D5.3.5 – Remote hybrid rendering: final tools

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:	M30
Submission date:	31/03/2014
Project start date:	01/10/2011
Project duration:	39 months
Deliverable lead organisation	USTUTT
Version:	1.0
Status	Final
Author(s):	Martin Aumüller (USTUTT), Markus Flatken (DLR), Timo Krappel (USTUTT)
Reviewer(s)	Alan Grey (EPCC), David Lecomber (ASL)

Dissemination level	
PU	*PU - Public*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers

0.1	03/03/2014	First version of the deliverable	Martin Aumüller (USTUTT)
0.2	24/03/2014	Improvements based on suggestions by co-authors and reviewers	Martin Aumüller, Timo Krappel (both USTUTT), Markus Flatken (DLR), Alan Grey (UEDIN), David Lecomber (ASL)
0.3	26/03/2014	Corrections	Martin Aumüller (USTUTT)
1.0	31/03/2014	Final version of the deliverable	Martin Aumüller (USTUTT), Markus Flatken (DLR), Timo Krappel (USTUTT)

Table of Contents

1 Executive Summary

This document accompanies the software delivered as the final tool for remote hybrid rendering (RHR).

RHR 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. RHR decouples local interaction from remote rendering and thus guarantees smooth interactivity during exploration of large remote data sets.

Together with the documentation extracted from the source code, this document describes the final tool for remote hybrid rendering. The client has been implemented as a plug-in to the OpenGL based virtual reality renderer OpenCOVER [5] of the visualization systems COVISE [6] and Vistle [14]. A server is also implemented as a plug-in for OpenCOVER. It can be used together with a compositor plug-in to scale the performance with the number of available GPUs. The source code of these plug-ins is open and can be retrieved from the CRESTA project subversion repository.

In order to scale with the number of nodes on systems that do not provide OpenGL support, a CPU based data-parallel interactive ray casting render module for Vistle has been implemented. This renderer also provides a server for remote hybrid rendering. This software is available as part of Vistle from its publically accessible GitHub repository [14].

Synchronization between the nodes attached to a tiled display naturally happens in the client application, as all data transfer is funnelled through the head nodes of the local and remote systems [3]. On the other hand, reprojection of 2.5D images according to current viewing parameters automatically brings all tiles into a synchronized state.

While implementing the first prototype of RHR for D5.3.3 [4], it became clear that the protocol proposed in D5.3.2 [3] based on the DoW had to be changed: interaction, including multi-touch interaction, and head tracking have to be handled by the client application, as these have to affect the handling of the local context information as well. In D5.3.4 [5] the protocol was updated to reflect this. Hence, the protocol for RHR only sends viewing parameters, derived from user interaction and head tracking, from client to server, which responds with 2.5D images, which are merged with locally rendered content.

This design enables the cooperation of light-weight renderers with display programs that contain most of the application logic and interaction handling. This allows for easy integration of RHR with a multitude of applications that operate on a 3-dimensional domain. The sole requirement is that the application is able to generate colour images together with depth data describing the distance of the visible pixels to the viewer. Within CRESTA, this would allow to extend the use of RHR from OpenFOAM to all other applications and especially HemeLB, as this already comes with its own integrated image generator.

First experience gained during development shows that the performance of the system could benefit from further latency reduction by providing better compression, and more overlap between rendering and compression/transmission. Additionally, it seems worthwhile to provide a software framework that enables easy integration of a RHR server into existing rendering software.

2 Overview

The next chapter summarizes the basic principles of remote hybrid rendering. In chapter 4 we describe our implementation. Chapter 5 shows typical usage of the software. Chapter 6 concludes by summarizing shortcomings which we would have liked to remedy, given more time. Additional documentation can be extracted from the source code with doxygen.

2.1 Glossary of Acronyms

2.5D	image data together with depth data
API	Application Programming Interface
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture (general purpose parallel GPU programming platform)
GPU	Graphics Processing Unit
JPEG	Joint Photographic Experts Group
OpenGL	Open Graphics Library (graphics rendering API)
RFB	Remote Framebuffer Protocol (used by VNC)
RGBA	Red/Green/Blue/Alpha (framebuffer format for colour and opacity)
RHR	Remote hybrid rendering
VNC	Virtual Network Computing
WP	Work Package

3 Remote Hybrid Rendering

As in many cases transferring the results of a large-scale simulation, e. g. a long-running OpenFOAM case, 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 final display, we call this technique "remote hybrid rendering" (RHR) or "hybrid remote rendering" [16]. This compositing usually takes pixel depth into account, but it might also use opacity information.

