D5.3.1 – Remote hybrid rendering: analysis and system 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:	M6
Submission date:	31/03/2012
Project start date:	01/10/2011
Project duration:	36 months
Deliverable lead organisation	USTUTT
Version:	1.2
Status	Final
Author(s):	Florian Niebling (USTUTT), James Hetherington (UCL), Achim Basermann (DLR)
Reviewer(s)	Mats Aspnäs (ABO), Uwe Kuester (USTUTT)

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

Copyright © CRESTA Consortium Partners 2011

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	17/02/2012	First version of the deliverable	Florian Niebling (USTUTT)
0.2	21/02/2012	Added section on raycasting	James Hetherington (UCL)
0.3	03/03/2012	Still missing: sections 4, 5.1, 7, summary	Florian Niebling (USTUTT)
0.4	05/03/2012	Added Section 7	Florian Niebling (USTUTT)
0.5	05/03/2012	Added more content to section 5.1	Florian Niebling (USTUTT)
0.6	06/03/2012	Proofread	Andreas Kopecki (USTUTT)
0.7	07/03/2012	Executive summary	Florian Niebling (USTUTT)
1.0	09/03/2012	Revision	Achim Basermann (DLR)
1.1	16/03/2012	Incorporated reviewer's comments (Uwe Kuester, USTUTT)	Florian Niebling (USTUTT)
1.2	19/03/2012	Incorporated reviewer's comments (Mats Aspnäs, ABO)	Florian Niebling (USTUTT)
1.2	22/03/2012	Check of the final version after internal review	Achim Basermann (DLR)

Table of Contents

Index of Figures

Copyright © CRESTA Consortium Partners 2011

Copyright © CRESTA Consortium Partners 2011

1 Executive Summary

In this work, we introduce remote hybrid rendering strategies to make exascale resources available for interactive visualisation of large-scale numerical simulation data. We use the CRESTA co-design vehicles *OpenFOAM* [14] and *HemeLB* [13] to work out system requirements for a remote rendering solution in large-scale cluster environments. We compare different rendering strategies based on the *Sort-Last* algorithm according to their suitability for compositing in large-scale cluster environments on the road to exascale systems. *Sort-Last* was selected since rendering of large-scale distributed data generated by distributed post-processing systems is inherently supported by the algorithm with adequate communication overhead between the partitions. There are multiple possibilities for compositing of rendered images in the *Sort-Last* algorithm. The *Binary Swap* algorithm and its derivative, *2-3 Swap*, were discovered to be promising candidates for *Sort-Last* compositing of interactive renderings in large-scale clusters. We propose various additions to existing remote rendering architectures to minimise the network bandwidth needed in the system during the compositing phase.

- Image and video compression techniques implemented on CPU as well as on accelerators such as GPGPU devices can be employed to minimise communication overhead between cluster nodes.
- Mechanisms such as bounding box projection and view frustum culling can be used to identify areas in rendered images that do not have to be (re-)transmitted to other cluster nodes or additional GPU devices during compositing.
- Similarities in consecutive images, as well as between images for the left and right eye during stereo rendering, could be exploited to reduce the amount of data that have to be communicated to remote nodes.

We identify two possibilities for the integration of distributed post-processing and remote hybrid rendering in future exascale systems:

1. Employing accelerators such as GPGPU devices not only for rendering, but also for post-processing algorithms throughout the visualisation pipeline.
2. Using cores in advanced many-core CPUs such as Intel's MIC architecture for post-processing, rasterisation and image compositing.

Both methods would allow for the elimination - or at least a drastic reduction - of data transfers between the host CPU and the accelerator, which is a major cause of latency in today's large-scale post-processing systems.

Concluding this deliverable, a software architecture for remote parallel rendering in large-scale cluster environments is defined consisting of a parallel rendering component and a parallel compositor component. Different modes of communication between these two components are sketched that have to be supported for interactive, low-latency post-processing and rendering. These modes of communication lead to the definition of an interface between these two components, alongside with the definition of an interface between the post-processing environment (see WP 5.2), and the remote hybrid rendering solution that is going to be developed in WP 5.3.

2 Introduction

Remote hybrid rendering is used to make the post-processing resources used in large-scale cluster systems available to remote users. The structure of this document is as follows: Section 3.1 will introduce the workflows in visualisation and rendering of data obtained from scientific simulations as well as state requirements for rendering in exascale systems obtained from the co-design vehicles OpenFOAM and HemeLB. In section 3.2, existing parallel rendering techniques will be introduced and compared according to their usefulness to exploit the compute power of large-scale visualisation clusters.

Section 4 compares different rendering environments used by scientists or teams of engineers in the assessment of large-scale numerical simulation data. In section 5, bandwidth and latency requirements of compositing algorithms in large-scale clusters will be gathered and further analysed.

In section 6, software components for a remote parallel rendering system that fulfil these requirements will be defined. Finally, section 7 concludes the deliverable with a description of interfaces between the components specified in section 6.

2.1 Purpose

The purposes of this deliverable are as follows:

- Introduce existing methods for parallel remote rendering and establish a context in which they can be useful to support post-processing of numerical simulation results towards exascale systems.
- Define software components that are to be used in a large-scale hybrid parallel rendering system.
- Specify interfaces between these components as well as interfaces to post-processing software that will be developed in the scope of WP 5.2.

