Modern Distributed Rendering Systems

Dissertation
Submitted TBD

to the Faculty of Economics, Business Administration
and Information Technology of the University of Zürich

for the degree of
Doctor of Science (Ph.D)

by Stefan Eilemann

Supervisor: Prof. Dr. Renato Pajarola
Co-referee: TBD

Abstract

We are living in the big data age: An ever increasing amount of data is being produced by users through data acquisition and simulations. While large scale analysis and simulatons have received significant attention for cloud computing and HPC systems, software to efficiently visualize large amounts of data is struggling to keep up.

We propose to research system software to facilitate and accelerate large data visualization through parallel rendering, and to validate the research and development of this system software by the development of new applications for large data visualization.

This research and development will enable domain scientists and large data engineers to better extract meaning from their data, making it feasible to explore more data by accelerating the rendering and allowing the use of high-resolution displays to see more detail.

Due to the nature of this research, we propose an engineering-driven, iterative research process. Based on the foundations of a generic parallel rendering system, individual research questions can be addressed in isolation and optimized through data-driven benchmarking, and integrated in product quality into the parallel rendering system.

Acknowledgements

The research leading to this proposal was supported in part by the Blue Brain Project, the Swiss National Science Foundation under Grant 200020-129525, the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 604102 (Human Brain Project), the Hasler Stiftung grant (project number 12097), and the King Abdullah University of Science and Technology (KAUST) through the KAUST-EPFL alliance for Neuro-Inspired High Performance Computing.

I would like to take the opportunity to thank the Blue Brain Project, in particular the visualization team, and all the other contributors for their support in the research leading to this proposal. Furthermore, I would like to thank Prof. Renato Pajarola for his long-term commitment to my research work.

Zürich, March 8, 2018 Stefan Eilemann

Contents

List of Figures

1 Background

1.1 Motivation

After decades of exponential growth in computational performance, storage and data acquisition, computing is now well in the big data age, where future advances are measured in our capability to extract meaningful information from the available data. Visual analysis based on interactive rendering of three-dimensional data has been proven to be a particularly efficient approach to gain intuitive insight into the spatial structure and relations of very large 3D data sets. These developments create new, unique challenges for applications and system software to enable users to fully exploit the available resources to gain insight from their data.

The quantity of computed, measured or collected data is exponentially growing, fueled by the pervasive diffusion of digitalization in modern life. Moreover, the fields of science, engineering and technology are increasingly defined by a data driven approach to conduct research and development. High-quality and large-scale data is continuously generated at a growing rate from sensor and scanning systems, as well as from data collections and numerical simulations in a number of science and technology domains.

Display technology has made significant progress in the last decade. High-resolution screens and tiled display walls are now affordabe for most organizations and are getting deployed at an increasing rate. This increased resolution and display size helps with understanding the data, but with the quadratic increase in pixels to be rendered, it increases the pressure on rendering algorithms to deliver interactive framerates. Furthermore, for larger system it becomes necessary to develop parallel and distributed applications.

However, not only applications are becoming more and more data-driven, but also the technology used to tackle these kinds of problems is rapidly witnessing a paradigm shift towards massively parallel on-chip and distributed parallel cluster solutions. On one hand, parallelism within a system has increased massively, with tenths of CPU cores, thousands of GPU cores and multiple CPUs and GPUs in a single system. On the other hand, massively parallel distributed systems are easily accessible from various cloud infrastructure providers, and are also affordable for on-site hosting for many organizations.

System software to exploit the available hardware parallelism capable of performing efficient interactive data exploration has not kept up with the pace in hardware developments and data gathering capabilities. On one hand, this is due to an inherent delay between hardware and software capabilities, since development typically only starts once the hardware is available. On the other hand, existing software engineered for different design parameters has a significant inertia to change, to the extreme of the necessity to rewrite it from scratch.