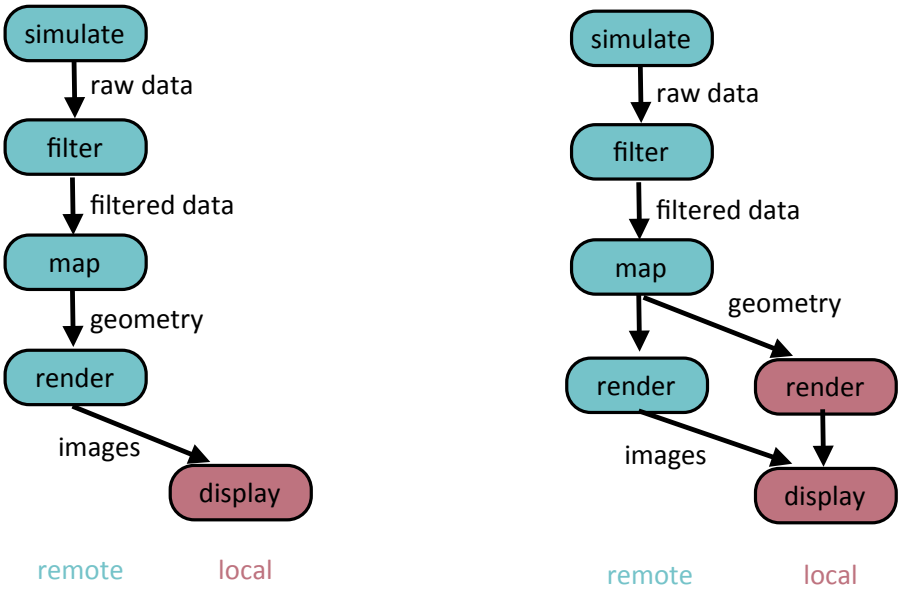

Figure 1: a pure remote vs. a remote hybrid rendering workflow.

Figure 1 illustrates the differences between a pure remote rendering and a remote hybrid rendering workflow. Please refer to section 5.4 for an illustration of this process by a concrete example.

4 Implementation of Remote Hybrid Rendering

4.1 Implementation Details

The client for remote hybrid rendering is implemented as the plug-in VncClient to OpenCOVER [5], the virtual reality renderer of the visualisation system COVISE [6] and its data-parallel successor Vistle [14], which is currently in development.

For the RHR server, there are two implementations: one is realized as the plug-in VncServer for OpenCOVER. As such, it is compatible with COVISE and Vistle. The other implementation is a light-weight rendering module for Vistle, which uses the CPU for interactive ray casting.

Both server implementations can make use of a cluster of rendering resources by means of sort-last parallel rendering. The IceT compositor framework [15] is responsible for depth compositing. While compositing is an integral part of the Vistle ray caster, OpenCOVER uses the plug-in CompositorIceT for this purpose. As the ray caster does not depend on GPU support, it will allow for scalability experiments with higher node counts: the largest GPU cluster that we have available comprises 20 nodes, while we have thousands of CPU nodes.

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

4.1.1 VncClient Plug-in

While viewing the colour image generated by a RHR server is possible with any VNC viewer, taking advantage of the compositing of local and remote data requires a specially adapted VNC client. The VncClient plug-in for OpenCOVER is such a 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.

4.1.2 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.

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].

Colour image data is compressed using VNC's possibilities by LibVNCServer, for compressing depth data the snappy entropy compressor library is used [8]. On CUDA capable GPUs, we implemented a method for lossy depth compression as described in D5.3.4 [5], which operates orthogonal to the entropy encoding.

4.1.3 CompositorIceT Plug-in

For data-parallel rendering on OpenGL supported hardware, the VncServer plug-in is augmented by the plug-in CompositorIceT. It is used to compose a complete image

from renderings of all parts of decomposed data sets. This requires 2.5D image data (colour and depth) for each partial image. The final image is obtained by selecting the colour of each pixel from the partial image with the smallest corresponding depth value, i.e. that is closest to the viewer. This step is executed by the IceT compositor, a library which provides highly efficient algorithms for combining images over MPI.

4.1.4 Vistle Ray Caster

In order to scale with the number of nodes on systems that do not provide OpenGL support, a CPU based data-parallel ray casting render module for Vistle has been implemented. It is based on the ray tracing framework Embree [12], which makes use of the SIMD units of CPUs to reach interactive frame rates. The sole purpose of this render module is to provide the remote hybrid rendering service. Because of this, a rather light-weight implementation was possible, as most of the application logic resides in the RHR client. The sort-last compositing is again realized with IceT. This is the only server capable of generating images for multiple views simultaneously, as it is necessary for e. g. CAVEs.

4.2 Obtaining the Software

4.2.1 COVISE and OpenCOVER

The three plug-ins are available as open source, while OpenCOVER is not. For compiling and using the plug-ins, at least Subversion revision 25946 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 three 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.

4.2.2 Vistle

The Vistle ray caster is a part of Vistle and as such is available on GitHub in the public Vistle Git repository [14]. Please refer to the file README.md available there for further instructions.

