Multi-Frame Rate Rendering for Standalone Graphics Systems

Jan P. Springer Stephan Beck Felix Weiszig Bernd Fröhlich

Bauhaus-Universität Weimar

Abstract: Multi-frame rate rendering is a technique for improving interaction fidelity in complex virtual environments. The technique renders the interactive elements of a scene on a separate graphics processor and composes it with the remainder of the scene using optical superposition or digital composition. Multi-frame rate rendering is naturally implemented on a graphics cluster. With the recent availability of multiple graphics cards in standalone systems multi-frame rate rendering can also be implemented on a single PC where memory bandwidth is much higher compared to off-the-shelf networking technology. This decreases overall latency and further improves interactivity. We report on our experiences of implementing multi-frame rate rendering on such a multi-graphics card system. In addition we have implemented multi-frame rate rendering on a single graphics processor by interleaving the rendering streams for the interactive elements and the rest of the scene. This approach enables the use of multi-frame rate rendering on low-end graphics systems such as laptops, mobile phones, and PDAs.

Keywords: Multi-Frame Rate Rendering, 3D Interaction, Multi-GPU Hardware

1 Introduction

In the past years the capabilities of graphics processing units (GPU) have been dramatically improved but the expectations on *visual quality* have increased even more. In general this affects the interactivity of applications, or *interaction quality*, because this interaction quality is traded for visual quality. The interactive and high-quality visualization of large models therefore is still a challenging problem. Visual quality mostly depends on scene complexity (e. g. the number of primitives), rendering method, illumination and shading model, as well as display resolution. All of these factors might also improve the interaction fidelity of an application but often lead to low frame rates when, at the same time, excellent visual quality is desired. Interaction quality highly depends on the immediate incorporation of user actions into the simulation and image generation process demanding high frame rates.

In [SBW+07] we introduced multi-frame rate rendering and display. This approach uses multiple image generators to render interactive parts of a scene, e. g. menus, cursor, as well as scene objects currently manipulated by the user, with the highest possible frame rates while the rest of the scene is rendered at regular frame rates. The results from individual asynchronously running image generators are optically or digitally combined into a multi-frame rate image. While multi-frame rate rendering introduces certain artifacts we were able to show in a user study that digitally composed multi-frame rate images performed nearly as good as rendering everything fast in a task-oriented setup. The potential of this method is such that it provides computational resources for high quality

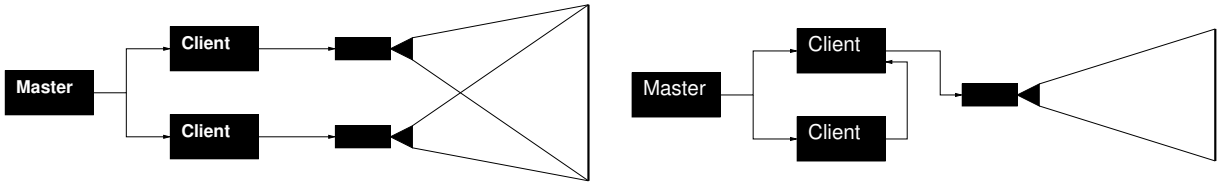
visualization techniques to be available while the user is still able to properly interact with the application.

Multi-frame rate rendering, due to its non-synchronization property, is naturally implemented on a cluster of graphics workstations. Still software capable of handling such a setup above multi-display configurations is not yet commonly found. With the introduction of the PCIe bus system for the PC hardware market multiple GPUs in one single PC are available for the first time. Such a multi-GPU system can be used to implement multi-frame rate rendering by combining the responsibilities from distributed graphics nodes, as described in [SBW+07], into one single PC. This avoids network bandwidth limitations and decreases latency, which in turn increases interaction fidelity for the user. Additionally certain artifacts, caused by the asynchronous image generation process that exhibits different frame rates as well as bandwidth limitations at the network transfer layer, can be controlled more precisely. First, memory and bus-transfer bandwidth on current PC systems is much higher than on today's off-the-shelf network setups, e. g. Gigabit Ethernet. Second, while the several graphics cards generate images at the same time communication about frame state and buffer transfer can be expressed as a local resource, i. e. shared variables in process shared memory. This avoids sending and synchronizing events between cluster nodes. Effectively end-to-end latency between an interaction event and its visual feedback is much lower in a multi-GPU setup than in a graphics cluster.