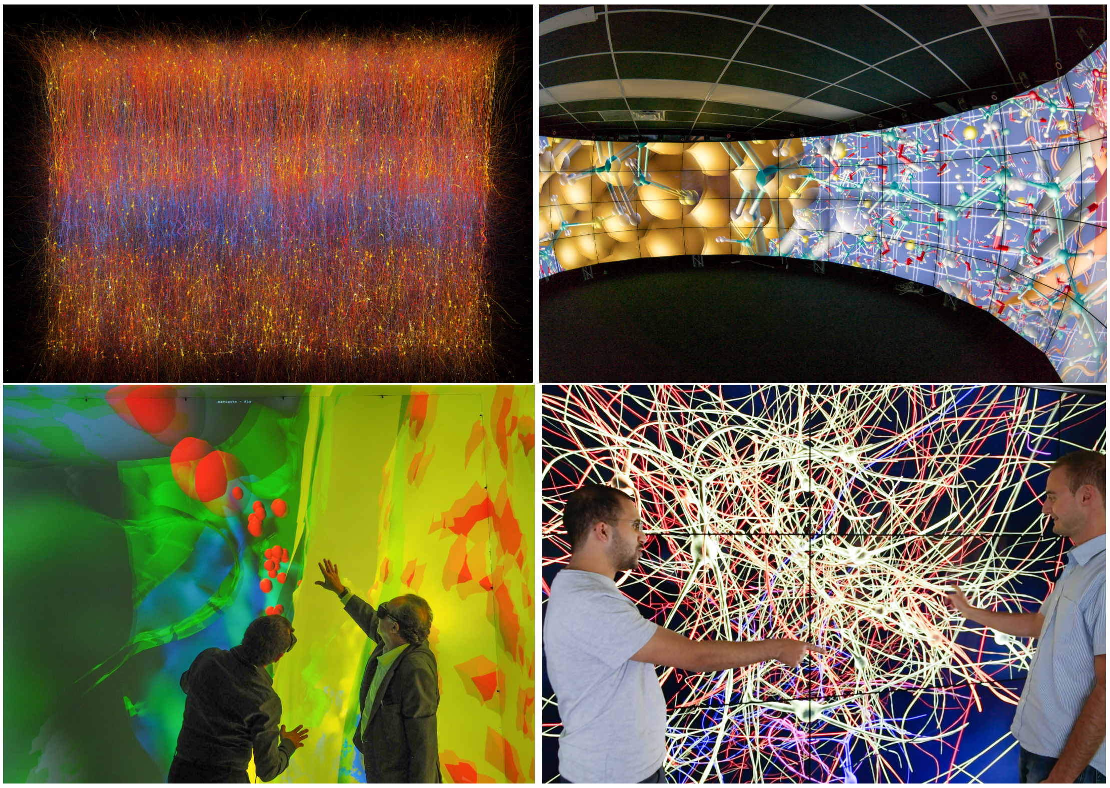

Figure 1 – Large Data Visualization: Large data visualization of a brain simulation, molecular visualization in the Cave[2], exploration of EM stack reconstructions in a Cave, collaborative data analysis on a tiled display wall.

In the context of emerging data-intensive knowledge discovery and data analysis, efficient interactive data exploration methodologies have become increasingly important. Visual analysis by means of interactive visualization and inspection of three-dimensional data is a particularly efficient approach to gain intuitive insight into the spatial structure and relations of very large 3D data sets. However, defining visual and interactive methods scalable with problem size and degree of parallelism, as well as generic applicability of high-performance interactive visualization methods and systems are recognized among the major current and future challenges.

1.2 Interactive Visualization

1.3 Parallel Rendering

The main performance indicator for Large Data Interactive Rendering is the performance of the rendering algorithm, that is, the framerate with which the program produces new images. This framerate can be improved by either using faster or more hardware, or by better algorithms exploiting the existing hardware and data. This proposal primarily focuses on the first approach using parallel rendering to exploit the CPU and GPU parallelism available on a single system or a distributed cluster. The early fundamental concepts have been laid down in [MCEF94] and [Cro97]. A number of domain specific parallel rendering algorithms and special-purpose hardware solutions have been proposed in the past, however, only few generic parallel rendering frameworks have been developed (Figure 2). We will focus on sort-last and sort-first rendering, since sort-middle architectures are only feasible in a hardware implementation due to the large amount of fragments processed and transferred in the sorting stage.

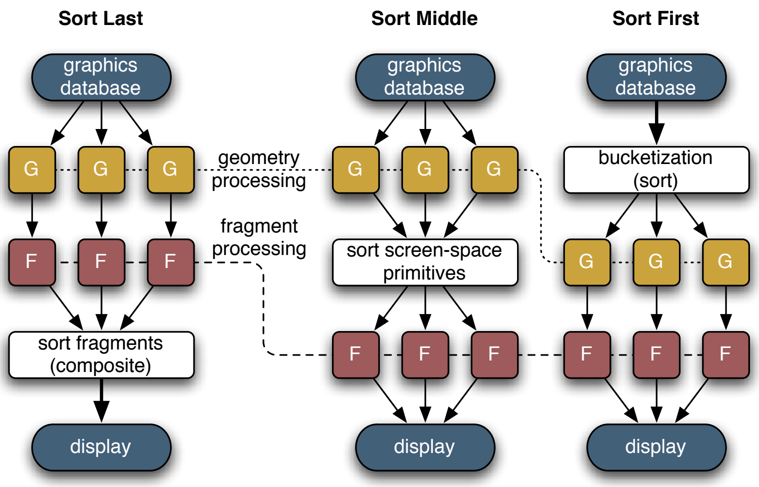

Figure 2 – Sort-last, sort-middle and sort-first parallel rendering

1.3.1 Domain specific solutions

Cluster-based parallel rendering has been commercialized for off-line rendering (i.e. distributed ray-tracing) for computer generated animated movies or special effects, since the ray-tracing technique is inherently amenable to parallelization for off-line processing. Other special-purpose solutions exist for parallel rendering in specific application domains such as volume rendering [LWMT97, Wit98, HSC$^+$00, SL02, GS02, NSJ$^+$05] or geo-visualization [VR91, AG95, LDC96, JLMVK06]. However, such specific solutions are typically not applicable as a generic parallel rendering paradigm and do not translate to arbitrary scientific visualization and distributed graphics problems.

In [NC07], parallel rendering of hierarchical level-of-detail (LOD) data has been addressed and a solution specific to sort-first tile-based parallel rendering has been presented. While the presented approach is not a generic parallel rendering system, basic concepts presented in [NC07] such as load management and adaptive LOD data traversal can be carried over to other sort-first parallel rendering solutions.

1.3.2 Special-purpose architectures

Historically, high-performance real-time rendering systems have relied on an integrated proprietary system architecture, such as the early SGI graphics super computers. These special-purpose solutions have become a niche product as their graphics performance does not keep up with off-the-shelf workstation graphics hardware and scalability of clusters.

Due to its conceptual simplicity, a number of special-purpose image compositing hardware solutions for sort-last parallel rendering have been developed. The proposed hardware architectures include Sepia [MHS99, Lev04], Sepia 2 [LMS+01a, LMS+01b], Lightning 2 [SEP+01], Metabuffer [BBDZ00, ZBB01], MPC Compositor [MOM+01] and PixelFlow [MEP92, EMP+97], of which only a few have reached the commercial product stage (i.e. Sepia 2 and MPC Compositor). However, the inherent inflexibility and setup overhead have limited their distribution and application support. Moreover, with the recent advances in the speed of CPU-GPU interfaces, such as PCI Express, NVLink and other modern interconnects, combinations of software and GPU-based solutions offer more flexibility at comparable performance.

