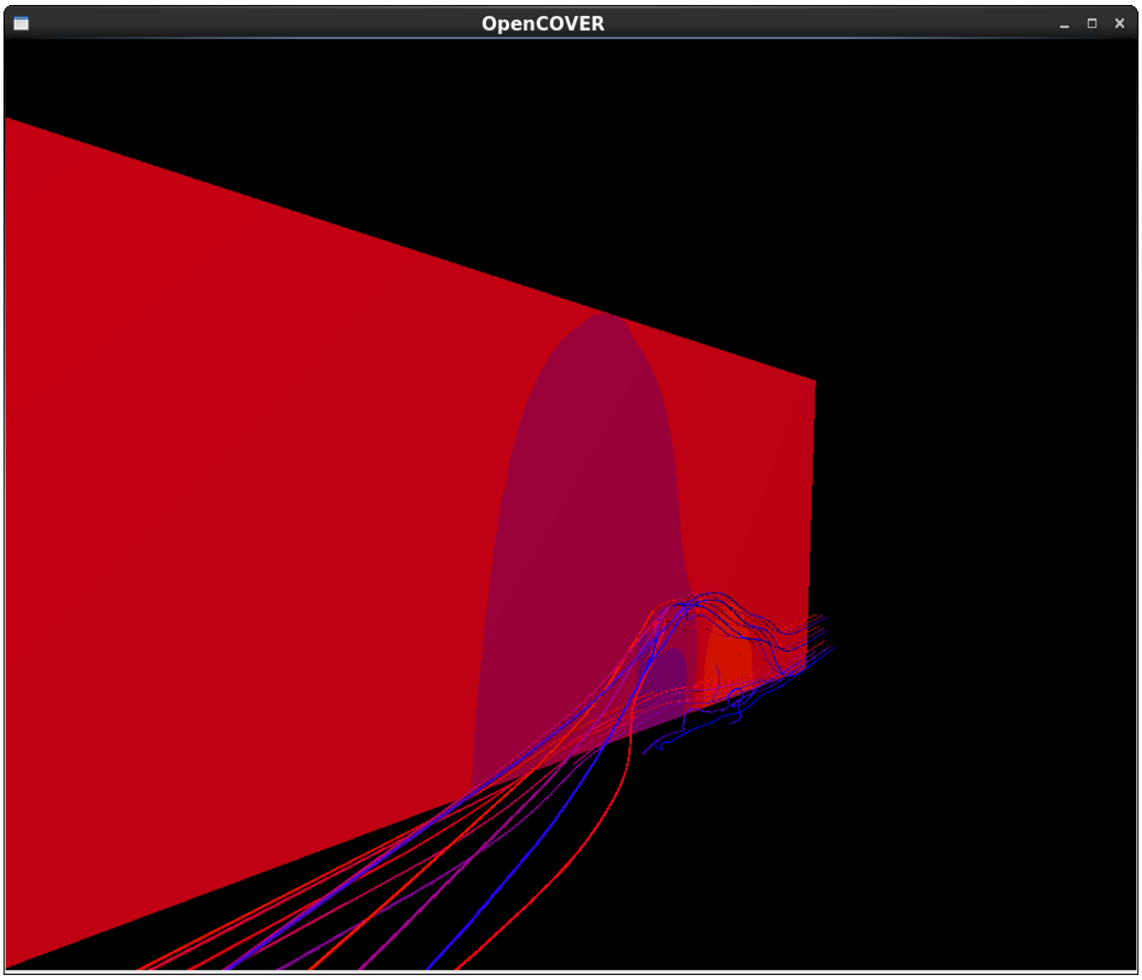


Figure 2: remotely rendered flow visualization.

The local system renders context information. This comprises the menu and interaction elements, e.g. for moving the cutting plane. But also the static geometry of the car is rendered locally. Figure 3 shows the corresponding image produced by the module labeled *OpenCOVER_2*.

