OctaVis:
A Simple and Efficient Multi-View Rendering System

Eugen Dyck, Holger Schmidt, Mario Botsch

Computer Graphics & Geometry Processing
Bielefeld University

Abstract: We present a simple, low-cost, and high performance system for multi-view rendering in a virtual reality (VR) context. In contrast to complex CAVE installations, which are typically driven by one render client per view, we arrange eight displays in an octagon to provide a full 360° horizontal view, and drive these eight displays by a single PC equipped with multiple graphics units (GPUs). We describe the hardware and software setup, as well as the necessary optimizations to fully exploit the parallelism of this multi-GPU VR system.

Keywords: Multi-View Rendering, Multi-GPU Rendering, Virtual Reality

1 Introduction

Thanks to the steady increase in computational resources and rendering performance over the last decade, virtual reality (VR) techniques have developed into valuable tools for a large variety of applications, such as automotive design, architectural previews, game development, or medical applications, to name just a few. In all these applications a high level of immersion is both desired and required.

Our approach is motivated by a medical research project that aims at diagnosing and training stroke patients based on neuro-psychological tests performed in a virtual environment. In order to enable the transfer of the patient's training successes to real-life situations, the VR setup has to be sufficiently realistic and immersive.

CAVE installations are known to provide a very high level of immersion, but disqualify for our project because of their high cost and maintenance effort, which is mainly due to the fact that CAVEs are usually driven by one rendering node per view. We propose a cost-efficient visualization system that consists of eight standard (non-stereo) touch screen displays arranged in an octagon around the patient, thereby providing a full 360° horizontal view and a simple interaction with the scene. To minimize the stress on the patient we abandon stereo rendering. In order to reduce hardware costs and maintenance effort, our so-called OctaVis solution is driven by a single PC, which consequently is equipped with several graphics processing units (GPUs) (see Figure 1).

This paper describes our hardware and software architecture, parallelization efforts, and performance optimizations that eventually enable the real-time rendering of complex VR environments on eight views using a single PC.

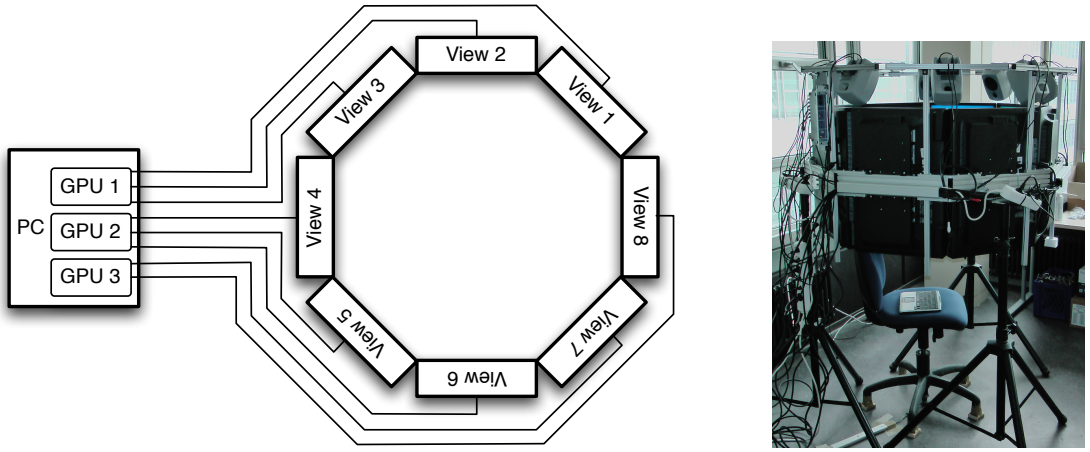

Figure 1: In our OctaVis setup a single PC is equipped with three GPUs that drive eight displays, which are arranged in an octagon to provide a full 360° visualization.

2 Related Work

The standard approach to drive an eight-display system is to use a render cluster consisting of one application server and eight render clients. This solution, however, is complex in terms of both hardware and software. First, it requires nine individual PCs that are properly synchronized and connected through a sufficiently fast network (Gigabit Ethernet, Infiniband). Second, the rendering has to be distributed to the render clients, by either employing a distributed scene graph (e.g., OpenSG [Ope10]) or by distributing OpenGL commands (e.g., Chromium [HHN+02]).