1.3.3 Generic approaches

A number of algorithms and systems for parallel rendering have been developed in the past. On one hand, some general concepts applicable to cluster parallel rendering have been presented in [Mue95, Mue97] (sort-first architecture), [SZF+99, SFLS00] (load balancing), [SFL01] (data replication), or [CMF05, CM06] (scalability). On the other hand, specific algorithms have been developed for cluster based rendering and compositing such as [AP98], [CKS02] and [YYC01, SML+03]. However, these approaches do not constitute APIs and libraries that can readily be integrated into existing visualization applications, although the issue of the design of a parallel graphics interface has been addressed in [ISH98].

Only few generic APIs and (cluster-)parallel rendering systems exist which include VR Juggler [BJH+01] (and its derivatives), Chromium [HHN+02] (an evolution of [HH99, HBEH00, HEB+01]), ClusterGL [NHM11] and OpenGL Multipipe SDK [JDB+04, BRE05, MPK]. These approaches can be categorized into transparent interception and distribution of the OpenGL command stream and into the parallization of the application rendering code (Figure 3).

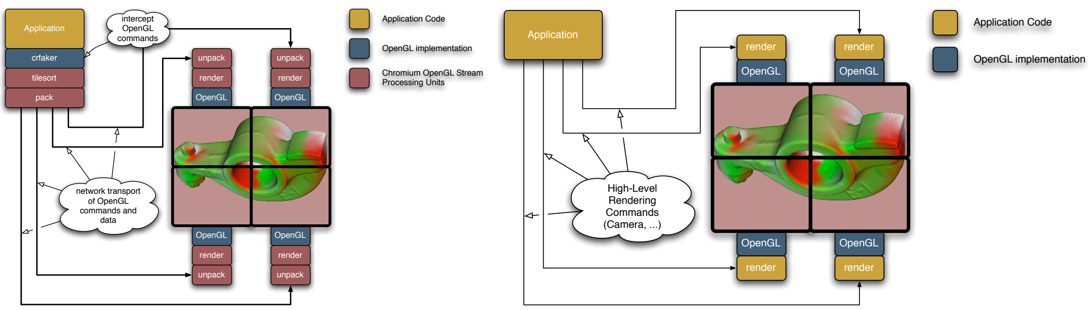

Figure 3 – Transparent OpenGL interception and parallelization of the rendering code

VR Juggler [BJH+01, JBBCN98] is a graphics framework for virtual reality applications which shields the application developer from the underlying hardware architecture, devices and operating system. Its main aim is to make virtual reality configurations easy to set up and use without the need to know details about the devices and hardware configuration, but not specifically to provide scalable parallel rendering. Extensions of VR Juggler, such as for example ClusterJuggler [BCN03] and NetJuggler [AGL+02], are typically based on the replication of application and data on each cluster node and basically take care of synchronization issues, but fail to provide a flexible and powerful configuration mechanism that efficiently supports scalable rendering as also noted in [SWNH03]. VR Juggler does not support scalable parallel rendering such as sort-first and sort-last task decomposition and image compositing nor does it provide support for network swap barriers (synchronization), distributed objects, image compression and transmission, or multiple rendering threads per process, important for multi-GPU systems.

While Chromium [HHN+02] provides a powerful and transparent abstraction of the OpenGL API, that allows a flexible configuration of display resources, its main limitation with respect to scalable rendering is that it is focused on streaming OpenGL commands through a network of nodes, often initiated from a single source. This has also been observed in [SWNH03]. The problem comes in when the OpenGL stream is large in size, due to not only containing OpenGL calls but also the rendered data such as geometry and image data. Only if the geometry and textures are mostly static and can be kept in GPU memory on the graphics card, no significant bottleneck can be expected as then the OpenGL stream is composed of a relatively small number of rendering instructions. However, as it is typical in real-world visualization applications, display and object settings are interactively manipulated, data and parameters may change dynamically, and large data sets do not fit statically in GPU memory but are often dynamically loaded from out-of-core and/or multiresolution data structures. This can lead to frequent updates not only of commands and parameters wich have to be distributed but also of the rendered data itself (geometry and texture), thus causing the OpenGL stream to expand dramatically. Furthermore, this stream of function calls and data must be packaged and broadcast in real-time over the network to multiple nodes for each rendered frame. This makes CPU performance and network bandwidth a more likely limiting factor.

The performance experiments in [HHN+02] indicate that Chromium is working quite well when the rendering problem is fill-rate limited. This is due to the fact that the OpenGL commands and a non-critical amount of rendering data can be distributed to multiple nodes without significant problems and since the critical fill-rate work is then performed locally on the graphics hardware.

Chromium also provides some facilities for parallel application development, namely a sort-last, binary-swap compositing SPU and an OpenGL extension providing synchronization primitives, such as a barrier and semaphore. It leaves other problems, such as configuration, task decomposition as well as process and thread management unaddressed. Parallel Chromium applications tend to be written for one specific parallel rendering use case, such as for example the sort-first distributed memory volume renderer [BHPB03] or the sort-last parallel volume renderer raptor [Hou05]. We are not aware of a generic Chromium-based application using many-to-one sort-first or stereo decompositions.

The concept of transparent OpenGL interception popularized by WireGL and Chromium has received some further contributions. While some commercial implementations such as TechViz and MechDyne Conduit continue to exist, on the research side only ClusterGL [NHM11] has been presented recently. ClusterGL employs the same approach as Chromium, but delivers a significantly faster implementation of transparent OpenGL interception and distribution for parallel rendering. Transparent OpenGL interception is an appealing apporach for some applications since it requires no code changes, but it has inherent limitiations due to the fact that eventually the bottleneck becomes the single-threaded application rendering code, the amount of application data the single application instance can load or process, or the the size of the OpenGL command stream send over the network.