2.2 Glossary of Acronyms

CPU	Central Processing Unit
DMA	Direct Memory Access
FBO	Frame Buffer Object
GPU	Graphics Processing Unit
GPGPU	General purpose computing on graphics processing units
MIC	Many Integrated Cores
OpenGL	Open Graphics Language
RMA	Remote Memory Access
VBO	Vertex Buffer Object
WP	Work Package

3 Remote Rendering of Scientific Simulation Data

3.1 The Visualisation and Rendering Pipeline

The workflow of visualising simulation results to gain scientific insight can be divided into several tasks, traditionally defined as shown in Figure 1. The transfer of data between data space and visualisation space, the post-processing of simulation results, is usually a highly interactive process and will be analysed in detail in CRESTA's WP 5.2 [12]. In contrast to that, the rendering of geometric data, as the final step in the visualisation pipeline and focus of WP 5.3, is completely non-interactive and vastly accelerated by programmable graphics processing units (GPUs).

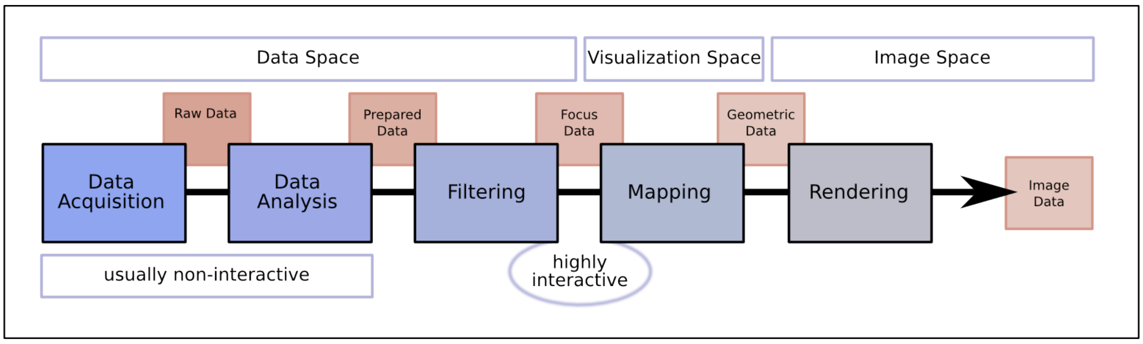


Figure 1: The visualisation pipeline (adapted from Haber and McNabb [1])

The rendering itself can be broken down into smaller parts, many of which can be influenced by small pieces of code, called *shaders*, which are supplied by the computer graphics developer and which are executed directly on the GPU as can be seen in Figure 2. Parallel rendering techniques which we will compare in section 3.2 considering their usefulness for exascale rendering can be classified according to the location in the rendering pipeline where the distribution of work using parallel processes is performed.

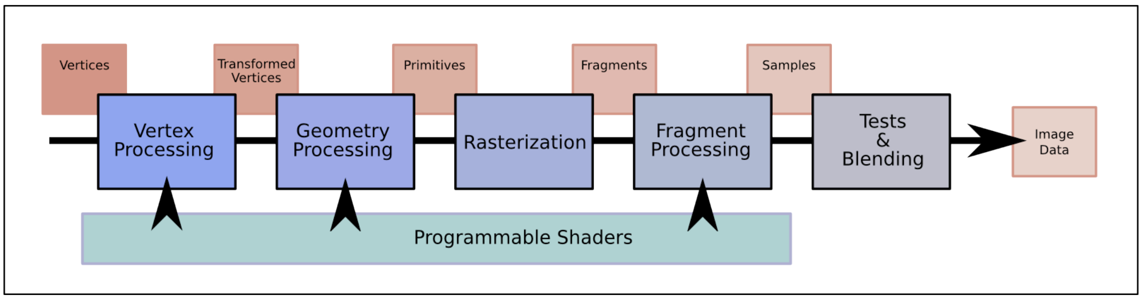


Figure 2: Simplified rendering pipeline

There are multiple possibilities for the implementation and execution of a rendering pipeline in modern computer architectures that are the basis of current high performance compute clusters.

1. Using the fixed-function pipeline implemented in the driver of the GPU.
2. Using programmable shaders to augment or replace parts of the fixed-function pipeline of the GPU.
3. Using custom software renderers that are either implemented on the CPU or on accelerators such as GPGPU devices.

There are numerical solvers that include methods for post-processing and rendering of these post-processed data for interactive analysis. *HemeLB*, a Lattice Boltzmann solver from UCL and one of the CRESTA co-design vehicles, uses software ray-casting for rendering of streamlines and isosurfaces that will be described in section 3.1.2.

3.1.1 Hardware-accelerated indirect Polygonal Rendering of Post-Processed Simulation Datasets from OpenFOAM Simulations

Data obtained from post-processing of numerical simulation results typically consist of a large number of geometric primitives (cf. Figure 1). These primitives, generally huge amounts of triangle data, can be rendered by GPUs using either the fixed function pipeline or programmable shaders. Because of memory limitations of single workstations or cluster nodes as well as the complexity of the computations that need to be performed, efficient parallel post-processing of large-scale simulation data is essential.

