Recent Advances in the Equalizer Parallel Rendering Framework

Stefan Eilemann[*]
Blue Brain Project, EPFL and Eyescale Software GmbH

Abstract

In this poster we present the recent advances in Equalizer, a framework for scalable parallel rendering based on OpenGL, which provides an application programming interface (API) to develop scalable graphics applications for a wide range of systems ranging from large distributed visualization clusters and multi-processor multi-GPU graphics systems to single-GPU desktop machines.

Recent advances include optimizations for visualization clusters using multi-GPU NUMA nodes, tile and subpixel decompositions, automatic configuration on multi-GPU machines and scalable visualization clusters, as well as novel features for Virtual Reality, most notably support for dynamic focus distance.

Index Terms: I.3.2 [Computer Graphics]: Graphics Systems/Distributed Graphics—Parallel Rendering; I.3.m [Computer Graphics]: Miscellaneous/Rendering Clusters—Scalable Rendering; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism/Virtual Reality—Immersive Environments

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 biomedical 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 poster we present a scalable parallel rendering framework called Equalizer that is aimed primarily at cluster-parallel rendering, but works as well in a shared-memory system. Figure 1 shows the main use cases for parallel rendering.

In addition to the basic framework described in [4], we will present the latest research results around Equalizer, namely optimizations for hybrid multi-GPU clusters [3], new tile and subpixel decomposition schemes, automatic configuration, advanced Virtual Reality features as well as new Equalizer-based applications.

2 Parallel Rendering Framework

Equalizer is a framework to develop and deploy distributed and non-distributed parallel rendering applications. The programming interface is based on a set of C++ classes, modeled closely to the resource hierarchy of a graphics rendering system. The application subclasses these objects and overrides C++ task methods, similar to C callbacks. These task methods will be called in parallel by the framework, depending on the current configuration. This parallel rendering interface is significantly different from Chromium [7] and more similar to VRJuggler [2] and MPK [1]. The class framework and in particular its use is described in more detail in [4].

[*]e-mail: stefan.eilemann@epfl.ch

An Equalizer application does not have to select a particular rendering configuration itself; it is configured automatically using local and remote resource discovery, or manually using simple ASCII configuration files. The application is written only against a client library, which communicates with the configuration server that does not have to be touched by the developer. The server also launches and controls the distributed rendering clients provided by the application. Thus the application itself can run unmodified on any configuration from a simple workstation to large scale graphics clusters, multi-GPU workstations and Virtual Reality installations.

3 New Features

3.1 Hybrid Multi-GPU Clusters

Achieving efficient scalable parallel rendering for interactive visualization applications on medium-sized graphics clusters remains a challenging problem. Framerates of up to 60Hz require a carefully designed and fine-tuned parallel rendering implementation that fits all required operations into the 16ms time budget available for each rendered frame. Furthermore, modern commodity hardware embraces more and more a NUMA architecture, where multiple processor sockets each have their locally attached memory and where auxiliary devices such as GPUs and network interfaces are directly attached to one of the processors. Such so called *fat* NUMA processing and graphics nodes are increasingly used to build cost-effective *hybrid* shared/distributed memory visualization clusters. We present important optimizations to achieve highly interactive framerates on such hybrid multi-GPU clusters, such as asynchronous readbacks, automatic thread placement and a novel algorithm for the optimization of 2D load-balancing reusing region of interest information from the compositing stage for refined load distribution, decribed in more detail in [3].

3.2 Tile and Subpixel Compounds

We present two new decomposition schemes: tile and subpixel compounds.

Tile compounds are a variant of sort-first decomposition, as classified by [8]. They decompose the rendering in screen-space, where each rendering unit pulls and processes regular tiles of the final view. Tile compounds are ideal for purely fill-limited applications such as volume rendering and raytracing. The inversion from the traditional push model (master precomputes sort-first tiling) to a pull model leads to a near-ideal load balance for typical display resolutions and tile sizes.

Subpixel compounds decompose the rendering of multi-sampling algorithms such as anti-aliasing and depth-of-field rendering. Without code modifications in the application, subpixel compounds transparently provide software full-scene anti-aliasing (FSAA). Applications can also be modified to support any multi-sampling algorithm with an arbitrary number of samples, e.g, depth-of-field which continually adds samples when the application is idle.

3.3 Automatic Configuration

Automatic configuration is based on the GPU-SD library [5] for discovering local and remote GPUs. Based on this information, a typical configuration is dynamically compiled, which greatly facilitates the management of visualization resources. The auto-configuration

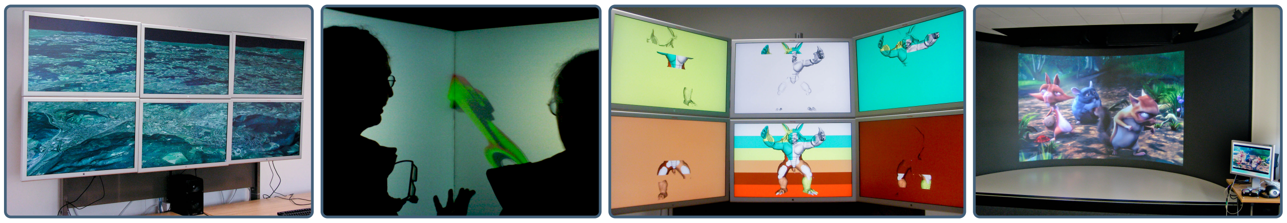
(a) (b) (c) (d)

Figure 1: Various Equalizer use cases: (a) display wall, (b) immersive CAVE (c) scalable rendering and (d) stereo projection with overlap.

uses the same format as the static configuration files, and can therefore be used to create template configurations.

