D5.3.2 – Remote hybrid rendering: protocol definition for exascale systems

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:	M12
Submission date:	22/10/2012
Project start date:	01/10/2011
Project duration:	36 months
Deliverable lead organization	Name of USTUTT
Version:	1.0
Status	Final
Author(s):	Martin Aumüller (USTUTT)
Reviewer(s)	Jens Doleschal (TUD), David Lecomber (ASL)

Dissemination level	
<PU/PP/RE/CO>	*PU - Public*

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	08/10/2012	First version of the deliverable	Martin Aumüller (USTUTT)
0.2	15/10/2012	Improve according to propositions by Jens Doleschal and David Lecomber	Martin Aumüller (USTUTT), Jens Doleschal (TUD), David Lecomber (ASL)
0.3	17/10/2012	Clarify reasons for protocol choice based on suggestions by David Lecomber	Martin Aumüller (USTUTT), Jens Doleschal (TUD), David Lecomber (ASL)
1.0	18/10/2012	Final version of the deliverable	Martin Aumüller (USTUTT)

Table of Contents

Index of Figures

1 Executive Summary

In this work we define a protocol for remote hybrid rendering. Remote hybrid rendering is used to access remote exascale simulations from immersive projection environments over the Internet. This protocol is used for the flow of communication between the head node of a visualisation cluster and the head node of a display system. The display system may range from a desktop computer to an immersive virtual environment such as a CAVE (Cave Automatic Virtual Environment). 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.

We start by evaluating the benefits of remote versus local rendering as well as describing the short-comings, which we will not be able to fully address within this work. We continue by establishing the requirements for such a protocol based on the intended use case as laid out in the description of work and the requirements of the co-design vehicles OpenFOAM [1] and HemeLB [2]. We evaluate existing protocols for similar purposes, mostly protocols for virtual framebuffers, according to our requirements. Because of its extensibility, its wide-spread use and the possibility of backwards compatibility of our system with existing clients we choose to base our protocol on the Remote Frame Buffer (RFB) protocol as used in all Virtual Network Computing (VNC) based systems. We define the extensions to the RFB protocol which are necessary to cover our use case, such as the handling of multi-touch and 6DOF input data, frame synchronisation across several display surfaces, encodings for 2.5D and opacity images as well as an image back-channel.

2 Introduction

The structure of this document is as follows: Section 3 shows the usefulness of remote hybrid rendering, the following section addresses the short-comings of this process. Section 5 summarises the setting in which the protocol defined in this document will be used. Section 6 establishes the requirements the protocol has to fulfil. Section 7 gives a short overview of existing similar protocols, before we lay out the protocol in section 8 and conclude with final remarks in the last section.

2.1 Purpose

The purposes of this deliverable are as follows:

- Establish requirements for a protocol enabling remote hybrid rendering on exascale systems, taking into account requirements from WP 5.1 and WP 5.2
- Study existing protocols for remote rendering and evaluate them according to the established requirements
- Define a protocol for remote hybrid rendering

2.2 Glossary of Acronyms

2.5D	Image data together with depth data
6DOF	6 degrees of freedom, usually position and orientation
CPU	Central Processing Unit
GPU	Graphics Processing Unit
HD	High Definition
JPEG	Joint Photographic Experts Group
POV	Point of View
RFB	Remote Framebuffer Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VNC	Virtual Network Computing
WAN	Wide Area Network
WP	Work Package
X11	X Window System Protocol Version 11
CAVE	Cave Automatic Virtual Environment

3 Motivation for Remote Rendering and Remote Hybrid Rendering

The co-design vehicle OpenFOAM as used by the Institut für Strömungsmechanik und Hydraulische Strömungsmaschinen at the University of Stuttgart for simulating the flow in an entire hydro turbine [3] produces huge amounts of output data. Based on the estimated requirement for a dependable simulation of approximately 1 billion nodes for the whole turbine, the size of a single time step is about ¼ TB. Storing a full turbine rotation with steps of one degree requires about 90 TB. Transferring that amount of data across a high-speed link (10 GigE) for off-line processing on a user workstation would take more than one day – and would require huge amounts of local storage and processing power. This shows that for exascale data the traditional way of transferring the post-processed geometry data to the display system for local rendering is not possible anymore. In comparison, streaming uncompressed HD-resolution (1920x1080 pixels) images at 30 frames/s for a whole day would require less than 15 TB of bandwidth – and the image the user is interested in is available immediately, not just after a lengthy preparational transfer. Additionally, this amount of data can be significantly reduced by employing image compression techniques without even incurring a visual loss. This technique of transmitting rendered images instead of post-processed data to the display system is called remote rendering. The much lower bandwidth and processing requirements of remote rendering allow for making the 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 [16]. 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.