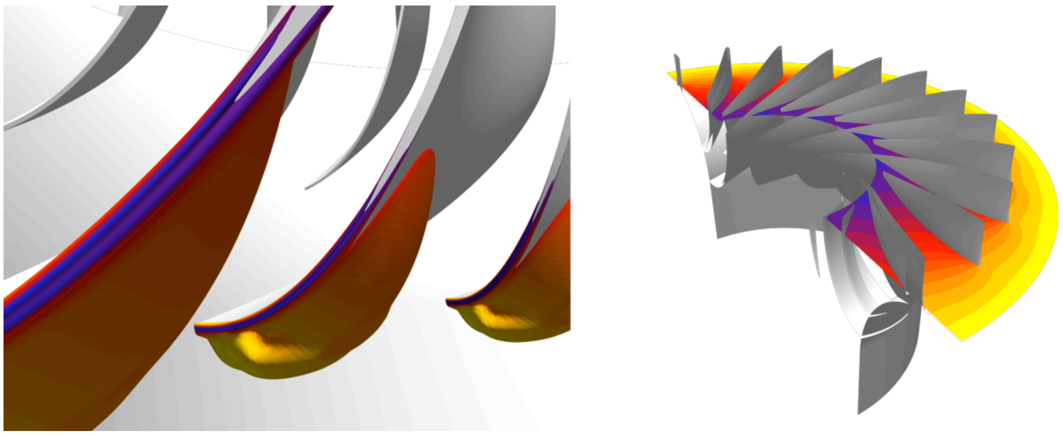


Figure 3: Post-processing of a turbomachinery simulation. Isosurface extraction for visualisation of cavitation (left), cutting plane showing colour-coded pressure distribution (right)

The size of engineering simulation datasets such as the ones shown in Figure 3, are predicted to grow to several 100 million volume elements with hundreds of time steps before the end of the CRESTA project. To reduce latency of user interactions within the visualisation environment, post-processing and rendering have to be tightly integrated and able to exploit locality of data. Parallel rendering of distributed graphics primitives offers the possibility to maintain interactive rendering speeds when visualising these large-scale, partitioned simulation data.

3.1.2 Raycasting in HemeLB

HemeLB's ray-tracing implementation is designed to be as local as possible and avoid unnecessary communication of data. Images are rendered locally by casting a ray from the viewpoint through each pixel of the image and accumulating data about the fluid sites that are near the ray's path through the geometry. Each core does this for its own fluid sites, grouping cubes of fluid site blocks into clusters and calculating in advance the area of each block mapped onto the screen in order to make the casting more efficient.

Each ray accumulates the distance from the screen to the first fluid site, the integrated length spent in fluid, the fluid density and stress nearest the screen. One implementation of the ray adds to this a colour-mapping of velocity and stress, which is integrated through the fluid. Another implementation adds "fog" to a velocity and stress integration, giving better depth-cuing.

Once each partition has rendered its local contribution to the overall image, the cast rays must be combined. To combine the images in this fashion, the cores are arranged in a binary tree, where lower-ordered ranks will have two child-cores with higher rank. In each subsequent iteration after rendering, each core passes its rays to its parent in the tree that combines the sets of rays from its children with its own set of rays. The properties accumulated by the rays are carefully chosen to allow this. Once the rays reach the topmost node in the tree, all cast rays have been combined and the final set can be transformed into an RGB image, ready for passing to an HTTP client or writing to disk.

HemeLB maintains scaling performance by ensuring that each ordered pair of cores will only perform one asynchronous send-receive per Lattice-Boltzmann iteration, reducing the amount of time each core spends on communication and minimising the effects of communication latency. The accumulation of ray information between cores therefore happens across several Lattice Boltzmann steps.

3.2 Parallel Rendering Techniques for Remote Rendering

Molnar et al. [15] classify parallel rendering techniques according to the location in the rendering pipeline where work is distributed to parallel rendering clients. Three classes of parallel rendering algorithms are identified.

3.2.1 Sort-First

Sort-First algorithms sort graphics primitives during geometry processing. Renderer partitions are responsible for a specific rectangle in screen-space. Graphics primitives can be assigned to renderer partitions by evaluating the screen-projection of the graphics primitives' bounding box. Sort-First parallel rendering is a suitable strategy when either the bandwidth requirements for re-distribution of geometric primitives is relatively small, or when rendering times are dominated by the limited fill-rate of the GPU.

3.2.2 Sort-Middle

Sort-Middle algorithms redistribute graphics primitives between geometry processing and rasterisation. After geometry processing, primitives transformed to screen coordinates are distributed to the partition responsible for rasterisation of the particular primitive. Sort-Middle algorithms do no longer play an important role in parallel rendering, since the data reduction between rasterisation and fragment processing is rather large and these processes are highly integrated on modern GPUs.

3.2.3 Sort-Last

Sort-Last rendering algorithms assign arbitrary geometry to each of the *n* partitions. Each rendering partition produces a full-size partial image, containing colour and depth rendering of the objects assigned to the specific partition. These *n* images are then sent to one or more compositing nodes where they have to be overlaid considering per-fragment z-visibility.

3.3 Sort-Last Compositing Methods

The most simple sort-last compositing solution is the serial approach which combines and merges all intermediate images on the destination rendering unit responsible for the final display (cf. the sample scenario in Figure 4 and Figure 5). Several other parallelisation schemes for software composition have been proposed. Most notably, these include *Direct Send* [2], *Binary Tree* [3], *Binary Swap* [4] and *Parallel Pipeline* [5], among which *Binary Swap* is the most commonly used algorithm [6].

