Equalizer: A Mature Parallel Rendering Framework

Stefan Eilemann* Renato Pajarola†

Visualization and MultiMedia Lab
Department of Informatics
University of Zürich

Abstract— We present the features, algorithms and system integration necessary to implement a parallel rendering framework usable in a wide range of real-world applications, based on the basic architecture and implementation of the Equalizer parallel rendering framework presented in [13].

Index Terms—Parallel Rendering, Scalable Visualization, Cluster Graphics, Immersive Environments, Display Walls

1 Introduction

The continuing improvements in hardware integration lead to ever faster CPUs and GPUs, as well as higher resolution sensor and display devices. Moreover, increased hardware parallelism is applied in form of multi-core CPU workstations, massive parallel super computers, or cluster systems. Hand in hand goes the rapid growth in complexity of data sets from numerical simulations, high-resolution 3D scanning systems or bio-medical imaging, which causes interactive exploration and visualization of such large data sets to become a serious challenge. It is thus crucial for a visualization solution to take advantage of hardware accelerated scalable parallel rendering. In this systems paper we describe a new scalable parallel rendering framework called *Equalizer* that is aimed primarily at cluster-parallel rendering, but works as well in a shared-memory system. Cluster systems are the main focus because workstation graphics hardware is developing faster than high-end graphics (super-) computers can absorb new developments, and also because clusters offer a better cost-performance balance.

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

a) generic application support, instead of special domain solution
b) scalable abstraction of the graphics layer
c) 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 display-wall configurations has also not been addressed in the past, but is of immense practical importance to scientists depending high-performance interactive visualization as a scientific tool. In this paper we present Equalizer, which is 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.

The main contributions that Equalizer introduces in a single parallel rendering system, and which are presented in this paper are:

i) novel concept of compound trees for flexible configuration of graphics system resources,
ii) easy specification of parallel task decomposition and image compositing choice through compound tree layouts,
iii) automatic decomposition and distributed execution of rendering tasks according to compound tree,
iv) support for parallel surface as well as transparent (volume) rendering through z-visibility as well as α-blending compositing,
v) fully decentralized architecture providing network swap barrier (synchronization) and distributed objects functionality,
vi) support for low-latency distributed frame synchronization and image compositing,
vii) minimally invasive programming model.

Equalizer is open source, available under the LGPL license from http://www.equalizergraphcis.com/, which allows it to be used both for open source and commercial applications. It is source-code portable, and has been tested on Linux, Microsoft Windows, and Mac OS X in 32 and 64 bit mode using both little endian and big endian processors.

2 Related Work

The early fundamental concepts of parallel rendering have been laid down in [29] and [12]. 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.

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 [25, 42, 17, 38, 15, 32] or geo-visualization [41, 2, 24, 20]. 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.

Recently in [33], 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 [33] such as load management and adaptive LOD data traversal can be carried over to other sort-first parallel rendering solutions.

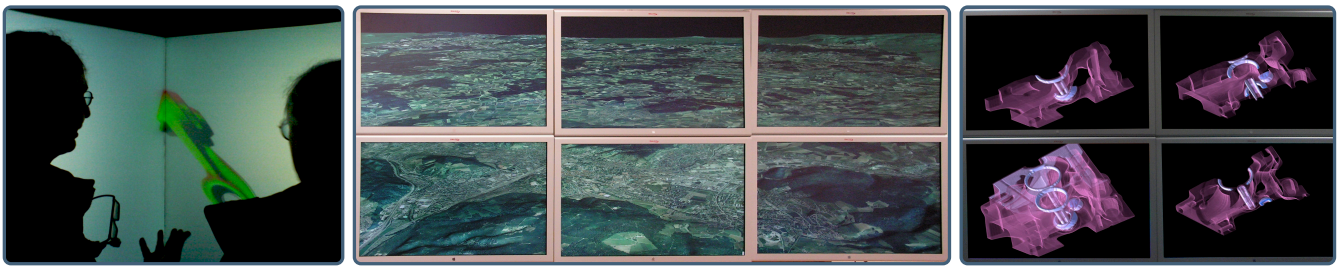
Special-purpose architectures

Traditionally, high-performance real-time rendering systems have relied on an integrated proprietary system architecture, such as the 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. However, cluster systems need more sophisticated parallel graphics rendering libraries, such as the one proposed in this paper.

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

*email: eilemann@gmail.com
†email: pajarola@acm.org

Manuscript received 5 February 2008; accepted ? April 2008; posted online ? June 2008.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.



