A Physically-Based Client-Server Rendering Solution For Mobile Devices

Matt Aranha
Computer Science Department
University of Bristol
Bristol, England
aranha@cs.bristol.ac.uk

Piotr Dubla
Warwick Digital Laboratory
University of Warwick
Coventry, England
P.B.Dubla@warwick.ac.uk

Kurt Debattista
Warwick Digital Laboratory
University of Warwick
Coventry, England
K.Debattista@warwick.ac.uk

Thomas Bashford-Rogers
Warwick Digital Laboratory
University of Warwick
Coventry, England
T.E.W.Bashford-Rogers@warwick.ac.uk

Alan Chalmers
Warwick Digital Laboratory
University of Warwick
Coventry, England
A.G.Chalmers@warwick.ac.uk

ABSTRACT

Mobile devices, also known as small-form-factor (SFF) devices such as mobile phones, PDAs and ultra mobile PCs have continued to grow in popularity. Improvements in SFF hardware has enabled a range of suitable applications such as gaming, interactive visualisation and mobile mapping. Although high-fidelity graphic systems typically have significant computational requirements, the time taken may be largely resolution dependent. The limited resolution of SFFs indicates such platforms are prime candidates for running high-fidelity graphics.

Due to the limited hardware available on mobile devices, it is not currently possible to produce high-fidelity graphics in reasonable time. However, most SFFs have some degree of network capability. Using a remote server in conjunction with a mobile device to render high-fidelity graphics on demand allows us to substantially reduce the total rendering time. This paper introduces a client-server framework for minimising rendering times using a cost function to predict optimal distribution of rendering.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/Server*; D.2.8 [**Software Engineering**]: Metrics—*Performance Measures*

Keywords

Mobile Devices, Distributed Rendering, Ray tracing

MUM'07, December 12-14, 2007, Oulu, Finland.

1. INTRODUCTION

Over the past few years significant growth in the popularity of small-form-factor (SFF) devices such as PDAs and mobile phones has taken place. The portability of these devices, enables delivery of key visual information to users *"in the field"*. High-end applications including interactive exhibit exploration, navigational tools and multi-user mobile games are highly suited to exploiting mobile technology. Such applications are however significantly constrained due to the physical and technical properties of these devices. It is not currently possible to produce physically based high-fidelity 3D graphics at interactive frame rates on such platforms.

High-fidelity graphics are a set of rendering algorithms derived from the physical interaction of light within an environment. As a result of their physically realistic basis, such methods are capable of accurately calculating physical measures and hence produce highly realistic imaging. The drawback of these techniques is the inherent cost of performing such complex calculations. Recent advances in this field have however made it possible to render non-complex scenes at interactive rates on a single high performance desktop PC [29].

The ultimate goal of our work is to achieve interactive frame rates for high-fidelity rendering techniques on SFF devices. Although, it may not be possible to achieve these rates on current mobile platforms, we demonstrate techniques that minimise the amount of computation required as well as introduce novel methods of optimising the necessary rendering times. It is also worthwhile taking into account that the next generation of SFF devices are likely to resolve some of the resource limitations such as bandwidth, memory and processing power. From the combination of improvements in hardware as well as our specialised rendering techniques, we intend to demonstrate that physically-based techniques will shortly provide a realistic alternative over existing rendering strategies, delivering high quality synthesis at a reasonable cost.

Although the hardware of SFFs is limited, we believe that some of their characteristics work to the advantage of physically-based rendering techniques. Ray tracing based solutions scale linearly with resolution with little impact from scene complexity. In addition, due to their relatively limited screen size, using ray tracing would require less pixels to be rendered than a standard desktop machine.

Another important method to improve rendering output is to make best use of available resources. The majority of mobile devices are equipped with some degree of networking capabilities and it is increasingly common to make use of networks to stream content on demand to such devices. It is clear that by exploiting this access to additional processing capability, a huge step towards reducing rendering times can be taken. An obvious question arises from proposing such a solution: why should we bother to render it at all? Why would we not pre-render and stream the entire content? In certain cases such a solution would be the ideal. In the majority of scenarios however, a highly customised render, tailored to the user's perspective would be more suitable. To pre-compute all possible combinations would be unreasonable.

Within high-fidelity graphics systems, it is common to produce greater detail than it is physically possible to perceive. This can be a result of several factors including scene complexity, visual acuity and attention. The cost of producing this additional detail is not always insignificant, so by minimising the required computation it is possible to maintain image fidelity at a reduced cost.

This paper is divided as follows. Section 2 presents related work in the fields of high-fidelity rendering and SFF devices. Section 3 provides details of our implementation, with the results and conclusions in Section 4 and 5 respectively.

2. BACKGROUND

In this section we discuss related work on mobile devices as well as high-fidelity rendering techniques and related optimisations.