Another approach is the use of multi-frame rate rendering on a single GPU. Basically a scheduling mechanism is used that renders interactive scene parts at the desired frame rate while successively rendering the rest of the scene in segments of smaller pieces in the time left between the rendering of the interactive parts and the actual frame buffer swap. With this mechanism, while not as scalable as a multi-GPU setup or a cluster of graphics nodes even users of low-end graphics systems can benefit from the improved interaction fidelity of our technique.

2 Related Work

[SBW+07] describes multi-frame rate rendering and display. A cluster of graphics nodes is used to implement either optical superposition or digital composition of images created by asynchronously running image generators, cf. figure 1 for schematic drawings of the respective setups. Multi-frame rate rendering uses a distributed setup consisting of a master node, a slow client (*SC*), and a fast



(a) Optical superposition. **(b)** Digital composition.

Figure 1: Multi-frame rate rendering methods. (a) Optical superposition of two projectors creating an optical (output) buffer and (b) digital composition of color buffer and depth buffer from two render nodes creating a digital (output) buffer.

client (*FC*). The *SC* is designated for rendering all parts of the scene the user is not currently interacting with. The *FC* is responsible for rendering only parts of the scene that are currently relevant to the user interaction, e. g. selected object(s), menus, cursors, etc. The scene as a whole is distributed by the master node to the render clients. Upon starting an interaction with an object this object is excluded from rendering at the *SC* and included in the render process of the *FC*. The amount of objects that must be rendered by the *FC* is much lower than on the *SC* which leads to higher frame rates on the *FC* and therefore for better interaction responses to the user while the *SC* may render at low even non-interactive frame rates. When the user ends the interaction the selected object is excluded from *FC*'s render process and included again into the render process of the *SC*. Multi-frame rate rendering by optical superposition combines the output of *SC* and *FC* by optically super-positioning the respective images using light projectors as shown in figure 1a. Multi-frame rate rendering by digital composition incorporates the frame buffer image of one frame (color and depth values) from *SC* into the render process of *FC* as shown in figure 1b. This avoids half-transparency artifacts of the optical method due to the image accumulation in the optical path. Visual artifacts may appear because of the different frame rates of *SC* and *FC*. However a user study comparing task performance times between rendering everything slow (10 Hz), multi-frame rate rendering by digital composition (10/30 Hz), and rendering everything fast (30 Hz) showed that the digital composition method performed as as good as rendering everything fast. Multi-frame rate rendering does not enhance navigation performance though. To ensure consistency of the multiple images view transform changes should be coupled with the frame rate of the *SC*.

Hardware support for multi-GPU setups exists in the form of NVIDIA SLI [nGPG05, chap. 8]. Two graphics cards are interconnected with a bridge that allows the transfer of post-frame buffer image data from one card to the other without host system intervention. The graphics cards can be configured to render alternating frames (alternate frame rendering, AFR) or to load balance the amount of pixels to be rendered by each graphics device (split frame rendering, SFR). Enhanced image quality can be achieved by using both cards to generate samples for the same image which are then combined in a final full screen anti-aliasing pass on the primary card. NVIDIA SLI is targeted at the end user who can configure the system with the methods described above globally or on a per application basis. However because the method(s) work in screen space they may not exhibit performance improvements for applications that are not fill limited. In contrast multi-frame rate rendering on multi-GPU systems, while also utilizing more than one graphics device, effectively enable the application developer to decide which parts of the scene are to be prioritized, e. g. based on user interaction, which allows for performance improvements in a variety of application scenarios.