(a) (b) (c)

Fig. 1. Various Equalizer use cases: (a) immersive CAVE, (b) display wall and (c) scalable volume rendering.

[28, 23], Sepia 2 [26, 27], Lightning 2 [**?**], Metabuffer [**?**, **?**], MPC Compositor [**?**] and PixelFlow [**?**, **?**], 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 and other modern interconnects, combinations of software and GPU-based solutions offer more flexibility at comparable performance.

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 [30, 31] (sort-first architecture), [37, 36] (load balancing), [35] (data replication), or [10, 9] (scalability). On the other hand, specific algorithms have been developed for cluster based rendering and compositing such as [3], [11] and [43, 40]. 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 [**?**]. Only few generic APIs and (cluster-) parallel rendering systems exist which include VR Juggler [8] (and its derivatives), Chromium [19] (an evolution of [**?**, **?**, 18]) and OpenGL Multipipe SDK [21, 6, 1].

VR Juggler [8, 22] 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 [7] and NetJuggler [4], 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 [39]. The presented system is different from VR Juggler in that it fully supports scalable parallel rendering such as sort-first and sort-last task decomposition and image compositing, it provides more flexible node configurations which for example allow specifying arbitrary task decomposition and image compositing combinations as simple compound layouts. Furthermore, it it is fully distributed which includes support for network swap barriers (synchronization), distributed objects as well as image compression and transmission. In contrast to VR Juggler, Equalizer supports multiple rendering threads per process, which is important for multi-GPU systems.

While Chromium [19] 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 [39]. 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. While preserving a minimally invasive API, the novel proposed system is better aimed at scalability as the actual data access is decentralized in the distributed rendering clients.

The performance experiments in [19] 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, thus making the development of parallel OpenGL applications harder than with Equalizer. 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 [5] or the sort-last parallel volume renderer raptor [16]. We are not aware of a generic Chromium-based application using many-to-one sort-first or stereo decompositions. This is another difference to Equalizer which provides a much more flexible task decomposition configuration. Applications written once for Equalizer can easily be run in any different task decomposition mode and for any physical display configuration without any changes to the application itself. While Equalizer provides an abstraction of all entities of the rendering pipeline (see Sections **??** and **??**), Chromium's infrastructure is primarily the compositing stage.

OpenGL Multipipe SDK (MPK) [6] implements an effective parallel rendering API for a shared memory multi-CPU/GPU system. It is similar to IRIS Performer [34] in that it handles multi-pipe 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. Compared to MPK, Equalizer supports a fully distributed parallel rendering paradigm and features a more flexible task decomposition approach.

3 USABILITY

3.1 Physical and Logical Visualization Setup

Canvas, Layout, Observer

multi-view with runtime switching,
2D overlays
Swap Barriers

3.2 Auto-Configuration

hwsd with VirtualGL support

3.3 Sequel

application, renderer, view data

3.4 Qt Windowing

Challenges in threading model, architecture

3.5 Tide Integration

Tiled interactive display environment, parallel pixel streaming, events

4 The Collage Network Library

4.1 Distributed, Versioned Objects

types (instance, delta, static), versioning, multicast, compression, serializable with dirty bits, mapping, blocking commits

4.2 Reliable Stream Protocol

UDP-based reliability protocol

4.3 Infiniband RDMA

reverse-engr impl

5 Virtual Reality

5.1 Dynamic Focus Distance

Focus what user is looking at

5.2 Asymmetric Eye Position

Better HMD by measuring user geometry

5.3 Application-specific Scaling

Gullivers world

5.4 Runtime Stereo Switch

5.5 Swap Synchronization and GPU affinity

6 Performance

6.1 New Decomposition Modes

The initial version of Equalizer implemented sort-first (2D), sort-last (DB) and stereo (EYE) decomposition. In the following we present new decomposition modes and motivate their use case.

6.1.1 Time-Multiplex

Interaction with threading

6.1.2 Tiles and Chunks

Equalizers do what?

6.1.3 Pixel

Equal load for fill-limited apps

6.1.4 Subpixel

MSAA and DOF, plus idle refinement algo

6.2 Equalizers

Equalizer are one addition to compound trees. They modify parameters of their respective subtree at runtime to optimize one aspect of the decomposition.