As the above example of the turbine simulation shows, transferring the post-processed simulation data is not viable. To improve frame rate and reaction times, we will try to decouple interaction from network latencies as far as possible, but still without requiring the transfer of huge data amounts 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. In order to account for view point changes that have occurred after the remote image was rendered, it can be reprojected before the compositing step.

In the description of work [25], the remote hybrid rendering method was selected as the mechanism to make the interaction with exascale simulations possible across the Internet.

4 Limitations of Remote Hybrid Rendering

The incentive for employing remote hybrid rendering is to achieve low latency for interaction and especially in head-tracked environments – even when pure remote rendering is not possible because the network connection does not provide enough bandwidth or does not fulfil the latency requirements for immersive virtual environments.

Hence it seems reasonable to sacrifice rendering accuracy for rendering speed in the general case in order to build a reactive interactive system. Cases where accuracy might not be able to be guaranteed include:

- Transparency in rendered data without sufficient depth information makes it impossible to mix context information into the rendered image at the correct depth. This might be the case when dealing with ray castings from HemeLB, as the ray along which data was accumulated into a single colour value might be partially occluded by the local overlay geometry – and then it is impossible to recover the contribution from in front of the occluding geometry.
- High latency makes it necessary to take recourse to image based methods, where 2.5D data is warped according to changed view points in order to provide timely updates in a head tracked environment – but reprojected images are just an approximation of what would be visible from the updated POV.

However, we will always allow the user to request a rendering with full accuracy, albeit with increased communication cost and at possibly non-interactive frame rates.

5 Considerations for Exascale Systems

The environments we try to adapt our remote hybrid visualisation system to comprise the following parts:

- a remote exascale compute resource,
- possibly a remote visualisation cluster, tightly coupled to the compute resource,
- a local display system.

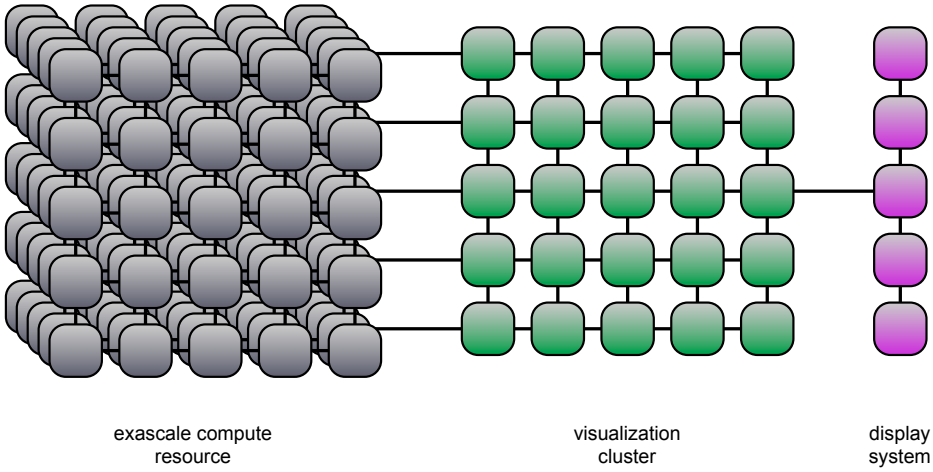

Figure 1: Typical network topology for a remote visualization task

In some cases, e. g. when there are GPUs inside each node of the exascale system, the compute system and the visualisation system might be the same resource and the GPUs might be used for both simulation and visualization. For all other cases, we assume a high-bandwidth low-latency link of a quality comparable to the exascale cluster interconnect between the compute and visualisation system. The network connection between the remote visualization cluster and the display system will provide a considerably lower bandwidth and higher latency. While it is desirable to have a higher quality link between the visualization and display systems, this will not always be possible in the case where remote hybrid rendering is used, as this connection will usually be across the Internet.

The network infrastructure might allow for direct connections from each node of the visualisation cluster to each node of the display system, but in the general case the network topology or firewalls might prohibit this. Hence, our system is designed to cope with a point-to-point connection between the head node of the visualisation cluster and the head node of the local display system.