3 Multi-GPU System Support

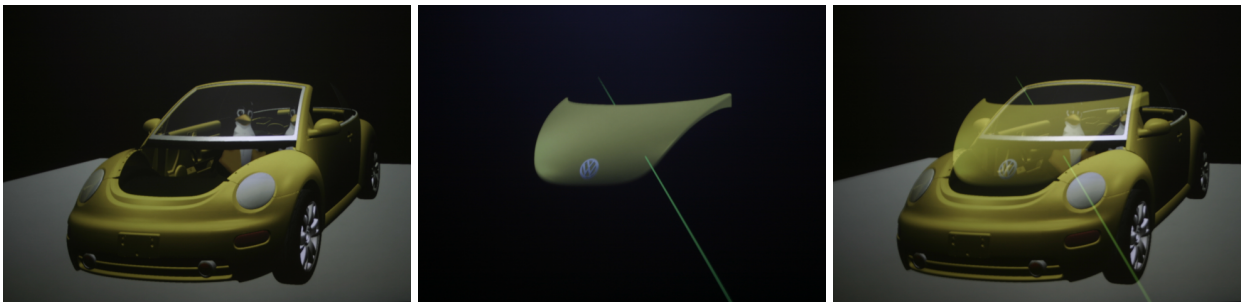
Multi-frame rate rendering intentionally builds upon the idea of several graphics systems running asynchronously on separate computers but participating in the creation of a single coherent image. However, multi-computer or cluster support is not yet a common feature for VR systems or graphics

software APIs at large. Also, the infrastructure to be build before deployment as well as maintenance efforts are considerable. In such cases a single computer system solution is preferable. With the introduction of the PCIe bus system for PCs a high-speed interconnect is available that allows for several graphics subsystem to be used within a single PC. With a system based on such hardware a single-computer solution for multi-frame rate rendering can be build.

3.1 Optical Superposition

Multi-frame rate rendering by optical blending on a single system can be achieved in much the same way as in a graphics cluster. The two image generators are assigned to the roles of *FC* and *SC*, respectively. The outputs of the image generators is connected to light projectors whose projection is completely overlapped on the same screen. If the part of the distributed application responsible for sensor data processing and event propagation is also running on that machine further network latency can be avoided. This enhances interaction fidelity. Visual artifacts inherent to this method will remain the same as discussed in [SBW+07].

Figure 2 shows digital photographs of an application prototype using multi-frame rate rendering by optical superposition on a single computer system with multiple image generators. Figure 2a shows the part rendered by *SC* while figure 2b shows the part rendered by *FC* only which consists of the interaction representation, a simple ray and the object the user is interacting with. Figure 2c finally shows the optical superposition of figures 2a and 2b as perceived by the user.



(a) Slow part on GPU 1. **(b)** Fast part on GPU 2. **(c)** Optical superposition.

Figure 2: Multi-frame rate rendering by optical superposition on a single computer system using multiple image generators. (a) Scene part rendered by *SC* on GPU 1. (b) Scene part rendered by *FC* on GPU 2. (c) Optical superposition of (a) and (b) creating the final image as perceived by the user.

3.2 Digital Composition

Multi-frame rate rendering by digital composition will benefit more from a single system solution than optical blending because it is possible to substitute external network bandwidth with host memory transfer bandwidth. Thereby latency can be reduced. On the computer system two image generators are installed and configured as two independent graphics devices. One of the image generators is used as the *FC* and the other image generator as the *SC*. Every time the *SC* finishes a frame the content of its frame buffer image (color and depth values) is read back into host memory

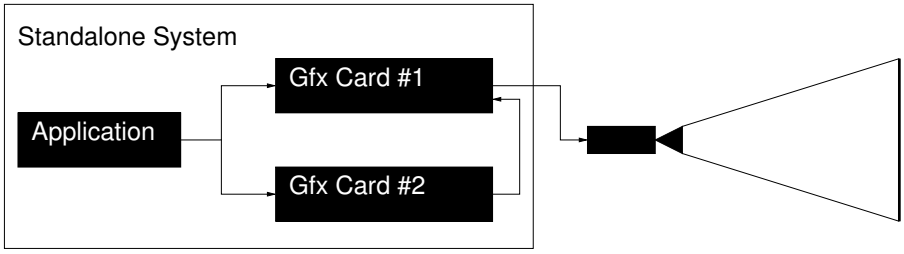
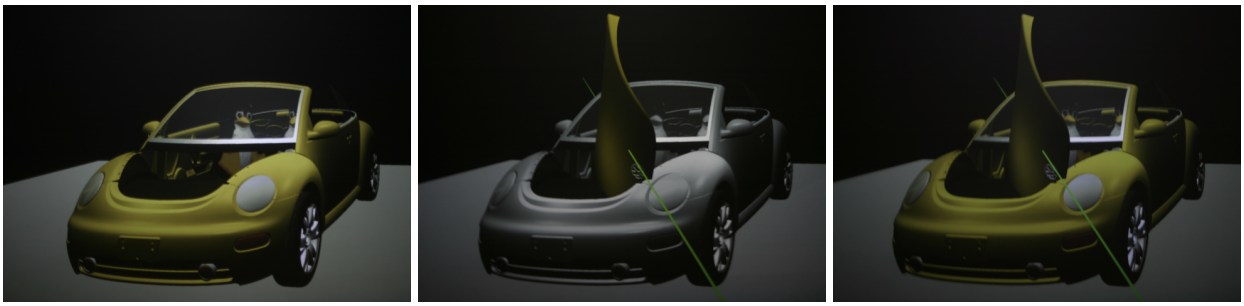