6.2.1 Sort-First and Sort-Last Load Equalizer

reactive load equalization, describe tunables (damping, resistance, delta, granularity), sort-first respects input channel size

6.2.2 Cross-Segment Load Equalizer

summarize [14]

6.2.3 Dynamic Frame Resolution

Constant frame rate for fill-limited applications

6.2.4 Frame Rate Equalizer

Predominantly used by DPlex to smooth output framerate

6.2.5 Monitoring

Control workstation in VR setups

6.3 Optimizations

6.3.1 Region of Interest

for compositing and load equalizer.

6.3.2 Asynchronous Readback

pipelines rendering with compositing

6.3.3 Download and Compression Plugins

GPU-CPU transfer plugins with optional compression (eg YUV) linked to CPU compression for network transfer

6.3.4 Thread Synchronization Modes

Per-node sync, draw sync, async

7 Applications

7.1 Livre

7.2 RTT Deltagen

7.3 RTNeuron

8 Experimental Results

9 Discussion and Conclusion

Acknowledgements

We would like to thank and acknowledge the following institutions and projects for providing the 3D geometry and volume test data sets: the Digital Michelangelo Project, Stanford 3D Scanning Repository, Cyberware Inc., volvis.org and the Visual Human Project. This work was partially supported by the Swiss National Science Foundation Grant 200021-116329/1.

References

[1] OpenGL Multipipe SDK.

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

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

[4] 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.

[5] W. E. Bethel, G. Humphreys, B. Paul, and J. D. 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.

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

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

[8] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira. VR Juggler: A virtual platform for virtual reality application development. In *Proceedings of IEEE Virtual Reality*, pages 89–96, 2001.

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

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

[11] W. T. Correa, J. T. Klosowski, and C. 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.

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

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

[14] F. Erol, S. Eilemann, and R. Pajarola. Cross-segment load balancing in parallel rendering. In *Proceedings Eurographics Symposium on Parallel Graphics and Visualization*, pages 41–50, 2011.

[15] A. Garcia and H.-W. Shen. An interleaved parallel volume renderer with PC-clusters. In *Proceedings Eurographics Workshop on Parallel Graphics and Visualization*, pages 51–60, 2002.

[16] M. Houston. Raptor. http://graphics.stanford.edu/projects/raptor/, 2005.

[17] J. Huang, N. Shareef, R. Crawfis, P. Sadayappan, and K. Mueller. A parallel splatting algorithm with occlusion culling. In *Proceedings Eurographics Workshop on Parallel Graphics and Visualization*, 2000.

[18] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. WireGL: A scalable graphics system for clusters. In *Proceedings Annual Conference on Computer Graphics and Interactive Techniques*, pages 129–140, 2001.

[19] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: A stream-processing framework for interactive rendering on clusters. *ACM Transactions on Graphics*, 21(3):693–702, 2002.

[20] A. Johnson, J. Leigh, P. Morin, and P. Van Keken. GeoWall: Stereoscopic visualization for geoscience research and education. *IEEE Computer Graphics and Applications*, 26(6):10–14, November-December 2006.

[21] K. Jones, C. Danzer, J. Byrnes, K. Jacobson, P. Bouchaud, D. Courvoisier, S. Eilemann, and P. Robert. SGI®OpenGL Multipipe™SDK User's Guide. Technical Report 007-4239-004, Silicon Graphics, 2004.

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

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

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

[25] P. P. Li, S. Whitman, R. Mendoza, and J. Tsiao. ParVox: A parallel splatting volume rendering system for distributed visualization. In *Proceedings IEEE Parallel Rendering Symposium*, pages 7–14, 1997.

[26] S. Lombeyda, L. Moll, M. Shand, D. Breen, and A. Heirich. Scalable interactive volume rendering using off-the-shelf components. Technical Report CACR-2001-189, California Institute of Technology, 2001.

[27] S. Lombeyda, L. Moll, M. Shand, D. Breen, and A. 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.

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

[29] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, 1994.

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

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

[32] W. Nie, J. Sun, J. Jin, X. Li, J. Yang, and J. 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.

[33] K. Niski and J. 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.

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

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

[36] R. Samanta, T. Funkhouser, K. Li, and J. P. 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.

[37] R. Samanta, J. Zheng, T. Funkhouser, K. Li, and J. P. Singh. Load balancing for multi-projector rendering systems. In *Proceedings Eurographics Workshop on Graphics Hardware*, pages 107–116, 1999.

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

[39] O. G. Staadt, J. Walker, C. Nuber, and B. 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.

[40] A. Stompel, K.-L. Ma, E. B. Lum, J. Ahrens, and J. 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.

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

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

[43] D.-L. Yang, J.-C. Yu, and Y.-C. 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.