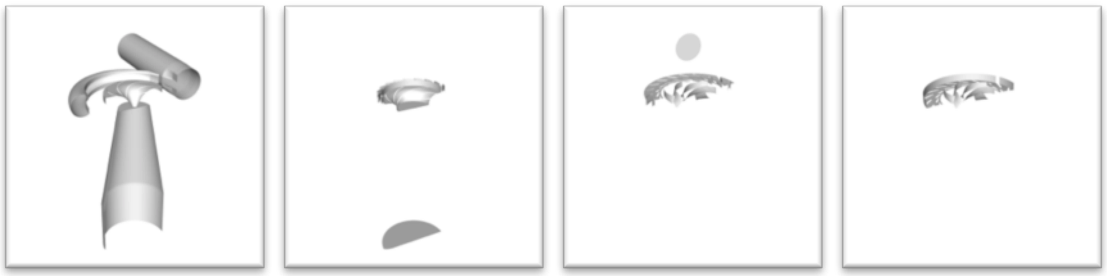


Figure 4: Rendered images of individual partitions on different post-processing cluster nodes

Figure 5: Result of final image composition

3.3.1 Serial Compositing

When using *Serial Compositing*, all nodes send their colour and depth images to a single compositing node. This leads to very high bandwidth requirements to the single compositor node and makes concurrent image transmission impossible. Since the total compositing time needed for this method increases linearly with the number of nodes in the system, interactive frame rates cannot be achieved when reaching a certain number of nodes.

3.3.2 Direct Send

With *Direct Send* compositing, the final image-gathering task is divided into *n* screen-space tiles to avoid exchanging full-size images between the n compositing nodes [6]. Each tile is associated to and composited by one cluster node, and the composited tiles are eventually assembled together to form the final image.

With direct send, each of the *n* nodes has to read back *n-1* tiles from GPU memory. Each node sends the tiles to their respective compositing node, where composition of sub-images to tiles is performed. Each tile is then sent to the final display node. Although direct send allows for concurrent transmission of image tiles to their corresponding compositor node, the algorithm requires all-to-all communication which will be suboptimal in large-scale cluster networks which are not fully-connected.

3.3.3 Binary Swap

Binary Swap algorithms [[4] also distribute the composition of parts of images to all nodes in the cluster. A naïve approach for parallel merging of the partial images is to do binary compositing. By pairing up processors in order of compositing, each disjoint pair produces a new subimage. Thus after the first stage, we are left with the task of compositing only *n/2* subimages. Then we use half the number of the original processors, and pair them up for the next level of compositing. Continuing similarly, after $log_2 n$ stages, the final image is obtained. One problem with the above method is that during the compositing process many processors become idle. At the top of the tree, only one processor is active [4].

To exploit the compute power of all nodes in the cluster, the key idea in *Binary Swap* is that, at each compositing stage, the two processors involved in a composite operation split the image plane into two pieces and each processor takes responsibility for one of the two pieces.

In the early phases of the *Binary Swap* algorithm, each processor is responsible for a large portion of the image area as can be seen in Figure 6. In later phases of the algorithm, the processors are responsible for a smaller and smaller portion of the image area. At the top of the tree, all processors have complete information for a small rectangle of the image. As in *Direct Send*, the final image can be constructed by sending the subimage tiles to the display node or a remote workstation. The bandwidth requirements for *Binary Swap* compositing are similar to the *Direct Send* algorithm. Fortunately, the communication patterns are much more suitable for cluster networks that are not fully connected, since larger image tiles are distributed in early stages of the compositing tree where directly neighbouring cluster nodes communicate.

There are further developments of binary swap such as *2-3 Swap* [8] that eliminate the need to use power-of-two number of nodes. *2-3 Swap* along with enhancements such as improved scanline methods, RLE encoding and the use of bounding boxes [9], make for a promising approach towards even very large numbers of rendering nodes in future high performance cluster systems.