CGLX [DK11] tries to bring parallel execution transparently to OpenGL applications, by emulating the GLUT API and intercepting certain OpenGL calls. In contrast to frameworks like Chromium and ClusterGL which distribute OpenGL calls, CGLX follows the distributed application approach. This works transparently for trivial applications, but quickly requires the application developer to address the complexities of a distributed application, when mutable application state needs to be synchronized across processes. For realistic applications, writing parallel applications remains the only viable approach for scalable parallel rendering, as shown by the success of Paraview, Visit and Equalizer-based applications.

OpenGL Multipipe SDK (MPK) [BRE05] implemented an effective parallel rendering API for a shared memory multi-CPU/GPU system. It is similar to IRIS Performer [RH94] in that it handles multi-GPU rendering by a lean abstraction layer via a conceptual callback mechanism, and that it runs different application tasks in parallel. However, MPK is not designed nor meant for rendering nodes separated by a network. MPK focuses on providing a parallel rendering framework for a single application, parts of which are run in parallel on multiple rendering channels, such as the culling, rendering and final image compositing processes.

Software for driving and interacting with tiled display walls has received significant attention, including Sage [DLR+09] and Sage 2 [MAN+14] in particular. Sage was built entirely around the concept of a shared framebuffer where all content windows are separate applications using pixel streaming but is no longer actively supported. Sage 2 is a complete, browser-centric reimplementation where each application is a web application distributed across browser instances. DisplayCluster [JAW+12], and its continuation Tide [Blu16], also implement the shared framebuffer concept of Sage, but provide a few native content applications integrated into the display servers. These solutions implement a scalable display environment and are a target display platform for scalable 3D graphics applications.

1.4 Dissertation Structure

2 Contributions

2.1 Generic Parallel Rendering Framework

2.2 New Parallel Rendering Modes

2.3 Load Balancing

2.4 Applications

3 The Architecture of a Parallel Rendering Framework

entities, roles, and relationships

3.1 Asynchronous Execution Model

3.2 Decomposition

3.3 Load Balancing

3.4 Compositing

4 New Decomposition Modes

5 Compositing Optimisations

6 New Load Balancing Algorithms

7 Data Distribution and Synchronization

8 Conclusion

8.1 Future Work

9 Problem Statement

Visualization of large amounts of data has been always been encumbered by sufficient system software. Compared to other domains, such as HPC simulations, large data visualization has received relatively little attention in both research and development. Consequently, there is a large amount of data which has not been explored sufficiently. In particular in scientific visualization, where data is often spatial and temporal, even simple visualizations can extract new information and provide a valuable tool to domain scientists for discovery.

The central theme of the proposed research is therefore: **How can we improve the capabilities of existing visualization algorithms for rendering large amounts of data?** This generic problem can be researched more concretely along the following research questions:

1. How can we improve the rendering performance of visualization applications to enable users to explore more data?
   (a) What new algorithms will decrease the time needed to composite rendering results, in particular for sort-last rendering?
   (b) How can we improve load-balancing for sort-first rendering, in particular for large display systems?
2. How can we reduce end-to-end system latency for better user experience?
   (a) In a generic parallel rendering framework, how can we schedule the different rendering stages to minimize the latency for the user?
   (b) How can we architect the parallel rendering framework to minimize synchronization between threads?
3. How can we maximize the impact of this research on large data scientists?

The obvious solution to the problem is to utilize more compute resources to parallelize and scale the rendering algorithm. The goal can either be to increase strong scaling (render a given data set faster) or weak scaling (render a larger data set at roughly the same speed). To efficiently use more resources, we need to research increasing the parallelism of existing algorithms, and to reduce the bottleneck in the image compositing stage.

An important collateral problem is the overall latency of the rendering system, that is, the time between a user input and its resulting output frame. While this is lower-bounded by the framerate of the rendering, oftentimes algorithmic or implementation choices increase the total system latency. Often this is a side effect of improving the rendering performance, but it also decreases the usability of the application for interactive usage. One typical example is pipelining of operations in the rendering pipeline.

Visualizing large amounts of data often goes hand in hand with the usage of high-resolution displays. Since large amounts of data tends to have a lot of detail, high-resolution desktop screens (4K or 8K resolution), as well as high-resolution display walls, help tremenduously in recognising and understanding details of the data. The high resolution however aggravetes all the aforementioned problems: The increased pixel count reduces rendering performance, requires better compositing algorithms, and increases the latency due to longer transfer times during compositing and display. Since pixel count increases quadratically with display size, this problem will become more important as display resolution increases.

10 Proposed Solution

Parallel rendering has received a lot of attention in the last couple of decades, yet libraries and frameworks to develop parallel rendering applications are scarcely available. Consequently, there are only a few applications which can utilize parallelism for rendering, let alone do so efficiently at scale. This research proposes to address these shortcomings by developing **reusable software components** to make parallel rendering programs easier to develop, by generalizing existing research into reusable software implementations.

Based on these foundations, we propose to research new algorithms to improve rendering performance. In particular we see potential in improving the scalability through better **load balancing**, **image compositing** algorithms, and **holistic optimization** of the whole rendering pipeline under control of our framework. Orthogonally to this algorithmic research, we propose to research **data processing and data access** strategies for parallel rendering applications. This is particularly important, yet underdeveloped, in the context of the visual analysis for HPC simulation results.

Previous parallel rendering approaches typically failed in one of the following system requirements:

1. generic application support, instead of domain-specific solution
2. scalable abstraction of the graphics layer
3. exploit existing code infrastructure, such as proprietary scene graphs, molecular data structures, level-of-detail and geometry databases

To date, generic and scalable parallel rendering frameworks that can be adopted to a wide range of scientific visualization domains are not yet readily available. Furthermore, flexible configurability to arbitrary cluster and multi-display configurations has also not been addressed in the past, but is of immense practical importance to scientists depending on high-performance interactive visualization as a scientific tool. We propose a novel flexible framework for parallel rendering that supports scalable performance, configuration flexibility, is minimally invasive with respect to adapting existing visualization applications, and is applicable to virtually any scientific visualization application domain.