Figure 3: Multi-frame rate rendering by digital composition on a single computer system using multiple image generators. FC and SC are assigned to separate image generators; frame buffer images (color and depth values) are transferred from SC to FC via host memory.

and used as the base for digital composition on the FC, cf. figure 3 for a schematic presentation. Visual artifacts inherent to multi-frame rate rendering by digital composition will remain the same as discussed in [SBW+07].

Figure 4 shows digital photographs of an application prototype using multi-frame rate rendering by digital composition on a single computer system with multiple image generators. Figure 4a shows the part rendered by SC. Figure 4b shows the frame buffer image transferred from SC in gray scale and the part rendered by FC colored. Figure 4c finally shows the digital composition as perceived by the user.



(a) Slow part on GPU 1. **(b)** Fast part on GPU 2. **(c)** Digital composition.

Figure 4: Multi-frame rate rendering by digital composition on a single computer system using multiple image generators. (a) Scene part rendered by SC on GPU 1. (b) Scene part rendered by FC on GPU 2, frame buffer image from SC in gray scale. (c) Final image output on GPU 2 as perceived by the user.

3.3 Comparison

Using a multi-GPU system for multi-frame rate rendering provides several advantages compared to the original cluster solution presented in [SBW+07]. Latency in this setup decreases because the frame buffer image is transferred to and from host memory only within a single computer. Figure 5 depicts the end-to-end latency for multi-frame rate rendering using digital composition in a cluster setup. Given a frame rate ratio of 10/30 Hz for SC/FC the following can be observed. Both SC and FC receive sensor data and event updates with a latency of ≈ 40 ms assuming an externally running tracking system sending new values at a rate of 60 Hz. On FC the frame time is split into the time needed for drawing the frame buffer image from SC (TDraw in figure 5) and the time for

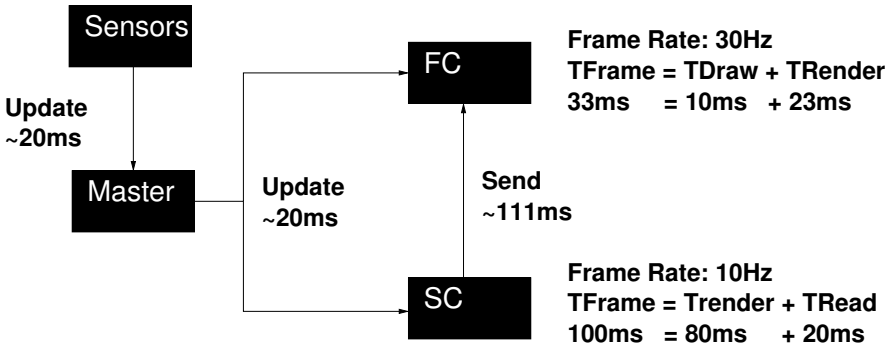


Figure 5: End-to-end latency for multi-frame rate rendering using digital composition in a cluster setup.