In contrast, we decided for a single-PC multi-GPU solution for driving our OctaVis system. Our workstation is a standard, off-the-shelf PC with four PCIe slots (Intel Core i7, Windows 7). We experimented with either four NVIDIA GeForce 9800 GX2 cards (two graphics outputs each) or three ATI Radeon HD 5770 cards (three outputs each). The question is how to design the rendering architecture, such that the parallel performance of this multi-GPU system is exploited as much as possible.

The major graphics vendors already provide solutions for combining several GPUs in order to increase rendering performance (NVIDIA's SLI, ATI's CrossFire). Note, however, that these techniques only support a single graphics output, and hence are not applicable in our multi-view setup. NVIDIA's QuadroPlex is a multi-GPU solution that combines up to four Quadro GPUs in an external case. Two QuadroPlex boxes can therefore be used to drive eight displays, but such a system comes at a price of more than $20,000. ATI's EyeFinity is a technology for driving several displays by one graphics board, and hence is rather a multi-view approach, but not a multi-GPU solution.

In our system we do not make use of any of these techniques, but rather handle each view and each GPU individually. This requires to distribute the rendering to the different GPUs in an as efficient as possible manner.

We experimented with higher-level APIs, such as the distributed scene graph OpenSG [Ope10] and the multi-GPU-aware scene graph OpenSceneGraph [OSG10]. However, these frameworks turned out not to be flexible enough to give (easy) control over the crucial details affecting multi-GPU performance. A distribution of OpenGL commands based on Chromium [HHN+02] was done by Rabe et al. [RFL07], who built a system similar to ours based on three NVIDIA cards. However, their performance results are rather disappointing, mainly due to the overhead induced by the Chromium layer. An elegant abstraction-layer for parallel rendering is provided by the Equalizer framework [EMP], which allows for distributed OpenGL rendering using render clusters, multi-GPU setups, or any combination thereof.

Our design goal was to keep the OctaVis system as simple as possible—both in terms of hardware and software—since this allows for easy maintenance and future extension. To this end, we do not employ any higher-level API for distributed or parallel rendering, but instead custom-tailor a (simple) low-level OpenGL solution for our target system. The resulting rendering architecture and required optimizations are described in the next section.

3 Rendering Architecture

Since we do not build our rendering architecture on top of a high-level framework, we have to (and are able to) take care of (1) the distribution of render commands to the different GPUs, (2) the data management, and (3) low-level OpenGL optimizations. In the following we discuss these three steps in detail.

3.1 Distributing OpenGL Commands

Our multi-GPU multi-view architecture consist of three GPUs with three outputs each, which drive the eight displays of our OctaVis system. We therefore have to be able to address OpenGL commands to a particular display attached to a particular GPU.

At application start-up, we generate a single full-screen window for each of the eight views, with each window having its own OpenGL context. For Unix operating systems distributing render commands is easy, since an individual X-server can be explicitly assigned to each view. In our medial research project, however, external constraints require Windows as an operating system, which does not provide this kind of explicit control.

The Windows Display Driver Model 1.1 used in Windows 7 provides improved support for multi-GPU applications, but it turned out that the actual performance strongly depends on the GPU driver. Our experiments revealed that the NVIDIA driver dispatches *all* OpenGL render commands to *all* available GPUs. This obviously prevents efficient parallelization, which is clearly reflected in the performance statistics in Section 4. In order to address a specific NVIDIA GPU the OpenGL extension `WGL_NV_gpu_affinity` has to be used, but this extension is only available for (expensive) Quadro-GPUs.