To that end, this work aims to significantly advance the system design and implementation of flexible, distributed and cluster-parallel rendering frameworks as well as algorithms and system design for large data processing in the context of interactive visualization. The core of this proposal is the Equalizer project, a foundation for scalable, multi-GPU visualization software in all application domains. The main contributions of such a parallel rendering system are:

1. novel concept for flexible runtime configuration of graphics system resources
2. easy specification of parallel task decomposition and image compositing algorithms
3. automatic decomposition and distributed execution of rendering tasks according to the configuration
4. support for polygonal and volume rendering for opaque and transparent geometries
5. fully decentralized software architecture providing network swap barrier synchronization and data distribution functionality

6. support for low-latency distributed frame synchronization and image compositing
7. minimally invasive programming model

The broader impact of this work revolves around the development and improvement of a generic, flexible and scalable parallel rendering infrastructure applicable to a large number of application domains. The expected improvements of the proposed activities in distributed parallel parallel rendering will be integrated into open source software libraries and as such will be available to the general public and especially to developers of high-performance visualization and interactive rendering applications. This approach will maximize the impact of this research on large data scientists (research question 4).

A strong focus during the development is to architect the framework for scalability to address research questions 1 and 2. Based on my previous work and the study of existing implementations, scalability in parallel rendering today is mostly limited by excessive synchronization between execution threads, imbalance in the task decomposition, and compositing performance. Equalizer only has the following necessary synchronization points:

1. Swap synchronization between output channels to the same display system
2. Finalization of rendering frames given a configurable latency for all render threads
3. Availability of image data for compositing between the source and destination threads

This architecture has proven to provide good scalability by inherently allowing the pipelining of data synchronization, rendering and compositing tasks (research question 1), while simultaneously minimizing the time to display results by letting render threads execute as early as possible (research question 2). Furthermore, this will provide better scalability for the more specific research questions 1(a) and 1(b).

The following publications are planned or already published to address the corresponding research question:

1. (a) In [EP07, MEP10, EBA+12] we presented different algorithms to optimize the compositing step for sort-last rendering and optimisations for modern multi-GPU NUMA nodes.
   (b) In [EEP11, SPEP16] we presented novel load-balancing algorithms for sort-first rendering.
2. Our first system paper [EMP09] introduces the architecture of a parallel rendering framework with minimal synchronization and optimized task scheduling, including an in-depth experimental analysis. [EBA+12] introduced further scheduling optimizations.
3. Our foundation systems paper [EMP09] and applications work [HBB+13] provide evidence on the sustainability of our approach. We submitted a follow-on systems paper to ACM Transactions on Visualization and Graphics [ESP18]. This paper also contains background on applications and integrations of Equalizer in other software packages.

Based on this generic parallel rendering framework, we propose to research concrete algorithms and applications. We propose an engineering-driven approach which will analyse existing algorithms, improve them incrementally by focusing on one aspect of a large data application, and then compare our new research against existing work. Since our research questions are largely performance related, this comparison will be in most cases performed through benchmarking. In particular, we see potential in:

Load-balancing for rendering resources: While basic algorithms have been proposed for reactive and proactive load-balancing of simple rendering tasks, research is still needed for improving the resource utilisation for large-scale parallelization, as well as for rendering in more complex multi-display environments such as tiled display walls and immersive installations. The results of this research is directly measurable through application benchmarking of representative data sets. We consider this research goal achieved if we proposed and implemented new algorithms which can consistently deliver better performance over existing work, addressing research question 1(b).

Compositing of the rendering results: Previous research has focused on the scalability for very large scale HPC runs in the order of hundreds of thousands of cores, which are oftentimes not interactive by nature. We propose to improve image compositing performance for interactive applications on medium-sized (up to hundreds of GPUs) visualization clusters through analysing and optimising image compositing algorithms. As with load-balancing, this area of research can be considered achieved if benchmarks show consistent improvement over state of the art algorithms, addressing research question 1(a).

Applications for parallel rendering: While a few parallel rendering applications exist, developing them is still a significant undertaking. We propose to extend existing rendering applications and algorithms for scientific visualization for parallel rendering. Not only will this create new results and capabilites for large data visualization, it also improves the general applicability and ease of use of our generic parallel rendering components. We consider this goal achieved if our framework is used in multiple visualization applications. A side effect of this goal is addressing research question 4.

Data management for visualization of HPC data: We see a substantial potential in combining big data management strategies from the cloud computing domain to processing and visualising HPC simulation data. Paradoxically, storage systems for HPC are often optimised for large, sequential access, which is predominant during write, but not typical for analysis and visualization which use more, but smaller scattered read accesses. This is an exploratory research goal, where we hope to demonstrate the usefulness of cloud computing storage systems for HPC storage and linked large data visualizations. We consider this goal reached if we evaluated multiple approaches to data storage against their traditional parallel filesystem implementation, and can make recommendations for future research. The evaluation will again be benchmark-based, by measuring the time to solution for typical data access patterns.

This research will have a sustainable impact on how we use large-scale visualization systems as their commoditization makes them affordable to many more organizations. This is due to the research approach of using an incremental, engineering-driven and data-validated strategy, open source implementation of most software artefacts, the development of high-quality foundations, as well as the collaboration with both research and industry partners during the research.

The proposed research will have a direct, significant impact on accelerating the simulation-based research performed in the Blue Brain Project. On one hand, the data distribution capabilities will allow faster development of scalable, neuroscience-specific visualization applications for the BBP. This is of particular importance as we foresee the need to visualize different modalities at different brain scales as the simulations grow in complexity and data size in the future.

11 Research Plan

The proposed research is heavily based on prior software engineering work in the domain, and will leverage a strong parallel rendering system to enable novel research within a non-trivial software stack. Consequently, a large portion of the plan is the development of the parallel rendering framework, where the timeline has been reverse-engineered from the work performed leading to this proposal.