rendering the relevant scene parts for the current interaction (TRender in figure 5). TDraw is the accumulated time to draw color and depth values at a certain resolution. Assuming 1280×1024 pixels for the image size table 1 shows this to be ≤ 10 ms. This amounts to an end-to-end latency for *FC* of 73 ms in case the event updates arrive at frame start or 106 ms if they arrive at frame end. On *SC* event updates also arrive with a latency of ≈ 40 ms. The frame time here consists of the time needed for rendering its relevant scene parts (TRender in figure 5) and the time to read back the frame buffer image (TRead in figure 5). TRead, like TDraw, is the accumulated time for reading color and depth data from the graphics device as shown in the lower part of table 1. We assume TRead to be ≤ 20 ms. Additionally the just read frame buffer image must be send over the network to *FC*. For a resolution of 1280×1024 this takes ≈ 111 ms as can be seen in table 2. The final end-to-end latency for incorporating the frame buffer image from *SC* into *FC* is then 251 ms or 284 ms at most if frame buffer image updates from *SC* must wait for frame finish on *FC*.

		1280×1024		1600×1200	
		MB/s	ms	MB/s	ms
Read	RGBA	521	10.1	684	11.6
	BGRA_EXT	997	5.3	939	8.4
	DEPTH	565	9.3	733	10.8
Draw	RGBA	1298	4.0	1412	5.6
	BGRA_EXT	2081	2.5	2166	3.6
	DEPTH	1213	4.3	1372	5.7

Table 1: Timings for reading from and writing to the graphics card (NVIDIA GeForce 8800 GTX, driver rev. 97.46, ASUS P5N32-E SLI based system). Note that reading/writing performance of color data heavily depends on the "native" hardware format.

When compared the bandwidth that can be achieved on current graphics systems and network setups differs by at least one order of magnitude. Gigabit Ethernet provides an approximate maximal bandwidth of 90 MB/s in practical experience for application-level usage. Thus sending a buffer of 10 MB, i. e. $1280 \times 1024 \times 64$ bits, takes ≈ 111 ms while reading and writing the same buffer from and to graphics hardware only takes 20 ms and 10 ms, respectively. In a stereoscopic setup, where

Resolution 64 Bit Color/Depth	Buffer Size	Transfer Time	Transfer Rate
1024×768	6 MB	66 ms	15 Hz
1280×1024	10 MB	111 ms	9 Hz
1600×1200	15 MB	166 ms	6 Hz

Table 2: Network transfer times at different buffer sizes for Gigabit Ethernet (observed for application level end-to-end buffer send and receive on a Cisco Catalyst 3560G switch).

two frame buffers must be send per frame, these times would double. Clearly network transfer is the limiting factor here. A multi-GPU system taking the roles of *SC* and *FC* reduces the end-to-end latency of frame buffer image updates from *SC* to *FC* from 251 ms to only 100 ms to 133 ms, or in the stereo case from 362 ms to 130/163 ms, because only reading and writing of image data on the same host memory is necessary. When also including the sensor and event update processes into the same machine latency can be reduced even further.

Another effect is the simplification of the underlying software infrastructure. While multi-frame rate rendering in a cluster setup needs an existing infrastructure for distributing the scene content as well as scene updates this is not necessary in a multi-GPU context. A standalone rendering API is all that is needed. Changes in the scene can be communicated by using local variables because the scene management takes place in a single process. If the hardware platform exhibits multiple processors the use of threads may also improve performance considerably while the communication and synchronization costs are still those for a single process and address space context.

There are certain disadvantages to multi-GPU multi-frame rate rendering though. While in a cluster setup it is possible to add more render nodes to the *SC*, for example to ameliorate the performance hit for rendering two views for a stereoscopic view or to setup a Sort-Last network, this is not possible in a single-system multi-GPU setup. Also, running an application that performs potentially computational expensive operations may have an adverse effect on the local graphics sub-system.

4 Single-GPU System Support

Implementing multi-frame rate rendering on a single PC with multiple GPUs exhibits certain performance improvements while avoiding the usage of several clustered machines. We also experimented with a single-GPU single-machine setup as it is available to most computer users today. The basic idea is to use the same GPU for the *FC* as well as the *SC* role for multi-frame rate rendering by interleaving the rendering streams of the *FC* and *SC* part. Interleaved rendering thus allows even portable computer users to benefit from our technique.