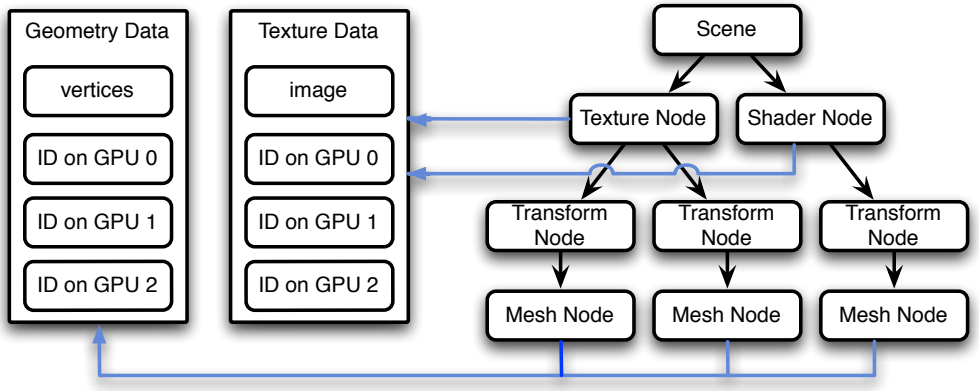

Figure 2: In our scene graph nodes share data in order to reduce memory consumption.

In contrast, the ATI driver dispatches render commands to just the one GPU responsible for the current window, which is the GPU attached to the display the window was created on. It therefore turned out to be crucial to create the eight windows at the correct initial position on the respective view. Creating all windows on the first view and moving them to the proper position afterward does not work. When taking this subtle information into account, the ATI driver allows for efficient parallelization between different GPUs.

3.2 Data Management

Thanks to our simple single-PC architecture we do not have to distribute scene data or render states over the network to individual render clients. However, in order to render the scene in our multi-view application each OpenGL context (i.e., each view) needs access to the scene data, such as, e.g., textures and shaders. In order to minimize memory consumption, we do not duplicate the scene data for each view, but instead store a copy for each GPU only, which is then shared by all views attached to this GPU.

This behavior can easily be implemented by shared OpenGL contexts. We incorporated this functionality into a very simplistic scene graph with standard nodes for transformations, textures, materials, and triangle meshes. Texture nodes, for instance, then store a reference to a texture object only. The texture object in turn stores the texture data on each available GPU (but *not* for each individual view). See Figure 2 for an illustration. In our case, where each GPU drives up to three views, memory consumption is reduced by a factor of three.

3.3 Performance Optimizations

In order to optimize rendering performance one has to identify and eliminate the typical performance bottlenecks: CPU load, data transfer from CPU to GPU, and GPU load. In a multi-view multi-GPU environment, even more attention has to be paid to these issues.

Rendering geometry in *immediate mode* quickly makes the application CPU-bound due to the massive amount of `glVertex()` function calls. We therefore store vertex positions and

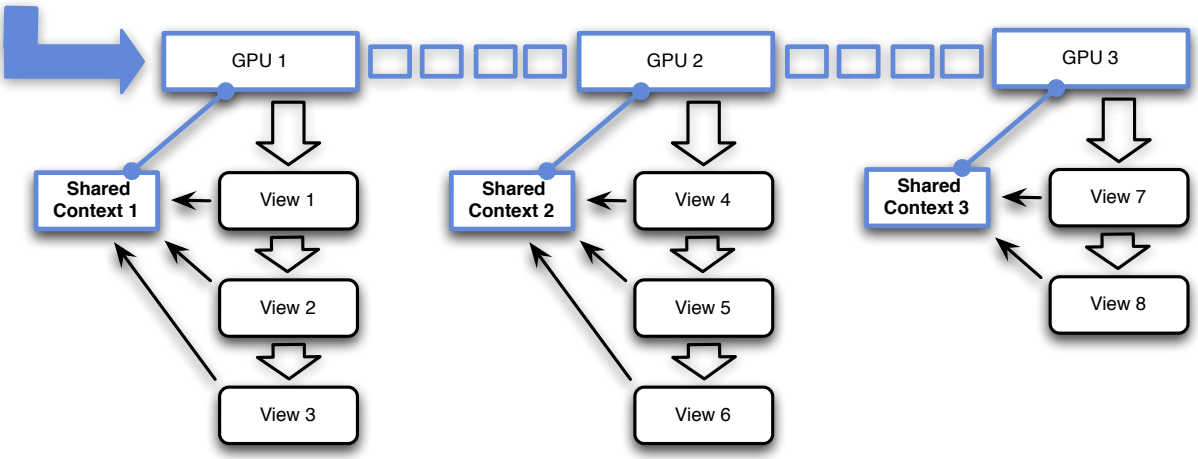

Figure 3: Rendering process: All views attached to the same GPU share a render context and are drawn consecutively, while the rendering is parallelized over the available GPUs.

triangle indices in *vertex arrays* (VA), which allows to render meshes with a single function call and thereby eliminates the CPU bottleneck. However, in our multi-view setup the data has to be transferred from main memory to the GPU eight times (for each individual view) in each frame, such that data transfer immediately becomes the bottleneck.