4.3 Changes Since D5.3.3 [4]

Since the last delivery of the software, the following notable improvements have been achieved:

- dynamic resizing of server framebuffer to match client window size,
- improved compression algorithms for depth data and a GPU-based implementation to provide faster read-back and lower transmission overhead,
- a CPU based data-parallel renderer providing the potential of scaling to high node counts, as it is independent of GPUs,
- synchronized handling of multiple views as necessary for CAVEs and tiled displays,
- the effects of high rendering times and latencies, especially for high display resolutions, are mitigated by reprojecting the 2.5D data of previous frames already available in the client according to the current view point,
- parallelizing the compression step allows for better overlap of rendering and compression as well as communication and helps to reduce rendering delays,
- interoperability with server-side parallel rendering as described in D5.3.1 [2] by integrating the IceT library for compositing, for both CPU and GPU based rendering.

5 Using the Software

5.1 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 OpenCOVER. 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 plug-in 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].

5.2 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 analysing the data.

The final subsection of this section describes such a work flow.

5.3 Vistle

Vistle provides two options for rendering on the remote cluster: if there are GPUs available, it is possible to use OpenCOVER with the VncServer and CompositorIceT plug-ins. Additionally, one can use the CPU based ray caster for in-situ visualisation on systems without GPUs. The parameters of the ray caster are configured with the Vistle user interface.

5.4 Usage Example

5.4.1 COVISE and OpenCOVER as RHR Server

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 2. 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.

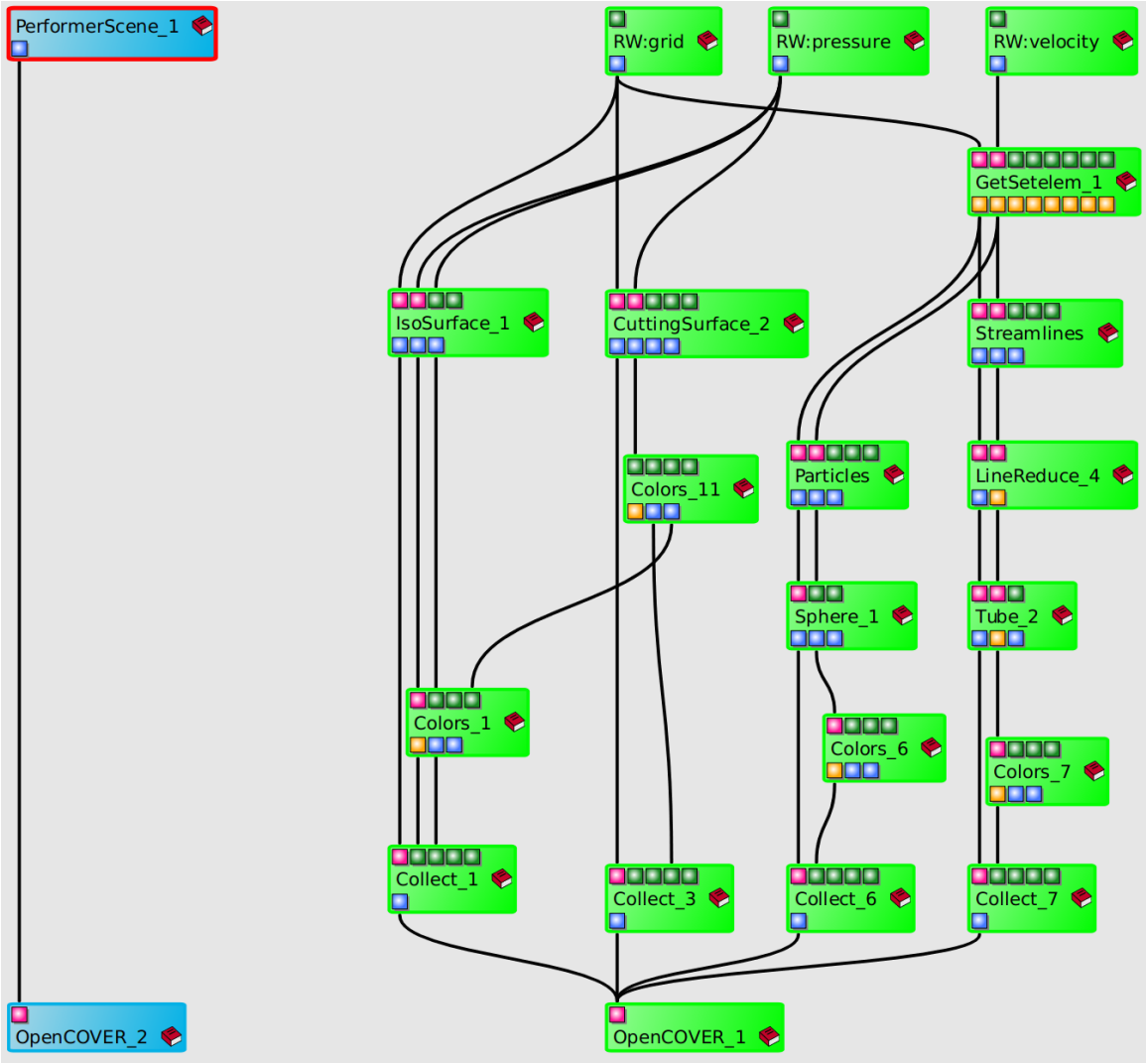

Figure 2: data flow network for remote hybrid rendering, local modules blue, remote modules green.