The integration of these frameworks into applications is not part of this research schedule, since it is a non-research activity. These developments have been and will be funded by other means. This work is nevertheless an important part of this project, as it validates the general applicability of this research in academia and industry.

The research is structured as follows:

M1-3: System Architecture: Outline the general system architecture, configuration structure and entities, class hierarchy and API. Research third-party technologies to be used in the implementation.

M4-10: Distributed Execution Layer: First iteration of the implementation of the distributed execution layer allowing dynamic configurations and communication patterns.

M10-16: Multi-Display Parallel Rendering Framework: Based on the distributed execution layer, develop a first parallel rendering framework capable of driving multi-display environments for monoscopic rendering, including the automatic launch of the rendering processes from the main application.

M17: Stereoscopic Rendering: Implement stereoscopic rendering using configurable interocular distance.

M18: Immersive Rendering: Head-tracking API and configuration entities, calculation of corresponding off-axis frusta.

M19: Scalable Rendering Architecture: Design configuration and class hierarchy for scalable rendering modes, including task decomposition and parallel compositing algorithms.

M20-24: Basic Scalable Rendering: Implement basic decompositions (2D, DB, Eye) and corresponding parallel compositing algorithms (2D, direct-send, binary-swap).

M25-30: Advanced Scalable Rendering: Implement advanced compounds (Pixel, DPlex) and compositing optimizations (realtime image compression, region of interest, etc.).

M30-32: First Publication: Benchmarking and systems paper.

M33-36: Load Balancing: Implement load-balancing for multi-display setups. improved automatic load-balancing using region of interest.

M37-38: Second Publication: Benchmarking and load-balancing paper.

M39-42: Compositing Research: Research compositing optimizations.

M43-44: Third Publication: Benchmarking and compositing paper.

PhD Proposal

M45-46: Fourth Publication: Benchmarking and updated systems paper.

M47-52: Dissertation: Write and defend dissertation.

References

[AG95] Gennady Agranov and Craig Gotsman. Algorithms for rendering realistic terrain image sequences and their parallel implementation. *The Visual Computer*, 11(9):455–464, 1995.

[AGL+02] J. Allard, V. Gouranton, L. Lecointre, E. Melin, and B. Raffin. NetJuggler: Running VR Juggler with multiple displays on a commodity component cluster. In *Proceeding IEEE Virtual Reality*, pages 275–276, 2002.

[AP98] James Ahrens and James Painter. Efficient sort-last rendering using compression-based image compositing. In *Proceedings Eurographics Workshop on Parallel Graphics and Visualization*, 1998.

[BBDZ00] W. Blanke, C. Bajaj, D.Fussel, and X. Zhang. The metabuffer: A scalable multi-resolution 3-d graphics system using commodity rendering engines. Technical Report TR2000-16, University of Texas at Austin, 2000.

[BCN03] Allen Bierbaum and Carolina Cruz-Neira. ClusterJuggler: A modular architecture for immersive clustering. In *Proceedings Workshop on Commodity Clusters for Virtual Reality, IEEE Virtual Reality Conference*, 2003.

[BHPB03] Wes E. Bethel, Greg Humphreys, Brian Paul, and J. Dean Brederson. Sort-first, distributed memory parallel visualization and rendering. In *Proceedings IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 41–50, 2003.

[BJH+01] Allen Bierbaum, Christopher Just, Patrick Hartling, Kevin Meinert, Albert Baker, and Carolina Cruz-Neira. VR Juggler: A virtual platform for virtual reality application development. In *Proceedings of IEEE Virtual Reality*, pages 89–96, 2001.

[Blu16] Blue Brain Project. Tide: Tiled Interactive Display Environment. https://github.com/BlueBrain/Tide, 2016.

[BRE05] Praveen Bhaniramka, Philippe C. D. Robert, and Stefan Eilemann. OpenGL Multipipe SDK: A toolkit for scalable parallel rendering. In *Proceedings IEEE Visualization*, pages 119–126, 2005.

[CKS02] Wagner T. Correa, James T. Klosowski, and Claudio T. Silva. Out-of-core sort-first parallel rendering for cluster-based tiled displays. In *Proceedings Eurographics Workshop on Parallel Graphics and Visualization*, pages 89–96, 2002.

[CM06] Xavier Cavin and Christophe Mion. Pipelined sort-last rendering: Scalability, performance and beyond. In *Proceedings Eurographics Symposium on Parallel Graphics and Visualization*, 2006.

[CMF05] Xavier Cavin, Christophe Mion, and Alain Filbois. COTS cluster-based sort-last rendering: Performance evaluation and pipelined implementation. In *Proceedings IEEE Visualization*, pages 111–118. Computer Society Press, 2005.

[Cro97] T. W. Crockett. An introduction to parallel rendering. *Parallel Computing*, 23:819–843, 1997.

[DK11] Kai-Uwe Doerr and Falko Kuester. CGLX: A scalable, high-performance visualization framework for networked display environments. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):320–332, March 2011.

[DLR+09] Thomas A. DeFanti, Jason Leigh, Luc Renambot, Byungil Jeong, Alan Verlo, Lance Long, Maxine Brown, Daniel J. Sandin, Venkatram Vishwanath, Qian Liu, Mason J. Katz, Philip Papadopoulos, Joseph P. Keefe, Gregory R. Hidley, Gregory L. Dawe, Ian Kaufman, Bryan Glogowski, Kai-Uwe Doerr, Rajvikram Singh, Javier Girado, Jurgen P. Schulze, Falko Kuester, and Larry Smarr. The optiportal, a scalable visualization, storage, and computing interface device for the optiputer. *Future Gener. Comput. Syst.*, 25(2):114–123, February 2009.