Data transfer can be eliminated by storing the vertex arrays in *vertex buffer objects* (VBO) on the GPU. This turned out to be absolutely crucial in our multi-view setting. The respective data storage follows the same shared context paradigm as described in the previous section. With VBOs the bottleneck is no longer data transfer, but the per-vertex computations of the GPU.

This GPU load can be reduced by caching computations performed for individual vertices. If a triangle is rendered and one of its vertices has been processed before and is still in the cache, these computations can be re-used. To maximize cache-hits we re-order the individual triangles of the mesh using the method described in [YL05], which (depending on the model) yields a significant performance gain.

Finally, in order to optimally exploit all available GPUs, the render traversal is parallelized: Each GPU is served by a dedicated render thread that processes all views attached to that GPU in a serial manner. Parallelizing over the (two or three) views on the same GPU did not increase performance. Figure 3 gives an overview of the rendering process.

4 Results

We analyze our rendering architecture on a standard PC with an Intel Core i7 CPU and running Windows 7. We experimented with two kinds of GPUs: four NVIDIA GeForce 9800 GX2 (two outputs each) and three ATI Radeon HD 5770 (three outputs each). For each card we used the latest driver (NVIDIA: 257.21, ATI: Catalyst 10.6).

# GPUs	# Views/GPU	# Views	FPS NVIDIA	FPS ATI
1	1	1	58.3	273.6
2	1	2	27.5	253.8
3	1	3	19.3	258.8
1	2	2	30	129.7
2	2	4	14.6	121.5
3	2	6	9.8	121.5
1	3	3	–	82.4
2	3	6	–	79.4
3	3	9	–	78.1

Table 1: Comparing frame rates for a 870k triangle model using NVIDIA GeForce 9800 GX2 and ATI Radeon HD 5770 cards, respectively, in several multi-GPU and multi-view setups.

# GPUs	# Views/GPU	# Views	FPS (VA)	FPS (VBO)	FPS (reorder)
1	1	1	38.8	214.3	344.2
2	1	2	22.3	214.6	344.8
3	1	3	14.2	214.7	344.7
1	2	2	19.2	107.7	171.3
2	2	4	10.2	107.5	171.2
3	2	6	7.0	107.5	171.1
1	3	3	12.7	71.6	113.6
2	3	6	7.2	71.4	113.5
3	3	9	4.7	71.3	113.3

Table 2: Frame rates for different optimizations (vertex arrays, vertex buffer objects, cache-friendly reordering), using ATI Radeon HD 5770 GPUs.

We first compare the performance scaling of NVIDIA and ATI cards/drivers with varying numbers of GPUs and varying numbers of views per GPU (see Table 1). For this experiment, a model of 870k triangles was rendered. We employed VBOs, such that the GPU performance (and not the data transfer) is measured. The performance difference for one GPU and one view is simply due to the fact that the two cards represent different generations of GPUs.

It is clearly visible that the NVIDIA system scales inversely proportional to the number of views: No matter whether two views are driven by one GPU or by two GPUs, the performance drops by a factor of about two. This is an immediate consequence of the NVIDIA driver sending OpenGL commands to all available GPUs, as also described in [EMP]. In contrast, the ATI system scales almost perfectly. Rendering one view per GPU gives the same performance on one, two, or three cards. When keeping the number of GPUs fixed, the performance decreases almost linearly with the number of attached views per GPU. Hence, the ATI system can fully exploit the parallelization between multiple GPUs.

Figure 4: The eight views of the VR supermarket used for the timings in Table 2.