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 3 shows the resulting image rendered by the modules labeled *OpenCOVER_1* in Figure 2.

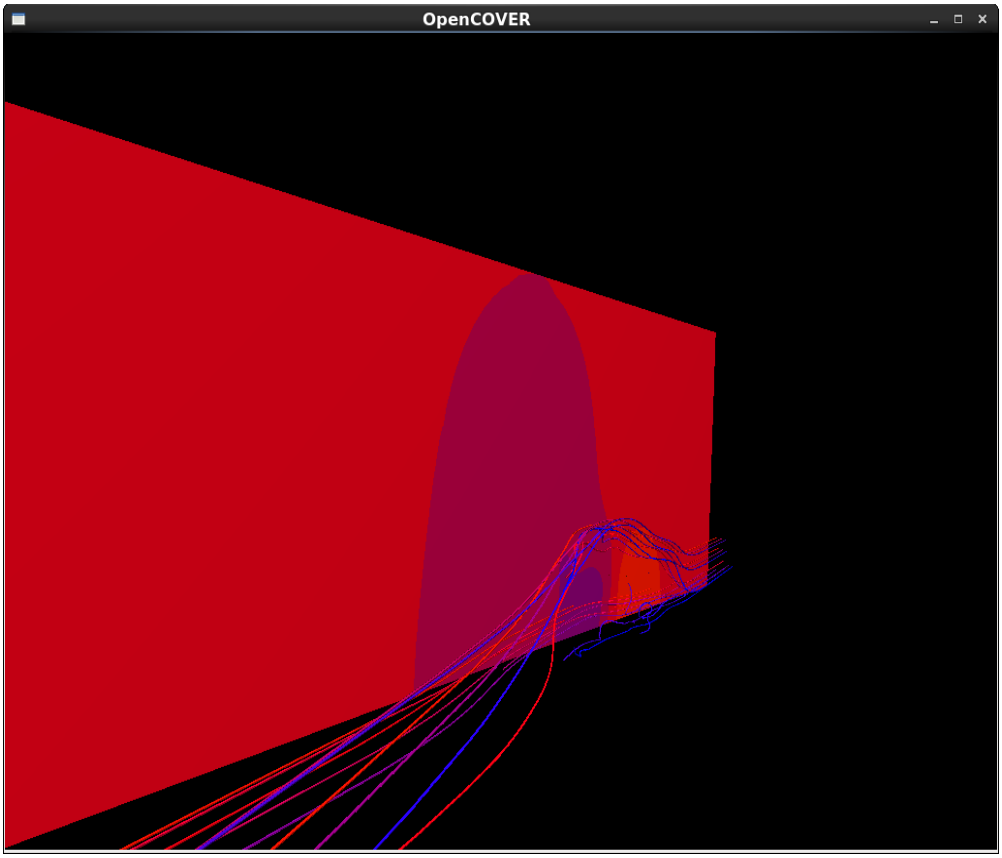

Figure 3: remotely rendered flow visualization.

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 4 shows the corresponding image produced by the module labeled *OpenCOVER_2*.