The display system may be a traditional desktop computer. But the focus of this work is to enable access to remote exascale visualizations from within immersive projection systems. These are distinguished from desktop systems by:

- input devices which record their position and orientation and input methods which exploit this additional information,
- tracking of the user's head position and continuously updating the rendered image according to the changing point of view (POV),
- 3D stereoscopic imagery, where each eye is presented with an image that is adapted to its position,
- multiple display surfaces, which are used for enhancing the resolution (e. g. in powerwalls, where several screens are tiled in one plane to form a larger display area) or to surround the viewer with images (e. g. in a CAVE [17], where the sides of a cube around the viewer are used as projection surfaces).

Sort-last [18] has been selected [19] as the method for parallelising the render process. This means that flat pixel images as present in a framebuffer are the result of the rendering phase. The available data for the remote rendering solution is one colour

value including opacity per pixel together possibly one depth value. Remote sort-last parallel rendering in a system with the described network topology provides the best performance if compositing happens on the visualisation cluster, as this saves bandwidth on the slower link between the visualisation and display systems. The requirement of a point-to-point connection between the head nodes of the visualisation cluster and the local display system makes it necessary that the composition of the rendered image data happens on the visualisation cluster.

Integrating the remote rendering facility with the application might enable further optimisations, as the application has more knowledge about which data is important. The application might chose to update the significant regions more often or at lower compression level with higher image fidelity. However, this does not affect the protocol, if only it is capable of sending sub-regions or images at different compression levels.

6 Protocol Requirements

The purpose of the protocol for remote hybrid rendering is to define the communication between the visualisation cluster and the local display system. I. e., the protocol for hybrid remote rendering connects the rendering stage to the display stage of the post-processing phase of the visualisation pipeline. The rendering stage employs parallel sort-last rendering and produces images with colour, opacity and depth information for each pixel. Hence, it is mostly independent of the applications that run remotely. Application independence is also a goal of this protocol, as it should be usable across multiple applications. Hence, we do not take solutions that require tight coupling with the application, such as IBRAC [20] or as implemented in Visapult [21] into account. Only the compositing stage and the handling of transparency in the renderer stage have an impact on the protocol requirements, as these determine what kind of image data has to be transmitted.

The protocol merely defines how the data is to be communicated and interpreted. It does not define how the transmitted data is to be processed. In particular, even though a certain protocol might not have been developed for an algorithm that reprojects pixels, this does not mean that it is not applicable or extendable for that use case.

we list how this requirement appears in the table at the end of the section discussing several virtual framebuffer protocols.

6.1 Feature Requirements

Based on the planned use case of driving desktop systems, tiled display walls and CAVEs with the remote hybrid rendering protocol, the following input data has to be handled by the protocol:

- keyboard input *(keyboard input)*
- 2D mouse input *(2D mouse input)*
- input from 6DOF devices, such as hand and head tracking devices or a space mouse *(6DOF devices)*
- multi-touch input – the protocol must allow for tracking touches across multiple frames, as the client has more knowledge to achieve this more easily *(multi-touch input)*

The topology of the envisioned compute and visualisation system means that it is more efficient that the compositing happens within the visualisation cluster. However, rendering context information locally requires a final compositing step in the display system. Depending on the context information to be shown and the rendered data, this might require depth *(depth channel support)* or opacity *(alpha channel support)* in addition to the colour information for each image pixel. And even with that information, it might only be possible to approximate the correct rendering. For those cases, an option to switch to slower but accurate rendering is necessary. This will require sending the context data overlay to the visualisation cluster before rendering each frame, such that it can be taken into account during rendering and compositing *(image back channel)*.

It is sufficient to serve one display system at a time. But such a system might possibly consist of several display surfaces *(multi-surface support)*, each of which may be a stereographic display *(3D stereo support)*. Updates to different display surfaces have to be synchronised in order to enable correct 3D stereoscopy across all surfaces *(frame barrier support)*. This might incur longer latencies, when the images for all tiles are not available at the display client at the same time, but this synchronisation is vital for immersive display environments.

The protocol should support disconnecting from a running visualisation session and reconnecting to it from a possibly different and differently configured (different screen

sizes, different number of screens, different screen orientations, different input modalities) display system *(connection re-establishment)*. This also enables error resilience, as a session can be restarted after crashes.