[EBA+12] Stefan Eilemann, Ahmet Bilgili, Marwan Abdellah, Juan Hernando, Maxim Makhinya, Renato Pajarola, and Felix Schürmann. Parallel Rendering on Hybrid Multi-GPU Clusters. In *EGPGV*, pages 109–117, 2012.

[EEP11] Fatih Erol, Stefan Eilemann, and Renato Pajarola. Cross-segment load balancing in parallel rendering. In *Proceedings Eurographics Symposium on Parallel Graphics and Visualization*, pages 41–50, 2011.

[EMP+97] John Eyles, Steven Molnar, John Poulton, Trey Greer, Anselmo Lastra, Nick England, and Lee Westover. PixelFlow: The realization. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics Hardware*, pages 57–68, 1997.

[EMP09] Stefan Eilemann, Maxim Makhinya, and Renato Pajarola. Equalizer: A scalable parallel rendering framework. *IEEE Transactions on Visualization and Computer Graphics*, 15(3):436–452, May/June 2009.

[EP07] Stefan Eilemann and Renato Pajarola. Direct send compositing for parallel sort-last rendering. In *Proceedings Eurographics Symposium on Parallel Graphics and Visualization*, 2007.

[ESP18] Stefan Eilemann, David Steiner, and Renato Pajarola. Equalizer 2.0 - Convergence of a Parallel Rendering Framework, 2018.

[GS02] Antonia Garcia and Han-Wei Shen. An interleaved parallel volume renderer with PC-clusters. In *Proceedings Eurographics Workshop on Parallel Graphics and Visualization*, pages 51–60, 2002.

[HBB+13] J. B. Hernando, J. Biddiscombe, B. Bohara, S. Eilemann, and F. Schürmann. Practical Parallel Rendering of Detailed Neuron Simulations. In *Proceedings of the 13th Eurographics Symposium on Parallel Graphics and Visualization*, EGPGV, pages 49–56, Aire-la-Ville, Switzerland, Switzerland, 2013. Eurographics Association.

[HBEH00] Greg Humphreys, Ian Buck, Matthew Eldridge, and Pat Hanrahan. Distributed rendering for scalable displays. *IEEE Supercomputing*, October 2000.

[HEB+01] Greg Humphreys, Matthew Eldridge, Ian Buck, Gordan Stoll, Matthew Everett, and Pat Hanrahan. WireGL: A scalable graphics system for clusters. In *Proceedings Annual Conference on Computer Graphics and Interactive Techniques*, pages 129–140, 2001.

[HH99] Greg Humphreys and Pat Hanrahan. A distributed graphics system for large tiled displays. *IEEE Visualization 1999*, pages 215–224, October 1999.

[HHN+02] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: A stream-processing framework for interactive rendering on clusters. *ACM Transactions on Graphics*, 21(3):693–702, 2002.

[Hou05] Mike Houston. Raptor. http://graphics.stanford.edu/projects/raptor/, 2005.

[HSC+00] Jian Huang, Naeem Shareef, Roger Crawfis, P. Sadayappan, and Klaus Mueller. A parallel splatting algorithm with occlusion culling. In *Proceedings Eurographics Workshop on Parallel Graphics and Visualization*, 2000.

[ISH98] Homan Igehy, Gordon Stoll, and Pat Hanrahan. The design of a parallel graphics interface. *Proceedings of SIGGRAPH 98*, pages 141–150, July 1998.

[JAW+12] G. P. Johnson, G. D. Abram, B. Westing, P. Navr'til, and K. Gaither. DisplayCluster: An Interactive Visualization Environment for Tiled Displays. In *2012 IEEE International Conference on Cluster Computing*, pages 239–247, Sept 2012.

[JBBCN98] Christopher Just, Allen Bierbaum, Albert Baker, and Carolina Cruz-Neira. VR Juggler: A framework for virtual reality development. In *Proceedings Immersive Projection Technology Workshop*, 1998.

[JDB+04] Ken Jones, Chrystie Danzer, Jenn Byrnes, Karen Jacobson, Patrick Bouchaud, Davy Courvoisier, Stefan Eilemann, and Philippe Robert. SGI®OpenGL Multipipe™SDK User's Guide. Technical Report 007-4239-004, Silicon Graphics, 2004.

[JLMVK06] Andrew Johnson, Jason Leigh, Paul Morin, and Peter Van Keken. GeoWall: Stereoscopic visualization for geoscience research and education. *IEEE Computer Graphics and Applications*, 26(6):10–14, November-December 2006.

[LDC96] P. Peggy Li, William H. Duquette, and David W. Curkendall. RIVA: A versatile parallel rendering system for interactive scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):186–201, 1996.

[Lev04] Paul G. Lever. SEPIA – applicability to MVC. White paper Manchester Visualization Centre (MVC), University of Manchester, 2004.

[LMS+01a] Santiago Lombeyda, Laurent Moll, Mark Shand, David Breen, and Alan Heirich. Scalable interactive volume rendering using off-the-shelf components. Technical Report CACR-2001-189, California Institute of Technology, 2001.

[LMS+01b] Santiago Lombeyda, Laurent Moll, Mark Shand, David Breen, and Alan Heirich. Scalable interactive volume rendering using off-the-shelf components. In *Proceedings IEEE Symposium on Parallel and Large Data Visualization and Graphics*, pages 115–121, 2001.

[LWMT97] Peggy P. Li, Scott Whitman, Roberto Mendoza, and James Tsiao. ParVox: A parallel splatting volume rendering system for distributed visualization. In *Proceedings IEEE Parallel Rendering Symposium*, pages 7–14, 1997.

[MAN+14] Thomas Marrinan, Jillian Aurisano, Arthur Nishimoto, Krishna Bharadwaj, Victor Mateevitsi, Luc Renambot, Lance Long, Andrew Johnson, and Jason Leigh. SAGE2: A new approach for data intensive collaboration using Scalable Resolution Shared Displays. In *Collaborative Computing: Networking, Applications and Worksharing*, pages 177–186, 2014.