Interleaved rendering uses the time left before buffer swap of the *FC* part to render a certain amount of geometry from the *SC* part. A simple example may clarify this in other terms: suppose an image generator can render 500,000 polygons at 60 Hz. Theoretically this means it can also render 100,000 polygons at 60 Hz and 1,600,000 polygons at 15 Hz. This idea stems from our observation

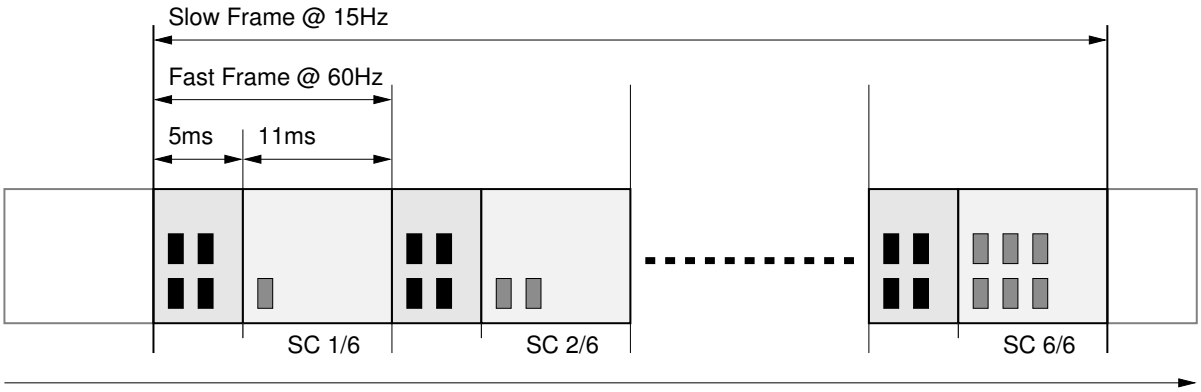
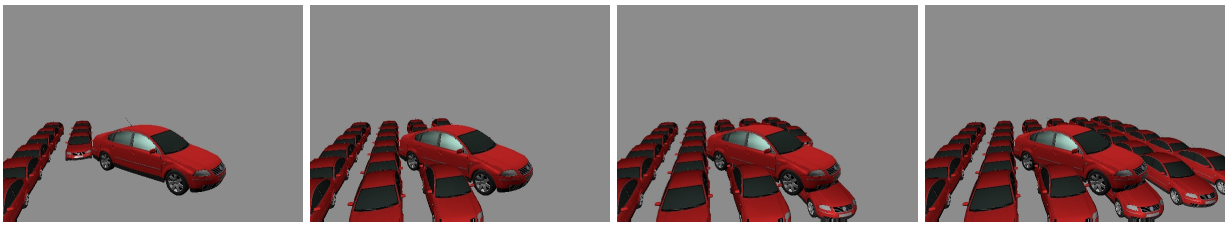


Figure 6: Multi-frame rate rendering by digital composition on a single computer system using a single image generator.

that the *FC* part in a multi-frame rate rendering setup is usually faster than the actual time needed to sync with the output display. In the following we must distinguish between frame rate and sync rate. Frame rate is the frequency with which frame buffer updates are available. Sync rate is the rate with which frame buffer content is "scanned-out" to the output device. While the sync rate is constant (e. g. 60 Hz or 75 Hz) the frame rate usually is not, i. e. for a desired frame rate of 60 Hz the actual frame time might be somewhere in-between 10 ms and 16.6 ms. In a clustered setup or a multi-GPU configuration this time can be used to update frame buffer data from the *SC*. In a single-GPU setup this leftover time until buffer swap will be used to render several "chunks" of geometry from the *SC* part, cf. figure 6. Rendering of arbitrary sized spatial parts of a scene is not a trivial problem to be solved in general. For our tests we used a simple mechanism to split up the scene in chunks of geometry with an equal amount of triangles. Chunk size was chosen so that its graphics processing time was small enough to create batches of several such chunks that would fill the remaining frame time. A special render context is used for the *SC* which is a double-buffered frame buffer object (front-FBO and back-FBO, similar to double-buffered rendering) to reuse the results from earlier passes. *FC* always reads from *SC*'s front-FBO in order to fill-in from *SC*'s buffer. *SC* always renders the parts assigned from a scheduler into its back-FBO. Once *SC* is finished with all of the assigned parts its front-FBO and back-FBO are swapped, i. e. just a pointer swap and therefore inexpensive. The now new front-FBO is used as input for the *FC* while *SC* starts with the next frame cycle rendering to its newly assigned back-FBO until all relevant parts of the scene are finished again. Figure 7 shows screen shots from an interleaved rendering application prototype. The intermediate results from *SC*'s frame completion at 25%, 50%, 75%, and 100% are used for emphasis here in the digital composition process.