In order to handle common network configurations where it is not possible to establish arbitrary connections between nodes in the visualisation cluster to nodes in the display system, we require that communication be channelled through a render server head node and a display client head node. At the same time, this simplifies the protocol, as this means that redistributing events and pixel data is the head nodes' task and beyond the scope of this protocol. This also ensures scalability of the protocol from desktop systems over powerwalls to multi-screen immersive environments with stereographic rendering. We consciously sacrifice the optimisation possibility of skipping a communication step by sending image tiles directly from the visualisation node that generates the image to the display node which is responsible to show this tile.

6.2 Performance Requirements

The protocol should not generate significant communication overhead. Network round trips, e. g. waiting for acknowledgement of successful delivery of messages, have to be avoided in order to guarantee good performance *(low overhead)*. User input data has to be prioritised over bulk data transfers, such as rendered pixels.

Occasionally, not the full image plane is filled with rendered data. In these cases it should be possible to only send sub-images in order to save bandwidth *(sub-image transmission)*.

In order to be able to balance visual accuracy against performance, the protocol has to allow for different encodings and compression algorithms, and for accommodating changing network circumstances (bandwidth and latency variations), these have to be switchable at run-time *(adaptability to network link quality)*. Compression should not visibly decrease image quality for both line drawings and images with huge amount of gradients, e. g. from volume rendering *(high image fidelity)*.

For local area connections, TCP based protocols have proven superior, whereas in wide area networks, UDP based protocols seem to have an advantage [4]. We expect the principal use case to be from within local area networks or within networks providing a similar connection quality, such that we prefer TCP over UDP.

The protocol should not hinder the off-load of suitable tasks, like image compression or decompression, to accelerators, such as GPUs. This mostly concerns the image codecs to be used. Hence, we want to allow for the easy addition of new codecs. This also allows for using codecs adapted to the requirements of the processing of the transmitted data on the display system, e. g. when the 2.5D image is reprojected [22]. Additionally, this allows the system to profit easily from algorithmic improvements available in new video codecs, such as H.265 [23], as soon as GPGPU solutions for real-time compression at high resolutions are available.

6.3 Security Requirements

As sensitive data might be transmitted, the users connecting to the remote rendering server have to be authenticated *(user authentication)*. Sometimes, the input data might contain passwords and other secret information, so the channel from display system to render system should be encryptable *(input data encryption)*. Because of the computational costs entailed by encrypting all the streamed image data, encryption of the channel from the render server to the display client is optional *(image data encryption)*.

6.4 Other Requirements

An existing protocol based on an open standard would be preferable *(standardized)*, but it has to be documented in order to be usable for our purposes *(documented)*.

The protocol should be extendable in order to accommodate future needs. It is of special importance that new image codecs can be integrated easily, in order to be able to make use of better image or video codecs *(easy extensibility for new codecs)*, other algorithmic improvements or added hardware compression acceleration *(extensibility)*.

The protocol has to be interoperable in heterogeneous environments as arbitrary display systems might want to connect to the render server *(platform independence)*.

The protocol should keep the number of simultaneous network connections to a minimum; the establishment of a connection should be possible from client to server and vice versa in order to cater for all possible circumstances *(firewall friendliness)*.

7 Existing Protocols

The main task of the protocol to be defined is transmitting remotely rendered pixel data to the display system. Hence, we only consider protocols where the primitives for transmitting image data are pixels, not higher-level primitives such as lines or glyphs from a font. Because of this, we do not take widely used protocols such as RDP (Remote Desktop Protocol, as used for Windows desktop sharing) [5] or X11 [6] into consideration.

7.1 RFB (VNC)

The Remote Framebuffer Protocol (RFB) is widely used across many platforms as part of every VNC installation. The client sends keyboard and mouse input data to the server and requests updates to screen regions that it wishes to draw. The server responds with image tiles containing the requested regions, although the responses to several update requests can be coalesced into a single update. The protocol defines various encodings for pixel images, some of which reference previous framebuffer states. The server does not push updates to the client, but delivers updates only upon request from the client. The protocol specification is open [7] and includes a mechanism for extensions. There are many implementations available, both open and closed source, most of them are interoperable, but most often not at full performance. Because of its wide spread adoption, it has become a de facto standard.