[MCEF94] Steve Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, 1994.

[MEP92] Steven Molnar, John Eyles, and John Poulton. PixelFlow: High-speed rendering using image composition. In *Proceedings ACM SIGGRAPH*, pages 231–240, 1992.

[MEP10] M. Makhinya, S. Eilemann, and R. Pajarola. Fast Compositing for Cluster-Parallel Rendering. In *Proceedings of the 10th Eurographics Conference on Parallel Graphics and Visualization*, EGPGV, pages 111–120, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[MHS99] Laurent Moll, Alan Heirich, and Mark Shand. Sepia: scalable 3D compositing using PCI pamette. In *Proceedings IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 146–155, 1999.

[MOM+01] Shigeru Muraki, Masato Ogata, Kwan-Liu Ma, Kenji Koshizuka, Kagenori Kajihara, Xuezhen Liu, Yasutada Nagano, and Kazuro Shimokawa. Next-generation visual supercomputing using PC clusters with volume graphics hardware devices. In *Proceedings ACM/IEEE Conference on Supercomputing*, pages 51–51, 2001.

[MPK] OpenGL Multipipe SDK.

[Mue95] Carl Mueller. The sort-first rendering architecture for high-performance graphics. In *Proceedings Symposium on Interactive 3D Graphics*, pages 75–84. ACM SIGGRAPH, 1995.

[Mue97] Carl Mueller. Hierarchical graphics databases in sort-first. In *Proceedings IEEE Symposium on Parallel Rendering*, pages 49–. Computer Society Press, 1997.

[NC07] Krzysztof Niski and Jonathan D. Cohen. Tile-based level of detail for the parallel age. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1352–1359, November/December 2007.

[NHM11] Braden Neal, Paul Hunkin, and Antony McGregor. Distributed OpenGL rendering in network bandwidth constrained environments. In Torsten Kuhlen, Renato Pajarola, and Kun Zhou, editors, *Proceedings Eurographics Conference on Parallel Graphics and Visualization*, pages 21–29. Eurographics Association, 2011.

[NSJ+05] Weifang Nie, Jizhou Sun, Jing Jin, Xiaotu Li, Jie Yang, and Jiawan Zhang. A dynamic parallel volume rendering computation mode based on cluster. In *Proceedings Computational Science and its Applications*, volume 3482 of *Lecture Notes in Computer Science*, pages 416–425, 2005.

[RH94] John Rohlf and James Helman. IRIS Performer: A high performance multiprocessing toolkit for real-time 3D graphics. In *Proceedings ACM SIGGRAPH*, pages 381–394. ACM Press, 1994.

[SEP+01] Gordon Stoll, Matthew Eldridge, Dan Patterson, Art Webb, Steven Berman, Richard Levy, Chris Caywood, Milton Taveira, Stephen Hunt, and Pat Hanrahan. Lightning-2: A high-performance display subsystem for PC clusters. In *Proceedings ACM SIGGRAPH*, pages 141–148, 2001.

[SFL01] Rudrajit Samanta, Thomas Funkhouser, and Kai Li. Parallel rendering with K-way replication. In *Proceedings IEEE Symposium on Parallel and Large-Data Visualization and Graphics*. Computer Society Press, 2001.

[SFLS00] Rudrajit Samanta, Thomas Funkhouser, Kai Li, and Jaswinder Pal Singh. Hybrid sort-first and sort-last parallel rendering with a cluster of PCs. In *Proceedings Eurographics Workshop on Graphics Hardware*, pages 97–108, 2000.

[SL02] Jürgen P. Schulze and Ulrich Lang. The parallelization of the perspective shear-warp volume rendering algorithm. In *Proceedings Eurographics Workshop on Parallel Graphics and Visualization*, pages 61–70, 2002.

[SML+03] Aleksander Stompel, Kwan-Liu Ma, Eric B. Lum, James Ahrens, and John Patchett. SLIC: Scheduled linear image compositing for parallel volume rendering. In *Proceedings IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 33–40, 2003.

[SPEP16] David Steiner, Enrique G. Paredes, Stefan Eilemann, and Renato Pajarola. Dynamic work packages in parallel rendering. In *Proceedings Eurographics Symposium on Parallel Graphics and Visualization*, pages 89–98, 2016.

[SWNH03] Oliver G. Staadt, Justin Walker, Christof Nuber, and Bernd Hamann. A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering. In *Proceedings Eurographics Workshop on Virtual Environments*, pages 261–270, 2003.

[SZF+99] Rudrajit Samanta, Jiannan Zheng, Thomas Funkhouser, Kai Li, and Jaswinder Pal Singh. Load balancing for multi-projector rendering systems. In *Proceedings Eurographics Workshop on Graphics Hardware*, pages 107–116, 1999.

[VR91] Guy Vezina and Philip K. Robertson. Terrain perspectives on a massively parallel SIMD computer. In *Proceedings Computer Graphics International (CGI)*, pages 163–188, 1991.

[Wit98] Craig M. Wittenbrink. Survey of parallel volume rendering algorithms. In *Proceedings Parallel and Distributed Processing Techniques and Applications*, pages 1329–1336, 1998.

[YYC01] Don-Lin Yang, Jen-Chih Yu, and Yeh-Ching Chung. Efficient compositing methods for the sort-last-sparse parallel volume rendering system on distributed memory multicomputers. *Journal of Supercomputing*, 18(2):201–22–, February 2001.

[ZBB01] Xiaoyu Zhang, Chandrajit Bajaj, and William Blanke. Scalable isosurface visualization of massive datasets on COTS clusters. In *Proceedings IEEE Symposium on Parallel and Large Data Visualization and Graphics*, pages 51–58, 2001.