The default local configuration discovers all installed GPUs in a system, and supports Windows, Mac OS X and Linux. The remote configuration relies on a daemon announcing locally discovered GPUs using zeroconf networking and a zeroconf discovery module to collect information on all GPUs announced by daemons.

3.4 Advanced Virtual Reality Support

We present support for advanced virtual reality installations and application features, such as support for head-mounted displays, multiple observers with separate tracking data and eye separation, as well as dynamic focus distance support.

The focal distance defines at which range the left and right eye converge into the same image. Objects placed at the focal distance do not have a stereo divergence. In typical implementations, the focal distance is equal to the distance of the physical projection wall, which causes discomfort when the observer is looking at an object behind or in front the physical wall. We have implemented support for changing the focal distance at runtime, e.g, to always focus the nearest object in the observer's view direction.

4 New Applications

We will present real-world applications based on our parallel rendering framework, demonstrating some use cases implemented on top of a generic rendering framework.

4.1 RTNeuron

RTNeuron [6] is a neuroscientific visualization application designed for the interactive visualization of detailed cortical circuit simulations. RTNeuron features several graphics techniques and algorithms tailored to the specific geometrical characteristics of neurons to allow the efficient rendering of neuronal circuits where simulation data is mapped onto the membrane meshes. Some of these features have to do with levels of detail, view frustum culling and correct alpha-blending.

To enable the visualization of very large circuits ($> 10,000$ neurons) at interactive speeds, RTNeuron makes use of Equalizer to implement two well known parallel rendering strategies, sort-first with load balancing and sort-last with a static decomposition of the scene. For sort-last two different modes are provided: round-robin distribution of neurons between the rendering nodes, and for transparent rendering, a spatial partition of the scene based on a heuristic kd-tree. Apart from parallel rendering, RTNeuron is also aware of the VR features provided by Equalizer and supports stereoscopic visualization in powerwalls and VR facilities alike.

4.2 osgScaleViewer

The osgScaleViewer is a scalable, cluster-ready viewer application based on the widely used OpenSceneGraph and Equalizer. It supports rendering multiple views of the same scene graph, using software or hardware swap synchronization, parallel, multi-threaded and distributed rendering as well as scalable rendering to aggregate the power of multiple GPUs for one or multiple views.

4.3 Bino

Bino is a stereoscopic movie player, using Equalizer to render 3D stereo movies, shown in Figure 1 (d). It supports large active and passive stereo display installations, including edge blending, bezel compensation, rotated displays and numerous stereo output modes, including active and passive stereo.

5 Conclusion

In this poster, we present a state-of-the art distributed parallel rendering framework, which is designed to be minimally invasive to facilitate the porting and development of real-world visualization applications. Equalizer is as generic as possible to support development of parallel rendering applications for different data types and rendering algorithms. Current advances in Equalizer have been mostly transparent to existing applications, that is, they can benefit from new fast asynchronous readbacks, subpixel compounds and automatic thread placement without any code modifications.

Acknowledgements

This work was supported in part by the Blue Brain Project, the Swiss National Science Foundation under Grant 200020-129525 and by the Spanish Ministry of Science and Innovation under grant (TIN2010-21289-C02-01/02) and the Cajal Blue Brain Project.

We would also like to thank github for providing an excellent infrastructure hosting the Equalizer project at http://github.com/Eyescale/Equalizer/.

References

[1] 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.

[2] 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.

[3] S. Eilemann, A. Bilgili, M. Abdellah, J. Hernando, M. Makhinya, R. Pajarola, and F. Schürmann. Parallel rendering on hybrid multi-gpu clusters. In *Proceedings Eurographics Symposium on Parallel Graphics and Visualization*, pages 109–117, 2012.

[4] S. Eilemann, M. Makhinya, and R. Pajarola. Equalizer: A scalable parallel rendering framework. *IEEE Transactions on Visualization and Computer Graphics*, May/June 2009.

[5] Eyescale Software GmbH. GPU Service Discovery Library. http://www.equalizergraphics.com/gpu-sd/, 2012.

[6] J. B. Hernando, L. Pastor, and F. Schürmann. Towards real-time visualization of detailed neural tissue models: view frustum culling for parallel rendering. In *BioVis 2012: 2nd IEEE Symposium on biological data visualization*, 2012.

[7] 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.

[8] 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.