7.2 Independent Computing Architecture (ICA)

Independent Computing Architecture (ICA) [8] is a proprietary protocol developed by Citrix for more than 15 years [9]. Communication happens across a single TCP connection between client and server. ICA can send both pre-rendered pixel data as well as higher level drawing primitives. The level of compression is adapted dynamically to the connection quality. Transfers are prioritised into different channels in order to improve interactivity. To our knowledge, there is no documentation available and it is only implemented by Citrix.

7.3 HP Remote Graphics Software

HP Remote Graphics Software [10] is a desktop sharing software based on a proprietary protocol, which transmits pixel data to the viewer. All data is sent encrypted. It uses a proprietary image codec named HP3, which encodes image areas differently based on its contents. This allows high fidelity for both line graphics and photographic images. We do not know of any documentation of this protocol or implementation by parties other than HP.

7.4 NX

The NX [11] protocol is used to tunnel the full X11 protocol over low-bandwidth high-latency links. For minimising client-server round trips, it employs server- and client-side proxies. Pixmaps (bitmap images) can be compressed using the lossy JPEG or with other algorithms. Large transfers are split into smaller pieces and are sent with a lower priority than smaller transfers, in order to increase interactivity. There are closed and open implementations of the protocol.

7.5 VirtualGL

The VirtualGL [12] protocol is only capable of transferring pixel data from server to client. It is used to augment the X11 protocol in order to off-load 3D rendering to the server and enable compressed transmission of the rendered images. It supports transmission of stereographic images. The protocol only seems to be used within the VirtualGL project, but it has on open-source implementation.

7.6 THINC

THINC [13] is a system for desktop virtualisation, which intercepts the drawing commands at the device driver layer. The basic protocol has similar features as VNC's RFB protocol, but it also supports an opacity channel. Later, support for some higher-level primitives has been added in order to better support video playback [14]. The source code for THINC is available.

7.7 SPICE

SPICE is a remote rendering protocol used commercially in Red Hat Enterprise Virtualization for Desktops [15]. Depending on the capabilities of the display client, more or less work is off-loaded to the server. The SPICE protocol is also implemented at the device driver level. For pixel transfers, it also supports an opacity channel. Several encodings are available. But it also supports higher-level primitives such as paths and glyphs. The SPICE X server implementation by Red Hat is available as open source. There is also in implementation for Windows.

7.8 Appliance Link Protocol (Sun Ray)

The Appliance Link Protocol (ALP) is a proprietary protocol used for communication between the Sun Ray Server Software and a Sun Ray client. It uses a TCP connection for session communication and several UDP ports for graphics, audio and other binary data. The UDP port numbers are determined by the server and communicated to the client over TCP. It is optimised for WAN communication. To our knowledge, there is no documentation available and it is only implemented by Sun.

7.9 Overview

A + means that the requirement is met, a – that not, +/- that is met partially (e. g. the documentation consists of an open source implementation). An empty field means that we do not possess this information, either because we did not find any documentation about this feature or did not look into the implementation to assess the availability of this feature.

Requirement	***Importance***	***RFB***	***ICA***	***HP***	***NX***	***Virtual GL***	***THINC***	***SPICE***	***ALP***
Keyboard input	Mandatory	+	+	+	+	-	+	+	+
2D mouse input	Mandatory	+	+	+	+	-	+	+	+
6DOF devices	Mandatory	-				-			
Multi-touch input	Mandatory	-				-			
Alpha channel support	Mandatory	-				-	+	+	
Depth channel support	Mandatory	-	-	-	-	-	-	-	-
Multi-surface support	Mandatory	-							

3D stereo support	Mandatory	-				+			
Frame barrier	Mandatory	-				+			
Image back channel	High	-				-			
Connection re-establishment	Mandatory	+			+	+			
Adaptability to network link quality	High	-	+			-			
Low overhead	Mandatory	+	+		+	+			
Sub-image transmission	Mandatory	+				+			
High image fidelity	Mandatory	+/-	+	+	+/-	+/-			
User authentication	Mandatory	+	+		+	-			
Input data encryption	High	-	+	+	+	n/a			
Image data encryption	Low	-		+	+	-			
Extensibility	Mandatory	+				-			
Easy extensibility for new codecs	Mandatory	+				+			
Platform independence	Mandatory	+	+		+	+			
Firewall friendliness	Mandatory	+	+/-		+/-	+/-			-
Standardized	Low	+/-	-	-	-	-	-	-	-
Documented	Mandatory	+	-	-	+/-	-	+/-	+/-	-

8 Protocol Definition

8.1 Possible Choices

Based on the desire to maintain compatibility with existing software and the wish to use proven solutions when possible, our choices are, in order of most to least favourable:

1. use an existing protocol that fulfils all the requirements as-is,
2. use an existing protocol that provides an extension mechanism powerful enough to implement all necessary features,
3. create a new protocol that builds upon an existing protocol, or
4. define a new protocol entirely unrelated to existing protocols.