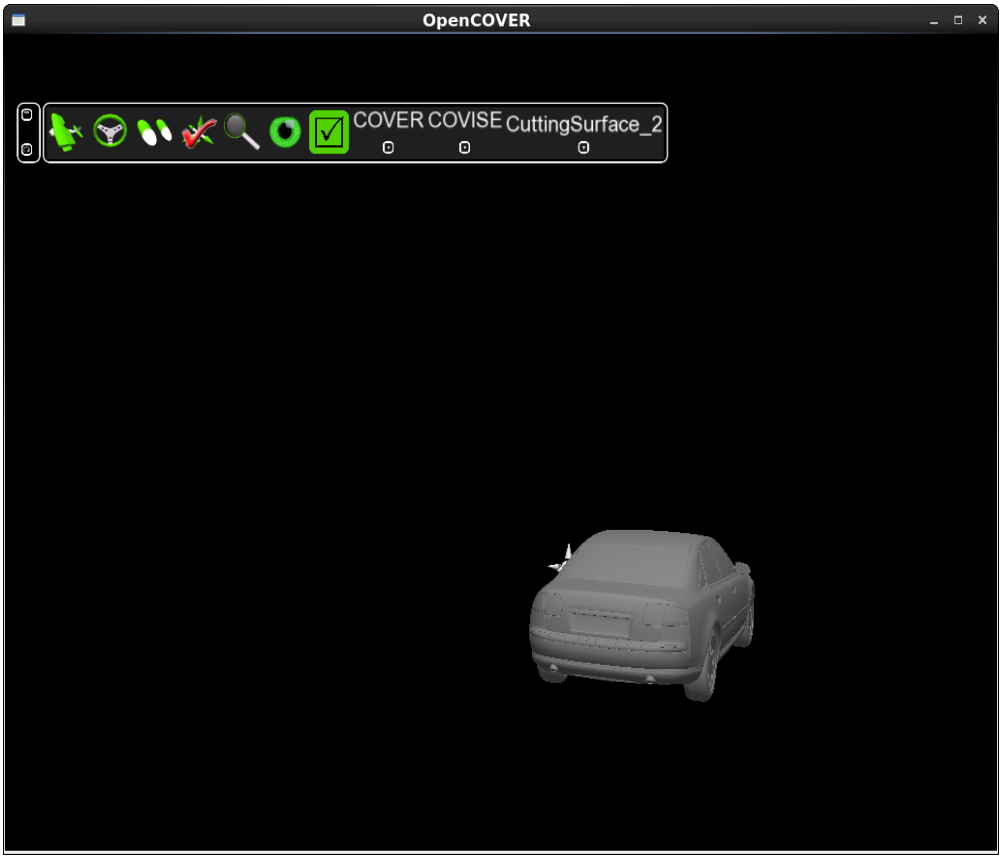

Figure 4: locally rendered context information.

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 5.

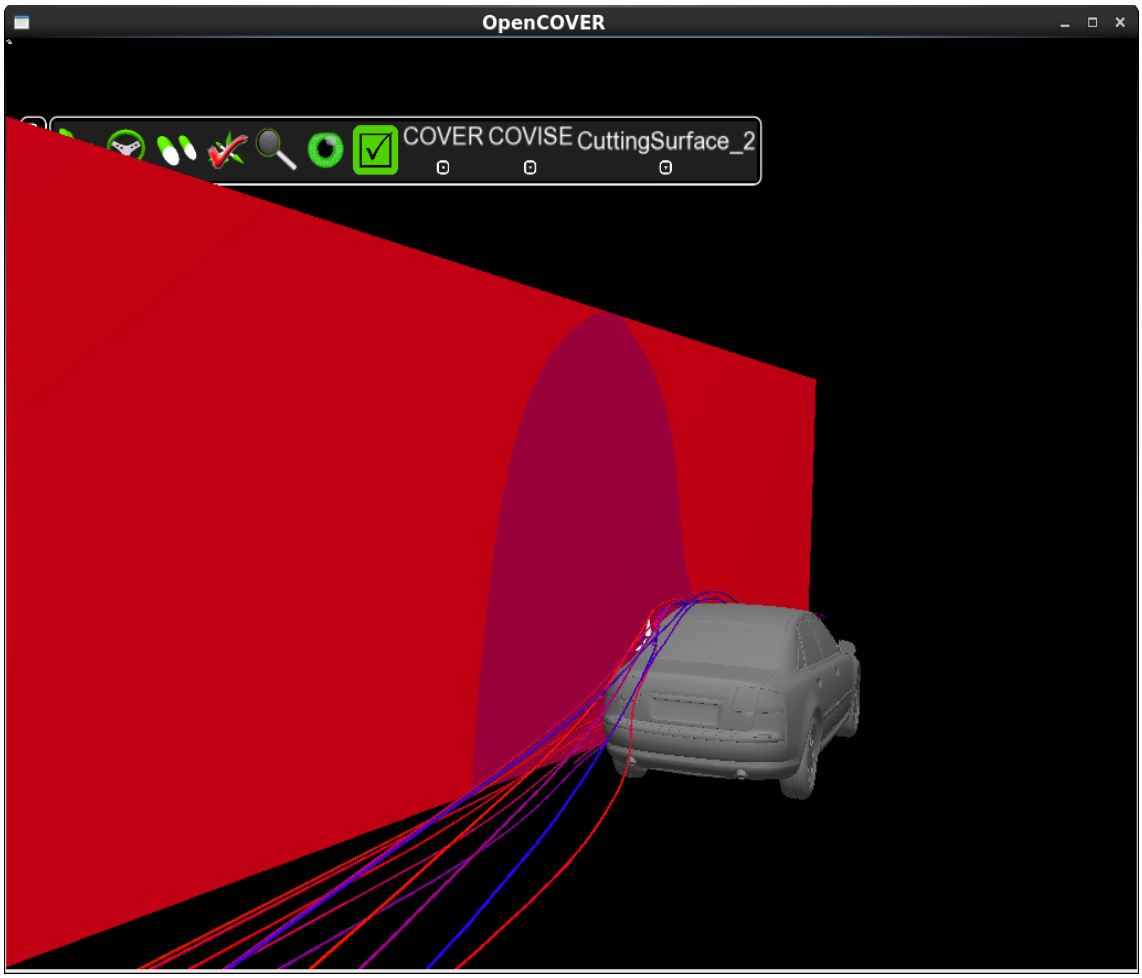

Figure 5: final image resulting from compositing local and remote contributions.

Figure 2 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 5.1.

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 makes this visualization distinguishable from a purely local visualization.