Using the ATI system, we now compare the different optimization techniques discussed in Section 3.3, as listed in Table 2. For these experiments we visualize the VR scene to be used in the medical project, which is a virtual supermarket of about 1.7M triangles shown in Figure 4. When the geometry is stored in vertex arrays (VA), but not in VBOs, the geometry data is transferred to the GPU for each active view. Consequently, the performance drops with each additional view, even when they are attached to different GPUs. Storing the geometry in VBOs eliminates the transfer costs, which then yields the convincing scaling behavior discussed above. The last column shows the performance after reordering the triangles in order to maximize cache usage [YL05]. With this last optimization, our rendering architecture is able to visualize a 1.7M triangle scene on nine display at a performance of more than 100 frames per second.

As mentioned above, we initially experimented with the two popular Open Source scene graph libraries OpenSceneGraph [OSG10] and OpenSG [Ope10], but did not achieve a comparable performance in terms of rendering speed and memory consumption:

- Since OpenSG is specialized to distributed rendering, our implementation uses eight local render server processes to drive the eight displays. Since each render server requires a full scene copy, this consumes significantly more memory than our shared contexts and therefore does not allow for highly complex scenes. Moreover, due to suboptimal window creation and setup (see Section 3.1) OpenGL commands are dispatched to all GPUs, which slows down the rendering to just about 10 frames per second (for a less complex supermarket model).

- OpenSceneGraph correctly creates and initializes windows, supports for shared OpenGL contexts, and allows for precise control of VBOs. However, its rendering performance is still only about 70% of ours, which we assume is due to the higher overall complexity of this scene graph system.

Due to our use of touch screen displays we could not employ seamless displays, resulting in clearly visible seams between the eight screens. In an empirical study [PK10] conducted by the Department of Psychology at Bielefeld University our OctaVis setup was evaluated by 27 subjects. Particularly asked about the effect of these seams, 21 subjects stated them to be not disturbing at all, 3 were disturbed slightly, and 3 were confused by them.

5 Conclusions

These experiments clearly demonstrate the potential of a multi-view rendering system based on a single PC equipped with multiple GPUs. However, the results also indicate that the multi-GPU performance can crucially depend on a few seemingly minor implementation details, on optimization techniques, and on GPU drivers and operating systems. This paper tries to provide a recipe to circumvent pitfalls and achieve a high performance single-PC multi-GPU system. In terms of hardware, our proposed OctaVis system is cheap and easy to maintain. In terms of software, the rendering architecture scales almost perfectly thanks to few but carefully done low-level optimizations.

So far we did not implement higher-level scene graph optimizations, such as different types of culling, spatial data structures, sorting with respect to state changes, or other well-known techniques. These are clearly the next steps and are expected to yield a similar performance gain compared to single-GPU solutions.

Acknowledgments: The authors are grateful to Christian Fröhlich and Bernhard Brüning for their support with the initial OpenSG-based version of the rendering system. Mario Botsch is supported by the Deutsche Forschungsgemeinschaft (Center of Excellence in "Cognitive Interaction Technology", CITEC). Eugen Dyck and Holger Schmidt are supported by the EFRE project "CITmed: Cognitive Interaction Technologies for Medical Applications".

EUROPÄISCHE UNION
Investition in unsere Zukunft
Europäischer Fonds
für regionale Entwicklung

References

[EMP] S. Eilemann, M. Makhinya, and R. Pajarola. Equalizer: A scalable parallel rendering framework. *IEEE Trans. on Visualization and Computer Graphics*, 15(3):436–452.

[HHN+02] 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 (SIGGRAPH)*, 21(3):693–702, 2002.

[Ope10] OpenSG. `http://www.opensg.org`, 2010.

[OSG10] OpenSceneGraph. `http://www.openscenegraph.com`, 2010.

[PK10] Martina Piefke and Sina Kühnel. Empirie- und Beobachtungspraktikum: Physiologische Psychologie, Departement of Psychology, Bielefeld University. Winter Term 2009/2010.

[RFL07] F. Rabe, C. Fröhlich, and M. E. Latoschik. Low-cost image generation for immersive multi-screen environments. In *Workshop of the GI VR & AR special interest group*, pages 65–76, 2007.

[YL05] S. E. Yoon and P. Lindstrom. Cache-oblivious mesh layouts. *ACM Transactions on Graphics (SIGGRAPH)*, 24(3):886–893, 2005.