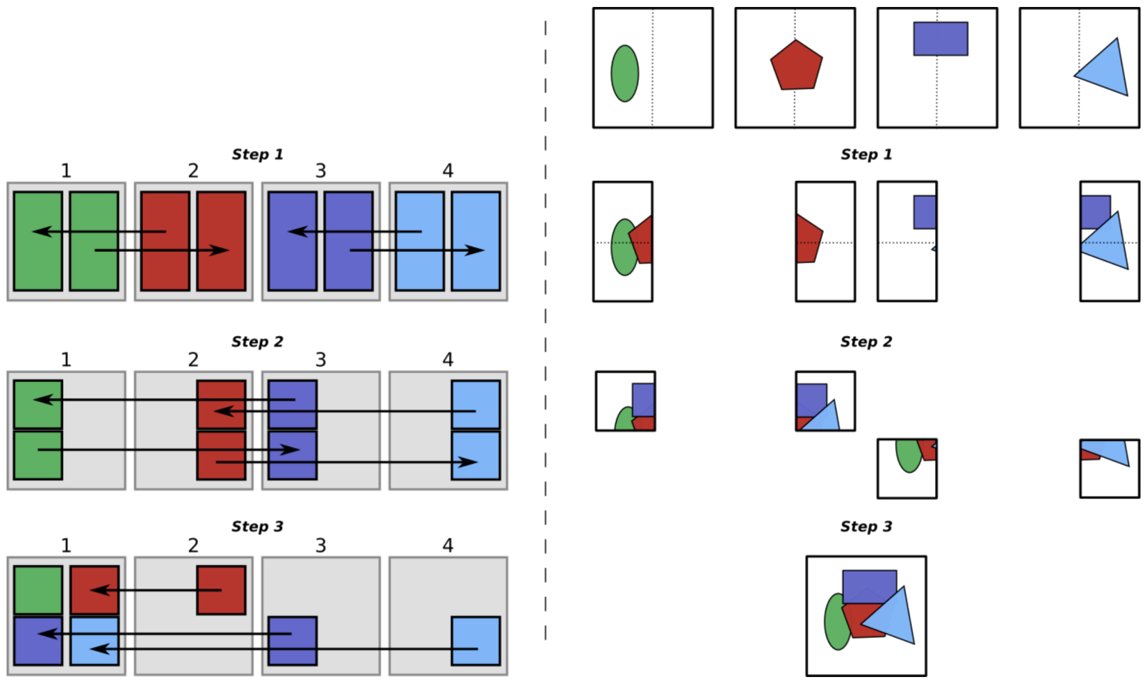


Figure 6: Communication between partitions (left), Composition of sub-images on partitions (right)

3.4 State of the Art in Remote Parallel Rendering and Compositing Frameworks

There are different types of software libraries that can be used to implement remote parallel rendering in cluster systems. We provide a short list of methods citing one typical implementation of each and discuss their advantages and disadvantages for remote parallel rendering regarding exascale resources.

Chromium [17] intercepts OpenGL calls and forwards them to one or more Chromium servers. This can be used to split an OpenGL command stream into smaller parts that can then be processed by different nodes. After rasterisation, depth-buffer composition of the resulting images can be applied, leading to an implementation of Sort-Last, using object decomposition at the OpenGL command level.

OpenSG [16] is a scenegraph API that provides functionality for parallel rendering, including network data distribution and scalable rendering modes.

Equalizer [15] is a parallel rendering middleware, which enables the development of fully distributed and parallel graphics applications.

Ice-T [18] (Image Composition Engine for Tiles) is a lightweight parallel compositing library that can be used for tiled displays as well as single displays, featuring different compositing algorithms, including sort-last based image compositing.

Libraries such as Chromium have been developed to enable existing serial programs to exploit parallel rendering resources. In applications that are to be optimised for large-scale parallel systems, developers typically are able to make much better optimisations themselves than what would be possible by an automatic parallelisation of library calls.

Distributed scenegraph APIs such as OpenSG enable the programmer to distribute rendering data among different cluster nodes. Typically, they require the scenegraph itself to be replicated among the cluster nodes, often leading to communication requirements among nodes when the scenegraph is changed in one partition of the parallel program, which is a prohibitive network overhead in large-scale programs.

Middleware libraries or frameworks are typically trying to be independent of a given scenegraph API. In large-scale real-world projects, the combination of Equalizer with scenegraph libraries such as OpenSG or OpenSceneGraph, which might be already used in the post-processing or rendering system, can become very cumbersome to the programmer. Since the functionality of rendering middleware and scenegraph libraries often overlap and the layout of data structures in use are incompatible, integration of the different libraries often is not possible in an efficient manner.

Integration of light-weight compositing libraries such as Ice-T into existing systems proves to be much easier since these libraries are often written to complement existing rendering libraries or scenegraphs. Often, a large number of different image encodings and layouts are already supported. Although the compositing libraries in existence today are not yet optimised for many-core CPUs or GPGPU devices, we believe that the methods used in these parallel compositing libraries could be an interesting starting point to enable visualisation software to make use of future exascale systems regarding distributed rendering of large-scale simulation data.

4 Rendering Environments

There are different rendering environments that make it possible for individuals or small teams of engineers to collaborate on exploring large-scale numerical simulation data. These environments have different requirements for the remote rendering system. In general, it should be possible to combine various rendering environments, even of different types, to allow sharing of a post-processing session by multiple, geographically separated users.

4.1 User Workstations

Workstations are typically used by single users and require renderings of comparably low resolution. Since the viewpoint as well as the post-processing data generally is not updated all the time, the load on the rendering is not constantly high. This enables the post-processing, rendering and compositing components to execute time-consuming load-balancing in between rendering steps. Workstation users also might be satisfied with compressed images and progressive updates, providing low-latency by lowering image quality. Collaborative analysis of simulation datasets can be assisted by making it possible for multiple desktop workstations to participate in the post-processing.

4.2 Immersive Virtual Reality

An immersive digital environment is an artificial, interactive, computer-created scene within which users can immerse themselves. Because of cost-efficiency, these installations are often a central resource in the company, university or research institution, in many cases featuring a high bandwidth and low latency network connection to the datacenter.

4.2.1 Multi-Wall Projections and Stereo Rendering

Virtual Reality environments such as large tiled display walls or CAVEs (cf. Figure 7) need to make high-resolution, very low latency, high bandwidth stereo image renderings available to the users. Since these installations also often feature tracking equipment for user interactions as well as viewpoint changing, the rendering needs to be updated much more often compared to desktop environments, often continuously. The targeted immersion of the users into the scene makes it prohibitive to introduce compression artifacts or progressive updates to the rendered images.