5.4.2 Vistle with Ray Caster as RHR Server

For testing the data-parallel ray caster, we used the results of an OpenFOAM simulation of the pump turbine test case conducted by IHS (Institute of Fluid Mechanics and Hydraulic Machinery at USTUTT) on 128 processors. Iso surfaces of pressure and turbulent eddy viscosity (nuSgs) have been used to visualize the turbulence structures in the diffusor after the runner. Figure 6 depicts the Vistle work flow to obtain the visualization in Figure 7. The visualization modules run in parallel on 160 processors on our 20 node visualization cluster. The nodes communicate via MPI, whereas OpenMP is used for parallelization within nodes. The local instance of OpenCOVER, which composites the remote visualization with its menus, is not shown in the work flow.

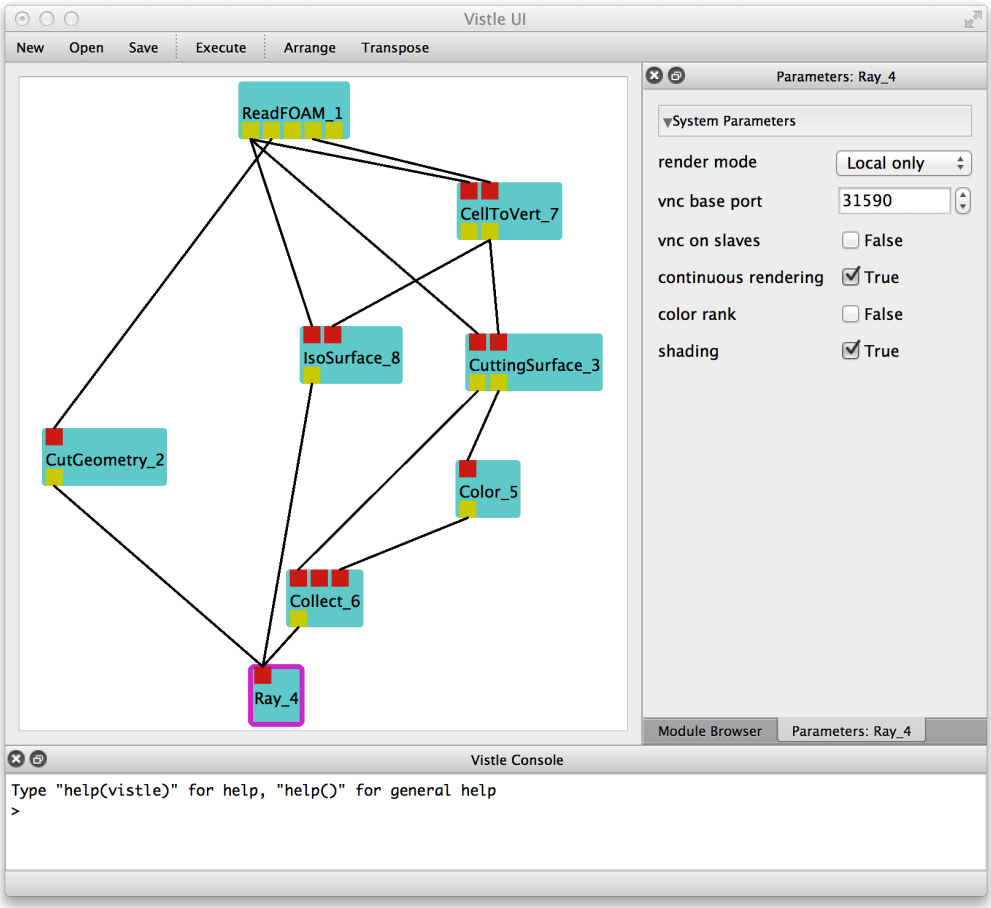

Figure 6: Vistle work flow for pump turbine visualization showing the parameters of the CPU ray caster.