This approach has some limitations. The main limitation is that *FC*'s draw time directly affects the draw time of SC. This results in a theoretically infinite draw time for *SC* if there is no time left between the end of *FC*'s frame and display sync. Multi-frame rate rendering by optical superposition is not possible because only one output image is generated. Successive rendering of parts of the scene requires a spatial segmentation of the scene similar to out-of-core algorithms, e. g. [IG03, Lin03]. Most of these algorithms, while solving the segmentation problem, may exhibit an excessive computation time; pre-processing is still state of the art for out-of-core rendering (e. g.



(a) SC at 25%. **(b)** SC at 50%. **(c)** SC at 75%. **(d)** SC at 100%.

Figure 7: Screen shots of a multi-frame rate application in interleaved rendering mode on a single GPU where the array of cars is rendered by the *SC* context only. (a) to (d) show the completion of *SC*'s frame content at 25%, 50%, 75%, and 100%, respectively.

the `OOCProxyBuilder` tool included in OpenSG 1.8 [nWa]). In a single-GPU multi-frame rate rendering setup re-computation of the spatial segmentation would be needed since it is not possible to determine chunk sizes a priori that correlate well with the current graphics hardware and graphics features used.

5 Conclusions and Future Work

Multi-frame rate rendering for standalone computer systems is a valuable complement to graphics cluster-based solutions. Using multiple GPUs within a single machine considerably decreases latency in comparison to multi-frame rate rendering on a cluster system. On the other hand, in a distributed setup further rendering nodes can be easily added whereas a standalone multi-GPU system is currently limited to only two or three graphics subsystems. In the near future this limitation may be lifted to a certain degree using vendor specific hardware, e. g. NVIDIA QuadroPlex [nWb].

Multi-frame rate rendering by digital composition on a multi-GPU system could be further improved by allowing frame buffer image transfer between the graphics cards without host system intervention. Frame buffer objects need to be tagged as a shared resource which should be visible on all GPUs. Changes to such an FBO on one GPU would trigger an automatic update of its contents to the other GPU(s) without transfer through host system memory. This would reduce latency even further because reading and writing of frame buffer images to and from host memory can be completely avoided. However such an extension is still not officially announced, but would be very beneficial to our approach.

Interleaved rendering tries to keep the graphics pipeline filled at the expense of a partitioned scene representation. It can be also considered as a multi-frame rate method, since its interactive objects and the rest of the scene are rendered at different frame rates and merged afterwards. It enhances the interactivity of otherwise non-interactive standalone applications considerably. Partitioning the scene could be avoided if a task on the GPU could be interrupted and continued at a later time. Much like process scheduling in operating systems this would enable the processing of high-frequency interaction visualization while high-quality visualization can be updated over several successive frames. With this method even users of single-GPU systems can benefit from the improved interaction fidelity of our technique.

Acknowledgments

We thank the NVIDIA Professional Solutions Group Europe for continued support and hardware donations.

References

[IG03] M. Isenburg and S. Gumhold. Out-of-Core Compression for Gigantic Polygon Meshes. *Trans. on Graphics*, 22(3):935–942, 2003.

[Lin03] P. Lindstrom. Out-of-Core Construction and Visualization of Multiresolution Surfaces. In *SI3D '03: Proceedings of the 2003 Symposium on Interactive 3D Graphics*, pages 93–102. ACM, 2003.

[nGPG05] NVIDIA GPU Programming Guide. NVIDIA, 2005. revision 2.4.0.

[nWa] OpenSG Website. http://opensg.vrsource.org/.

[nWb] NVIDIA Quadro Plex Website. http://www.nvidia.com/page/quadroplex.html.

[SBW+07] J. P. Springer, S. Beck, F. Weiszig, D. Reiners, and B. Froehlich. Multi-Frame Rate Rendering and Display. In *Proceedings IEEE Virtual Reality 2007 Conference*, pages 195–202. IEEE Computer Society, 2007.