We do not know of any protocol fulfilling all the requirements. Hence, the second option is our best choice.

The RFB protocol is extensible and allows the incorporation of the required features with low overhead. By basing the remote hybrid rendering protocol on RFB, we allow for universal access to remote visualisation resources by enabling the huge multitude of existing VNC clients to interact with remote hybrid rendering servers, albeit with limited features and performance. Hence, we propose to extend the RFB protocol for our purposes. However, as we do not expect remote hybrid rendering clients to connect to plain VNC servers, we will not cater for that use case.

Compared to SPICE, RFB has the advantage of being in wide-spread use, it is simpler to implement than NX, VirtualGL does not have the required features, THINC is still in the research stage, and all the other protocols would require reverse engineering. Hence, our choice is to extend the RFB protocol. The following section lays out the required extensions to RFB.

8.2 RFB Protocol Extensions

In order to meet the requirements, we propose the following extensions to the RFB protocol. The byte level wire protocol will be determined when the extensions are implemented.

8.2.1 3D Stereo Rendering

Additional image codecs for supplying images for both the left and right eye for one display surface will have to be implemented.

8.2.2 Multiple Display Surfaces

Support for handling multiple display surfaces can be added by providing a coordinate mapping for each surface to the available 65536x65536 pixel coordinate space. The protocol has to be augmented for establishing this mapping.

8.2.3 Frame Barrier

In order to advance the display to the next rendered frame synchronously on all display surfaces and for both the left and right eye, the server has to send an event when a frame is fully transmitted. This should include a frame counter and a time stamp.

8.2.4 Multi-Touch Input Data

As the display client has the most knowledge about previous touch events, we make it the display client's responsibility to track touches across frames. Hence, the protocol has to support transmitting this information as well as the current position and shape of touch points. The time stamp or frame counter of the image visible at the time of the input event should be included in the data returned to the server. RFB's `PointerEvent` will serve as a template for this extension.

8.2.5 6DOF Input Device Data

The protocol shall allow for multiple 6DOF inputs and for defining roles for those, e.g., “head of primary viewer” or “dominant hand”. Information about additional input data such as button press states should be communicated together with the 6DOF data in order to be able to handle these at the correct position. The frame time stamp and counter should be returned as well. RFB’s `PointerEvent` will serve as a template for this extension.

8.2.6 Server Controlled Framebuffer Updates

In order to minimise latency, rendered frames should be sent to the client as soon as they are available. The client has to be able to request continuous framebuffer updates from the server, i. e. the communication flow has to become more asynchronous than with standard RFB. For that purpose, TigerVNC [24] already has an extension containing an `EnableContinuousUpdates` message, this will be reused.

8.2.7 Encodings for Depth and Transparency Data

The available image encodings shall be augmented by appropriate codecs for transparency and depth data, in order to composite context information display with the visualisation image on the client.

8.2.8 Efficient Image Codecs

Compression quality can be enhanced by employing efficient image and video codecs. The available image codecs shall be augmented by video streaming codecs for which GPGPU implementations are available.

8.2.9 Image Data Back-Channel

In order to be able to generate images where the context information is correctly composed with the visualisation data, the rendering of the context information together with corresponding depth and opacity data has to be made available to all visualisation nodes. The `FrameBufferUpdate` message, which is normally sent from server to client only, will be used for that purpose.

9 Final Remarks

In the requirements section and the definition of the protocol we listed features which are essential or desirable for such a protocol for remote hybrid rendering. But this does not mean that the prototype tool, which is to be implemented as part of deliverable 5.3.3 and 5.3.5, will include all of them. We will select a subset of these features, which demonstrates the general feasibility of such a system.

10 References

[1] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby, “A tensorial approach to computational continuum mechanics using object-oriented techniques,” *Computers in physics*, 1998.

[2] M. D. Mazzeo and P. V. Coveney, “HemeLB: A high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries,” *Computer Physics Communications*, vol. 178, no. 12, pp. 894–914, Jun. 2008.