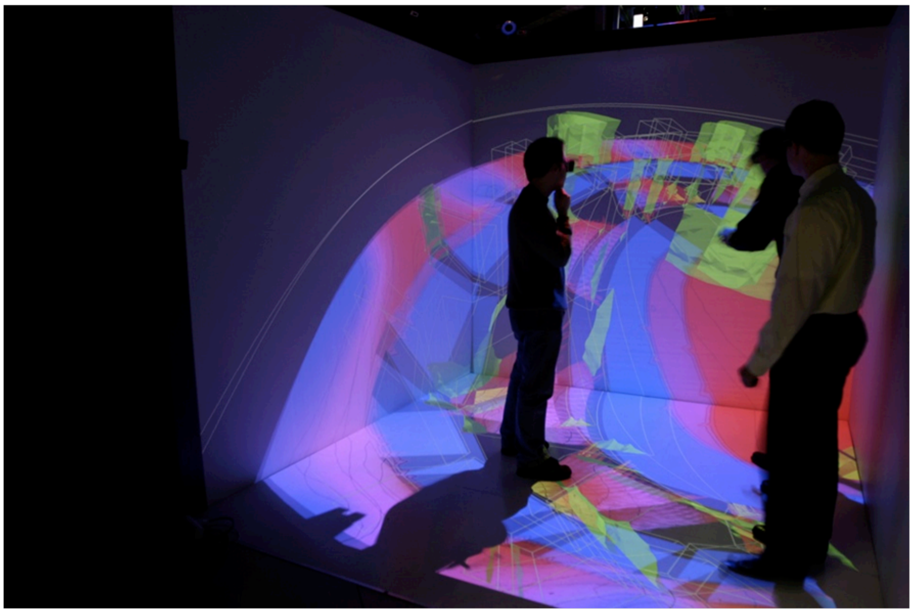


Figure 7: Post-processing of numerical simulation data in a CAVE

5 Analysis of Requirements for Remote Rendering on Exascale Systems

5.1 Exascale Systems

5.1.1 Massively Parallel Systems

Large-scale clusters in use today provide the possibility to exploit parallelism on different levels.

- Data parallelism can be exploited on the cluster nodes' CPUs as well as on accelerator cards such as GPGPU devices that are many-core processing units themselves. On GPGPU devices, data parallelism through the use of stream-processing *must* be exploited to achieve adequate speedups when employing hundreds of relatively simple GPU cores. Data parallelism in clusters has to be exploited at another scale by simultaneously processing partitions of data on different nodes in a distributed memory environment.
- Task parallelism makes it possible to execute different algorithms on the same or distinct data in parallel. In the case of visualisation, post-processing, rendering and compositing of a dataset or a partition thereof have to be executed sequentially and cannot be overlapped. Though, visualisation of data partitions might be, depending on the post-processing algorithms that are employed, computed in parallel to the visualisation of other partitions.

5.1.2 Accelerators

HPC clusters often feature accelerators such as GPGPU devices that can be used for post-processing and rendering. Since many post-processing algorithms such as surface extraction are relatively easy to implement in a data-parallel form, these devices provide an optimal target for data analysis. GPGPU post-processing is even more attractive in low-latency applications, since expensive data transfers from host to GPU memory are no longer necessary between post-processing and rendering. The availability of accelerators makes it possible to free host CPUs for load balancing tasks that require re-partitioning of data, or to develop hybrid algorithms that share computations between CPU and accelerators.

5.1.3 Many-Core CPUs

The introduction of many-core CPUs such as Intel's *MIC* architecture (*M*any *I*ntegrated *C*ores) indicates the possibility that the distance between traditional CPUs and GPGPU cores might be shrinking. In future exascale systems, applications may exploit the potential to efficiently generate rasterised, ray-casted or ray-traced images directly on some cores of the utilised many-core CPUs without the need for additional accelerators. This would also allow for a reduction of the latency of interactive post-processing algorithms as has been argued in section 5.1.2. The integration of the different memory spaces that exist now in accelerator-based systems would also lead to improvements in programmability as well as to optimisation potential in image compositing algorithms, since memory on remote nodes would hopefully be accessible by RMA.

5.2 Performance Considerations for Massively Parallel Rendering

5.2.1 Bandwidth Requirements

The total number of pixels to be transmitted, discarding potentially expensive optimisations to both groups of algorithms for the moment, is the same for *Direct Send* and *Binary Swap* and *2-3 Swap.* The most important advantage of *Binary Swap* and *2-3 Swap* in large-scale clusters is the exploitation of fast nearest neighbour communication paths. When using *Direct Send*, link contention is likely to happen since multiple nodes are sending messages to the same node at the same time.

Yu et al [15] demonstrate empirically that *Binary Swap* and *2-3 Swap* show a much more pleasant behaviour than *Direct Send* using both moderate image sizes (1024x1024) as well as a relatively small number of parallel rendering nodes (1024). In their implementation, the total compositing time of *Direct Send* grows much faster compared to *Binary Swap* and *2-3 Swap* when the number of nodes or the image size is increased.