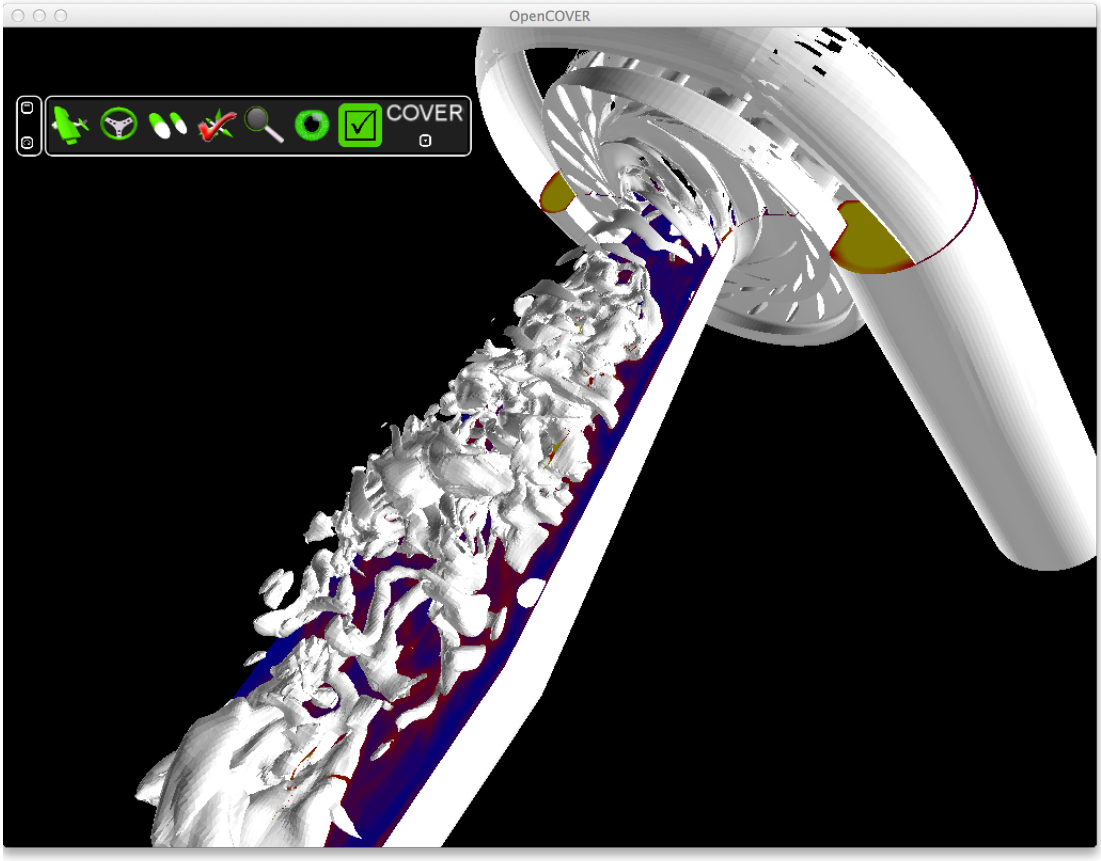

Figure 7: IHS pump turbine test case with cutting plane and iso surface of turbulent eddy viscosity (nuSgs) illustrating the turbulent flow in the diffusor after the runner.

Figure 8 shows just the bounding geometry and the iso surface, Figure 9 depicts the contribution of each node to the final image in an individual colour.

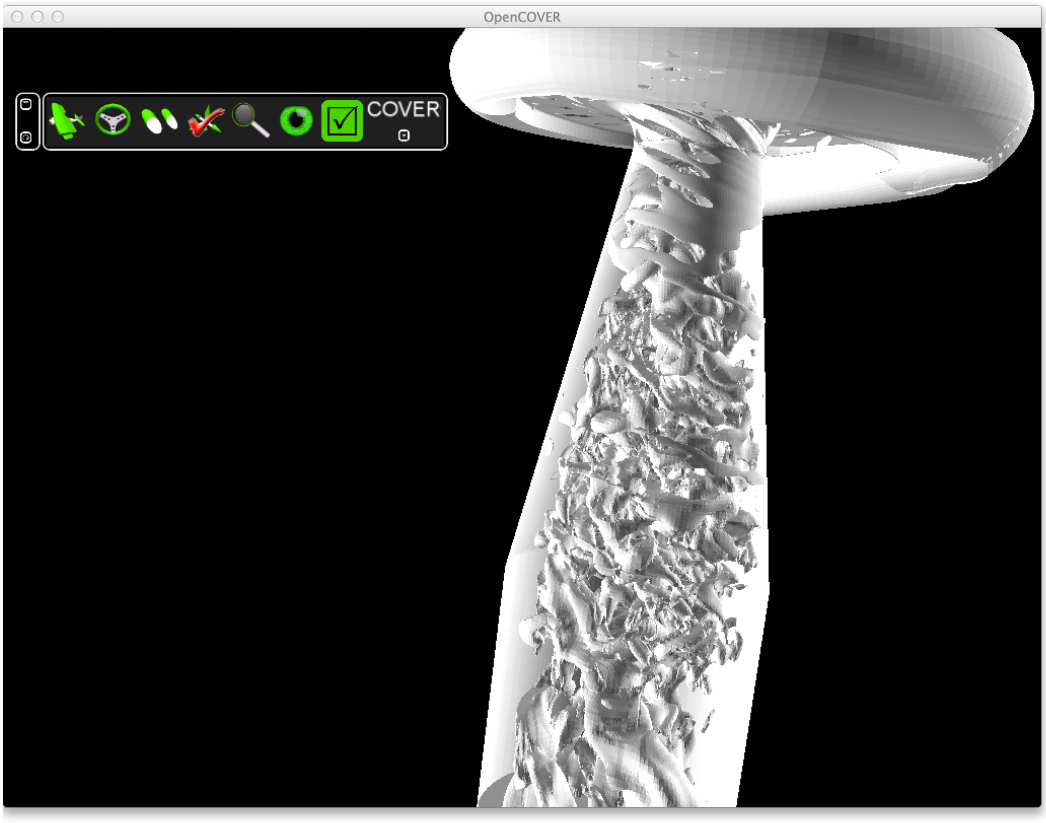

Figure 8: Iso surface and bounding geometry rendered in parallel on remote system together with local 3D menues.