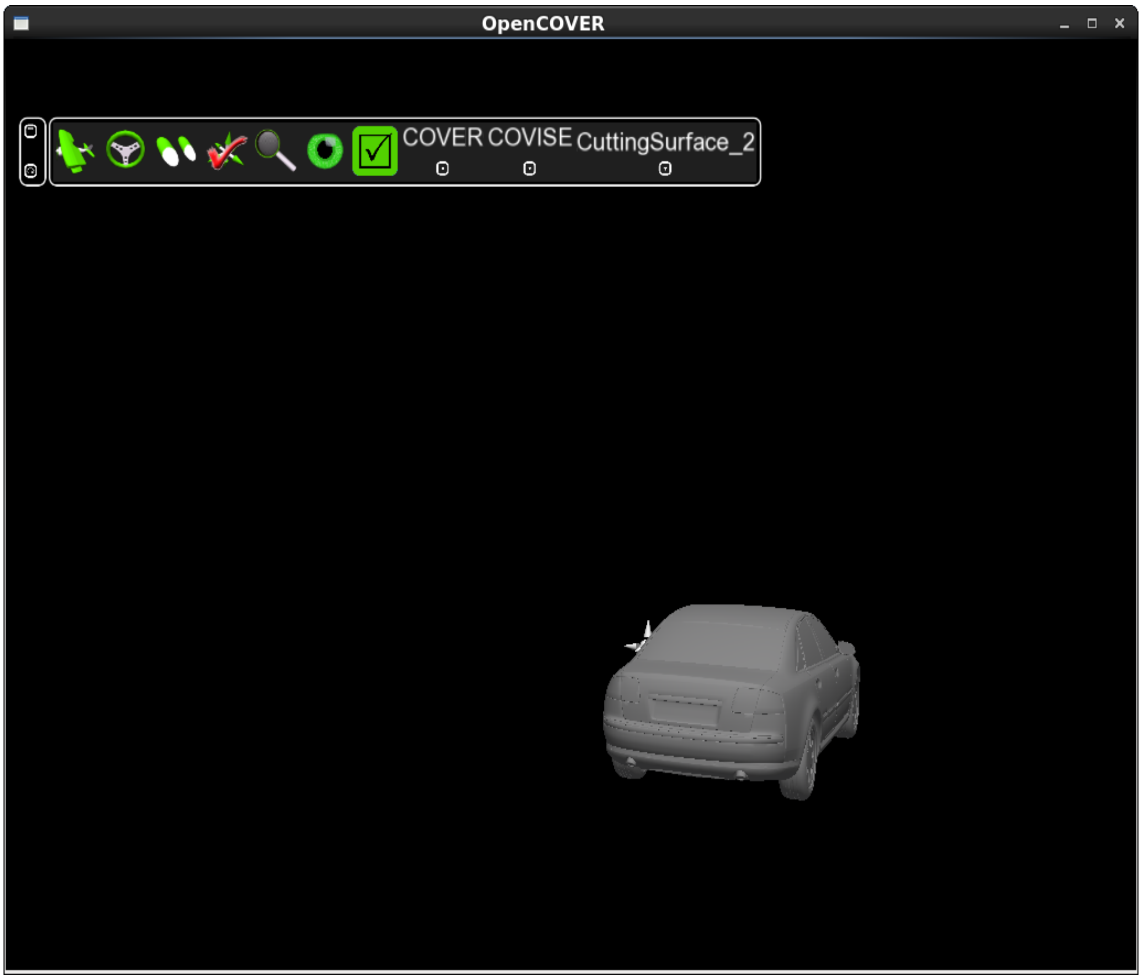


Figure 3: locally rendered context information.

In a final step before displaying the result, locally and remotely rendered images are composited taking the distance to the viewer of the geometry object contributing the pixel's colour into account: the closer pixel of the two images is copied into the final image, as shown in Figure 4.