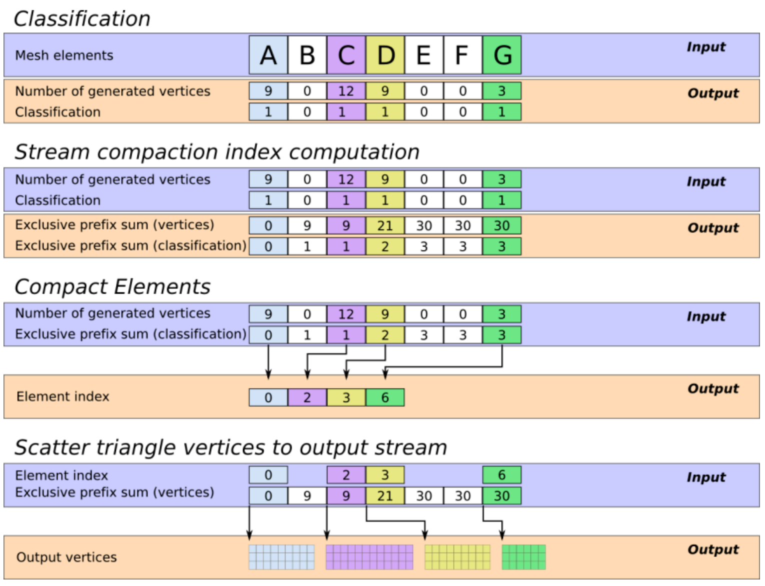
5.2.2 Latency Considerations

Direct send may require a total number of n * (n – 1) messages to be sent during compositing. In *Binary Swap* compositing, each node sends exactly $log_2 n$ messages, making the total number of messages $n * log_2 n$. When implemented in systems with a high message passing overhead, this behaviour will certainly decrease the applicability of *Binary Swap* or generally of tree-based composition algorithms. In an asynchronous message passing environment, *Direct Send* latency costs are *O(1)*. Since *Binary Swap* requires a total number of $log_2 n$ compositing passes, latency grows logarithmically with the number of nodes in the cluster participating in parallel rendering.

6 Definition of a Remote Rendering System Towards Exascale

6.1 Software Component: Massively Parallel Renderer

The parallel renderer has to be tightly integrated with the post-processing environment. To optimise the system for locality of data, immediate rendering has to be performed on the same cluster nodes that are used for parallel post-processing. Data transfer between post-processing and rendering can be further sped up when both can make use of the same accelerator, for example a dedicated programmable GPU per node in the post-processing cluster. The output generated by the post-processing algorithms should be optimised for immediate rendering on the GPU without having to perform additional conversions or copying of data as can be seen in the surface extraction algorithm example in Figure 8.



Classification
Mesh elements
A B C D E F G
Input
Number of generated vertices 9 0 12 9 0 0 3
Classification 1 0 1 1 0 0 1
Output
Stream compaction index computation
Number of generated vertices 9 0 12 9 0 0 3
Classification 1 0 1 1 0 0 1
Input
Exclusive prefix sum (vertices) 0 9 9 21 30 30 30
Exclusive prefix sum (classification) 0 1 1 2 3 3 3
Output
Compact Elements
Number of generated vertices 9 0 12 9 0 0 3
Exclusive prefix sum (classification) 0 1 1 2 3 3 3
Input
Element index 0 2 3 6
Output
Scatter triangle vertices to output stream
Element index 0 2 3 6
Exclusive prefix sum (vertices) 0 9 9 21 30 30 30
Input
Output vertices
Output

Figure 8: Surface extraction using GPGPU computing with focus on rendering performance

For post-processing algorithms that do not finish their computation in time for interactive feedback, multiple different scenarios have to be supported by the renderer:

- The post-processing algorithm may generate partial results for immediate rendering before the whole computation is finished. These partial results may be updated by the post-processing algorithm in further calculations.
- The post-processing algorithm may generate incomplete results that are not suitable for immediate rendering, such as geometry that was already processed but is still missing data for appropriate colour mappings.
- The post-processing algorithm may generate an inaccurate result for intermediate rendering that will be corrected in further calculations. These inaccurate results may be generated e.g. by using a coarser sampled grid instead of the high resolution CFD grid to get an approximate initial result.

For accurate blending of translucent graphics primitives in the subsequent parallel compositing step, the renderer has to support proper sorting and additional multi-pass rendering of these primitives.

6.2 Software Component: Massively Parallel Compositor

After independent parallel rendering of partitioned simulation data obtained from post-processing, a parallel compositor has to blend the various sub-images to form a final image (or final images) for display at the remote site. The parallel compositor should allow for exchangeable algorithms such as *Direct Send*, *Binary Swap* and *2-3 Swap*. To support low-bandwidth networks to remote sites, an optional compression of image streams will have to be supported. This encoding could also be used to speed up communication of sub-images between the cluster nodes themselves. CPU- as well as GPU-based compression schemes should be evaluated regarding their suitability for large-scale rendering. For scenarios where consecutive frames bear high affinity, such as constant small amounts of head movement in virtual environments, similarities between frames should be exploited to optimise for network bandwidth. Similarities between frames in stereo-rendering environments may also prove to be a promising target for further optimisation.

In addition to high throughput, low-latency application networks in compute clusters, the compositor has to support outgoing communication to remote hosts to be able to send images to workstations or virtual reality environments outside of the local cluster network.

7 Software Architecture Interfaces

7.1 Post-Processing to Rendering

Since post-processing of numerical simulation data is a highly interactive process, new post-processing data is generated continuously. These data are converted to geometric primitives, such as texture-mapped triangles, through the visualisation pipeline. These large amounts of data have to be passed to the renderer with a minimum amount of added latency, as described in section 6.1. When the post-processing and rendering are executed on the same GPU, the interface between the two components can be optimised to the passing of pointers to vertex buffer objects (VBOs). Due to implementation constraints in today's graphics hardware and drivers, this method only works when post-processing and rendering are implemented to share a graphics context, which in turn is only possible if the two are actually implemented as a single process. Since modern GPUs and GPU drivers are currently underway to develop virtual memory and shared address spaces between multiple processes or graphics contexts, this issue should be solved in the medium term.