[3] “Requirements of WP6 (OpenFOAM) to WP5 (Pre- and Postprocessing),” CRESTA, Jan. 2012.

[4] B. Jeong, J. Leigh, A. Johnson, L. Renambot, M. Brown, R. Jagodic, S. Nam, and H. Hur, “Ultrascale Collaborative Visualization Using a Display-Rich Global Cyberinfrastructure,” *Computer Graphics and Applications, IEEE*, vol. 30, no. 3, pp. 71–83, 2010.

[5] “Remote Desktop Protocol,” *Microsoft*. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/aa383015(v=vs.85).aspx. [Accessed: Oct.-2012].

[6] R. W. Scheifler and J. Gettys, “The X window system,” *ACM TOG*, vol. 5, no. 2, Apr. 1986.

[7] T. Richardson, “ The RFB Protocol,” *realvnc.com*, 2010. [Online]. Available: http://www.realvnc.com/docs/rfbproto.pdf. [Accessed: 06-Sep.-2012].

[8] J. Harder and J. Maynard, “Technical Deep Dive: ICA Protocol and Acceleration,” *citrix.com*. [Online]. Available: http://www.citrix.com/site/resources/dynamic/additional/ICA_Acceleration_0709a.pdf. [Accessed: Oct.-2012].

[9] T. Valovic, “The Coming Display Protocol Wars -- Virtualization Review,” *virtualizationreview.com*, Jan.-2009. [Online]. Available: http://virtualizationreview.com/articles/2009/05/01/display-protocol-wars.aspx. [Accessed: Oct.-2012].

[10] “Advantages and implementation of HP Remote Graphics Software,” *h20331.www2.hp.com*, Apr.-2009. [Online]. Available: http://h20331.www2.hp.com/Hpsub/downloads/RGS_WP_April09.pdf. [Accessed: 06-Oct.-2012].

[11] S. Regis, “Introduction to NX Technology,” *nomachine.com*, Jul.-2009. [Online]. Available: http://www.nomachine.com/documents/pdf/intr-technology.pdf. [Accessed: 05-Oct.-2012].

[12] D. R. Commander, “VirtualGL: In Depth Background,” *virtualgl.org*. [Online]. Available: http://www.virtualgl.org/About/Background. [Accessed: Aug.-2011].

[13] R. Baratto, L. Kim, and J. Nieh, “THINC: a virtual display architecture for thin-client computing,” *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, Oct. 2005.

[14] R. A. Baratto, “THINC: A Virtual and Remote Display Architecture for Desktop Computing and Mobile Devices,” systems.cs.columbia.edu, 2011.

[15] “Spice remote computing protocol definition v1.0,” *spicespace.org*. [Online]. Available: http://www.spicespace.org/docs/spice_protocol.pdf. [Accessed: 07-Oct.-2012].

[16] 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.

[17] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti, “Surround-screen projection-

based virtual reality: the design and implementation of the CAVE," *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, p. 142, 1993.
[18] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs, "A sorting classification of parallel rendering," *Computer Graphics and Applications, IEEE*, vol. 14, no. 4, pp. 23–32, 1994.
[19] F. Niebling, J. Hetherington, and A. Basermann, "D5.3.1 – Remote hybrid rendering: analysis and system definition for exascale systems," CRESTA, Mar. 2012.
[20] I. Yoon and U. Neumann, "IBRAC: Image-Based Rendering Acceleration and Compression," *Computer Graphics Forum*, 2000.
[21] W. Bethel, B. Tierney, J. Leigh, D. Gunter, and S. Lau, "Using high-speed WANs and network data caches to enable remote and distributed visualization," *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM*, Nov. 2000.
[22] D. Pajak, R. Herzog, E. Eisemann, K. Myszkowski, and H.-P. Seidel, "Scalable Remote Rendering with Depth and Motion-flow Augmented Streaming," *Computer Graphics Forum*, vol. 30, no. 2, pp. 415–424, 2011.
[23] G. J. Han, J. R. Ohm, W.-J. Han, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, no. 99, p. 1, 2012.
[24] *tigervnc.org*. [Online]. Available: http://www.tigervnc.org. [Accessed: 15-Oct.-2012].
[25] "Collaborative Research into Exascale Systemware, Tools and Applications - Annex I: Description of Work", CRESTA, Jun. 2011.