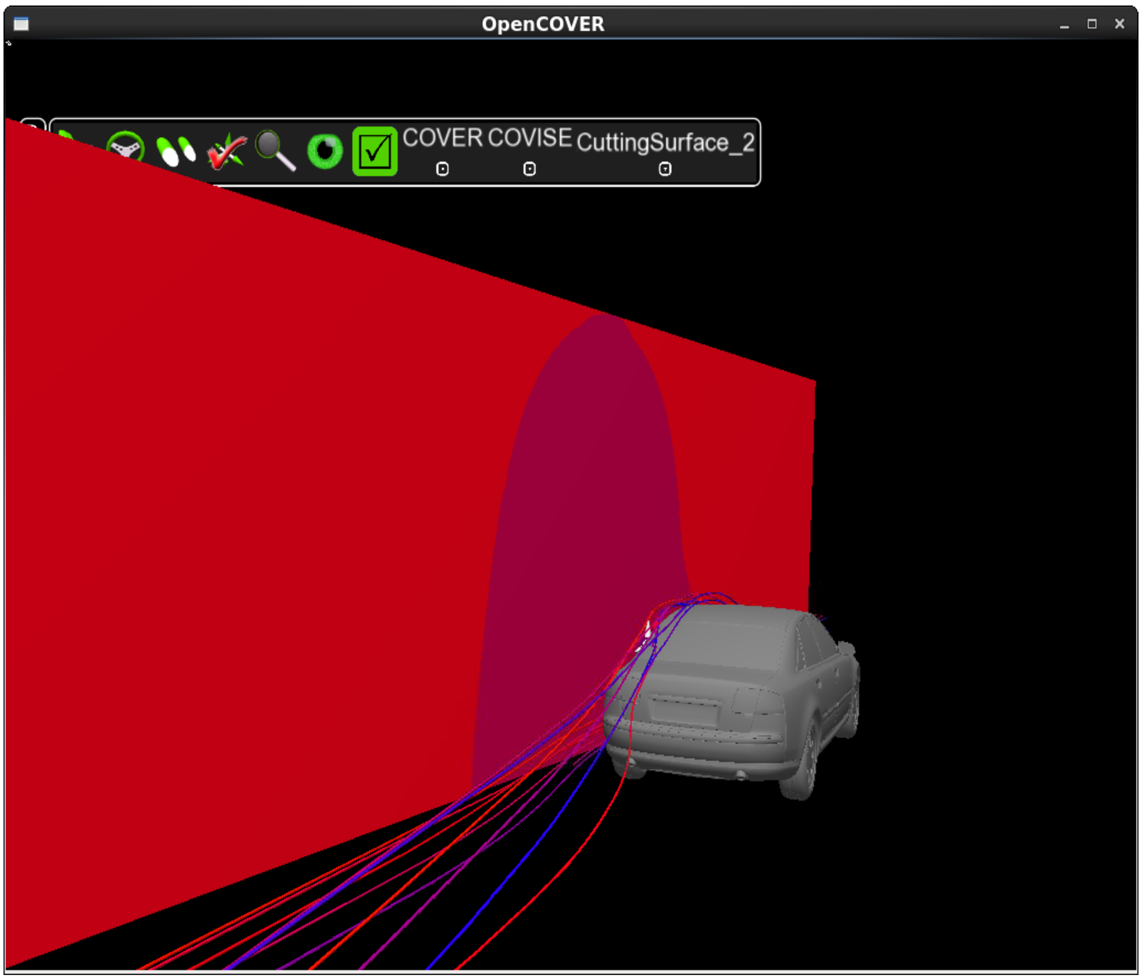


Figure 4: final image resulting from compositing local and remote contributions.

Figure 1 does not show the VNC connection used for transmitting the image and user interaction between the remote *OpenCOVER_1* and the local *OpenCOVER_2*. This has to be configured as described in section 4.1.

All the interactive features of the visualization system are available even though parts of the rendering are delegated to a remote system. E.g. new seed points for stream lines can be placed by interacting with the visualization. Only the fact that the remote parts of the image are updated less frequently make this visualization distinguishable from a purely local visualization.

5 Changes to the Protocol Draft

While implementing the remote hybrid rendering server and client, it became apparent that the protocol drafted in D5.3.2 [2] has to be revised:

- It is not necessary to send input data, such as from multi-touch or 6DOF devices, from client to server. Instead, input events have to be processed by the local client application.
- The state of the local application has to be synchronized with the remote application, e. g. matrices describing the viewer's position and the transformation of objects have to be transmitted to the remote server, such that locally rendered images and remote images can be matched during compositing.
- Some operations have to be carried out collectively on the remote and local renderer, e. g. the scene bounding sphere has to comprise bounding spheres for both local and remote data, this requires support from the protocol.
- Application state has to be synchronized between client and server applications, this requires application-specific protocol extensions. E. g., the rendering and lighting modes configured on the client also have to be applied on the client.
- Some actions have to be carried out cooperatively between client and server. E.g. moving a cutting plane to another position requires sending the updated parameters from client to server, where the application has to extract the corresponding data and to update the rendered image. This requires protocol support for sending the data describing the interaction capabilities from server to client as well as for updating parameters from client to server.

6 Future Work

Future versions of the software will be improved regarding the following aspects:

- resizing of server framebuffer to match client window size,
- improved compression algorithms for depth data,
- integration with the prototype tool for massive parallel visualization that is currently being developed,
- interoperability with server-side parallel rendering as described in D5.3.1 [4],
- reprojection of 2.5D data according to current view point,
- better synchronization of client and server state.

In addition, the prototype available now will be instrumented to obtain first performance results and latency measurements. The results will be used for used for improving the prototype for D5.3.4 (Remote hybrid rendering: revision of system, protocol definition).

7 References

[1] Wössner, U. and Rainer, D.: COVISE Installation & Configuration, 2008, https://fs.hlrs.de/projects/covise/doc/pdf/cover_inst_config.pdf.

[2] Aumüller, M.: CRESTA D5.3.2: Remote hybrid rendering: protocol definition for exascale systems, 2012.

[3] M. Usoh, K. Arthur, M. Whitton, R. Bastos, A. Steed, M. Slater, and F. Brooks, "Walking > walking-in-place > flying, in virtual environments," *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, Jul. 1999.

[4] F. Niebling, J. Hetherington, and A. Basermann, "D5.3.1 – Remote hybrid rendering: analysis and system definition for exascale systems," CRESTA, Mar. 2012.

[5] U. Woessner, D. Rantzau, and D. Rainer, "Interactive Simulation Steering in VR and Handling large Datasets," *IEEE Virtual Environments*, Jan. 1998.

[6] D. Rantzau, K. Frank, U. Lang, D. Rainer, and U. Woessner, "COVISE in the CUBE: An Environment for Analyzing Large and Complex Simulation Data," *2nd Workshop on Immersive Projection Technology*, 1998.

[7] F. Niebling, A. Kopecki, and M. U. Aumüller, "Integrated Simulation Workflows in Computer Aided Engineering on HPC Resources", International Conference on Parallel Computing, 2011, Ghent.

[8] Snappy – a fast compressor/decompressor [Online], Available: https://code.google.com/p/snappy/, [Accessed: 23 Feb. 2013].

[9] LibVNCServer/LibVNCClient [Online], Available: http://libvncserver.sourceforge.net, [Accessed: 20 Feb. 2013].