The interface between post-processing and rendering should provide a feedback mechanism to make the load on the rendering and compositing processes available to the post-processing. The post-processing environment will then be able to decide how and when to move data between different rendering partitions to perform load-balancing. The interface to the rendering has to communicate the type of data that should be rendered, as well as the mode of update that is performed as described in section 6.1: full results, partial results, incomplete data or inaccurate data.

7.2 Parallel Rendering to Parallel Compositing

The parallel rendering component has to provide an interface for communication between the graphics device memory and the host memory. This can either be implemented as a traditional readback/upload, but should be able to be adapted to methods that make DMA possible to remote graphics cards such as NVIDIA's GPUs directly over HPC network interfaces such as Infiniband. As the compositing should be able to be accelerated by GPGPU devices, the compositor has to be granted access to the rendered image, for compression of the images to reduce bus and network communications, as well as for the actual compositing of (sub-) images. In *Binary Swap* algorithms, where the composition is divided into multiple stages, explicit attention has to be spent on the performance overhead of the interface between rendering and compositing.

To accelerate compositing, the rendering should be able to provide bounding rectangles of the area that has actually been rendered in this partition. This can be implemented e.g. as a projection of the object's bounding boxes to the viewing plane. A bounding rectangle enables the compositing to optimise memory transfers from GPU device to host, and from cluster node to cluster node by transferring only those parts of the image that have actually been affected by rendering.

8 References

[1] Haber, R. and McNabb, D.: Visualization idioms: A conceptual model for scientific visualization systems. In Visualization in Scientific Computing (1990), G. Nielson, B. Shriver, and L. Rosenblum, Eds., IEEE Computer Society Press, pp 74-92.

[2] Stompel, A. and Ma, K.-L. and Lum, E. B. and Ahrens, J. and Patchett,: J. SLIC: Scheduled linear image composition on a PC cluster system. Parallel Computing 30, 2 (2004), pp 285-299.

[3] Shaw, C. D. and Green, M. and Schaeffer, J.: A VLSI architecture for image composition. In Advances in Computer Graphics Hardware III. Springer Verlag, pp 183-200.

[4] Ma, K.-L. and Painter, J. S. and Hansen, C. D. and Krogh, M.F.: Parallel Volume Rendering using Binary-Swap Image Composition. IEEE Computer Graphics and Algorithms, 1994.

[5] Lee, T. Y. and Raghavendra, C. S. and Nicholas, J. N.: Image composition methods for sort-last polygon rendering on 2D mesh architectures. In Proceedings of the IEEE Symposium on Parallel Rendering 1995, pp. 55-62.

[6] Eilemann, Stefan and Pajarola, Renato.: Direct Send compositing for parallel sort-last rendering. ACM SIGGRAPH ASIA 2008, ACM, New York, pp. 39:1-39:8

[7] Molnar, S. and Ellsworth, D. and Fuchs, H.: A sorting classification of parallel rendering. In IEEE Computer Graphics and Applications (1994), vol. 14. pp. 23-32.

[8] Yu, H. and Wang, C. and Ma, K.-L.: Massively parallel volume rendering using 2-3 swap image compositing. In Proceedings of the 2008 ACM/IEEE conference on supercomputing (Piscataway, NJ, USA, 2008), SC08, IEEE Press, pp 48:1-48:11.

[9] Takeuchi, A. and Ino, F. and Hagihara, K.: An improved binary swap compositing for sort-last parallel rendering on distributed memory multi-processors. In Parallel Computing 29 (November 2003), pp. 1745-1762.

[10] Neumann, U. Parallel Volume-Rendering Algorithm Performance on Mesh-Connected Multicomputers. In Proceedings of Parallel Rendering Symposium 1993, pp. 97-104.

[11] Matura, G. and Basermann, A. CRESTA D 5.1.1: Pre-processing: analysis and system definition for exascale systems

[12] Wagner, C. and Chen, F. and Basermann, A. CRESTA D 5.2.1: Post-processing: analysis and system definition for exascale systems

[13] Mazzeo, M.D. and Coveney, P.V. HemeLB: A high performance parallel lattice-Boltzmann code for large scale fluid flow in complex geometries, Computer Physics Communications, Volume 178, Issue 12, 15 June 2008, Pages 894-914

[14] Weller, H.G. and, Tabor, G. and Jasak, H. and Fureby, C. A Tensorial Approach to Computational Continuum Mechanics using Object Orientated Techniques, *Computers in Physics*, Vol. 12, No. 6. (1998), pp. 620-631

[15] Eilemann, S. and Makhinya, M. and Pajarola, R. Equalizer: a scalable parallel rendering framework. In Proceedings of SIGGRAPH Asia 2008

[16] Roth, M. and Reiners, D. "Sorted Pipeline Image Composition ", in Proceedings of EGPGV, 2006, pp.119-126.

[17] Humphreys, G. and Houston, M. and Ng, R. and Frank, R. and Ahern, S. and Kirchner, P.D. and Klokowski, J.T. Chromium: a stream-processing framework for interactive rendering on clusters. SIGGRAPH Asia 2008 Courses

[18] Moreland, K. and Kendall, W. and Peterka, T. and Huang, J. An Image Compositing Solution at Scale. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*. November 2011.