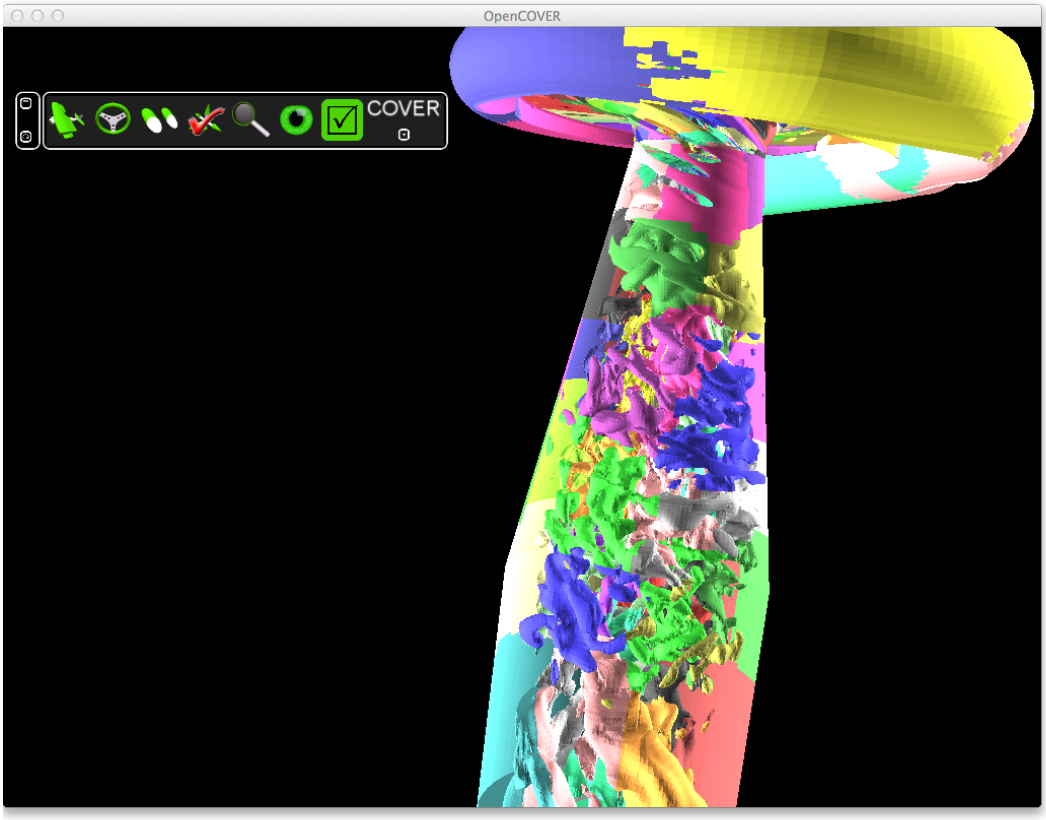

Figure 9: Contributions from each node are shown in different colours.

6 Future Work

Based on the experience gained with implementing and using the current tool for remote hybrid rendering, we propose that the software should be improved in the following ways:

- while the CUDA based lossy depth compression is a considerable improvement over only entropy based compression, bandwidth requirements and latency could benefit from further refinements of the compression algorithms for depth data,
- commonalities between the server-side implementations of RHR (OpenCOVER plug-in and Vistle ray caster) should be identified to reduce code duplication and to provide a basis for integrating RHR support into other rendering software.

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

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

[4] Aumüller, M.: CRESTA D5.3.3: Remote hybrid rendering: first prototype tool, Mar. 2013.

[5] Aumüller, M.: CRESTA D5.3.4: Remote hybrid rendering: revision of system and protocol definition for exascale systems, Sep. 2013.

[6] 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.

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

[8] 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.

[9] 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.

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

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

[12] S. Woop, L. Feng, I. Wald, and C. Benthin, “Embree ray tracing kernels for CPUs and the Xeon Phi architecture. ,” *SIGGRAPH Talks*, p. 44, 2013.

[13] M. Aumüller, “Remote Hybrid Rendering of Exascale Data in Immersive Virtual Environments,” presented at EASC, 2013.

[14] Vistle GitHub repository [Online], Available https://github.com/vistle/vistle, [Accessed: 01 Mar. 2014].

[15] K. Moreland, W. Kendall, T. Peterka, and J. Huang, “An image compositing solution at scale,” presented at the High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for, 2011, pp. 1–10.

[16] C. Wagner, M. Flatken, F. Chen, A. Gerndt, C. Hansen, and H. Hagen, “Interactive Hybrid Remote Rendering for Multi-pipe Powerwall Systems,” in *Virtuelle und Erweiterte Realität - 9. Workshop der GI-Fachgruppe VR/AR*, C. Geiger, J. Herder, and T. Vierjahn, Eds. Aachen: Shaker Verlag, 2012, pp. 155–166.