Figure 1. The GP2X Linux-based handheld game console with ray traced scene

2.1 Mobile Devices

The increase in popularity of SFF devices as well as recent advances in computer graphics has produced a significant market for high-end graphics performance on mobile devices. The ability to generate such imagery on these devices is however severely limited by the available processing power, battery power and bandwidth. In order to overcome these constraints novel algorithms to optimise usage have been developed [18, 17, 13]. Computationally intensive applications, especially high-fidelity graphics are suitable candidates for optimisation.

Current top end graphics make use of rasterising methods on dedicated graphics cards [19]. Despite recent hardware advances, top end mobile graphics are still relatively lightweight compared to desktop PCs. As well as the limited processing power, a typical high end SFF device has a display size of 3.5" and 320 $\times$ 240 resolution.

Another key capability of mobile devices is their networking functionality. As detailed in the specifications for the future 802.11n and HSPA+ wireless networking standards, it is expected that within the next few years transfer rates will reach up-to 74Mbps [2] [8].

Previous rendering implementations for mobile devices include Duguet et al. [6] who presented a system using hierarchical packed point representations based on recursive grid data structures, demonstrating how such point-based approaches are well adapted to mobile devices with limited memory and screen resolution. In order to communicate visual conceptual information, techniques such as Illustrative rendering are used to efficently present such data. Work by Huang et al. investigates how such rendering techniques can be adapted to mobile platforms [12].

To our knowledge, any previous attempts at physically-based rendering systems for mobile platforms have focussed on purely GPU approaches. A thorough overview of GPU-based ray-tracing implementation strategies on mobile devices, taking into account their limited power constraints is described by Lohrmann [15]. We believe by making optimal use of the processing power and network capabilities as well as using the display size to our advantage, it is possible to produce high-fidelity graphics in reasonable time.

2.2 High-Fidelity Rendering

High-fidelity rendering methods compute accurate physically-based simulations of the lighting distribution in a virtual environment. These methods use physically-based materials and lighting and simulate the transport of light using algorithms termed global illumination algorithms. There are two major approaches to high-fidelity rendering: radiosity [10] and ray tracing [32]. Of these ray tracing and similar algorithms have become more popular recently.

Ray tracing algorithms simulate the transport of light as groups of particles travelling in straight lines and interacting with the environment. Rather than shooting photons from the lights and allowing them to eventually be absorbed by the camera, it is more efficient to trace these paths in reverse, from the camera into the virtual environment, typically one ray or more per pixel. When rays interact with surfaces and media in the environment, further rays are spawned which simulate certain lighting properties such as reflection, refraction, colour bleeding, shadow generation, participating

media *etc.*

Ray tracing methods use acceleration data structures, based on spatial subdivision methods such as kd-trees, grids and octrees (see Havran [11] for an overview), to identify which objects a ray hits. These algorithms make ray tracing scale logarithmically with the number of objects in a scene as opposed to the linear scaling of rasterisation algorithms. The computational complexity of ray tracing is linear to the number of rays shot from the virtual camera. This makes it an ideal candidate for rendering complex scenes on SFF devices with low resolutions.

Unfortunately, ray tracing computation can be computationally expensive and only recently has interactive ray tracing become viable on modern desktop PCs [26]. Interactive ray tracing algorithms make use of multi-core and instruction level parallelism found in modern CPUs [30], careful memory management, fast acceleration structures [23] and spatial coherence [21] to maintain interactive rates.

2.3 Client-Server Rendering

The first client-server rendering solution was introduced by Funkhouser [9]. This system demonstrated that by computing participants' visibility on the server, a reduction in transmitted data is possible. The concept of on-demand transmission of data according to regions of interest was employed by Schneider [22] and Teler [25] who used the idea of 'Level of Detail' to adaptively transmit data based on criteria such as available bandwidth and computational power. Engel et al. developed a remote visualisation system for OpenInventor and Cosmos3D applications using a Java client to receive compressed images and return interaction events via CORBA requests [7].

Recently, Quillet et al. [20] presented a system to transmit conceptual information using a client-server system for remote rendering on mobile devices. A system utilising the computational power on both the client and server to render line graphics was presented by Diepstraten et al. [5]. In order to reduce transmission of complete images, taking advantage of the primitives of such lines enables a significant reduction in the data to transmit.

3. FRAMEWORK FOR HIGH FIDELITY RENDERING ON MOBILE DEVICES

The computational requirements for high-fidelity graphics are far beyond the current capabilities of current mobile devices. In order to achieve reasonable rendering rates on such platforms a great deal of optimisation is required. We will give an overview of a possible framework for delivering high fidelity content.

Although it is possible to render complete images on a server and stream these to our mobile system this solution does not make use of the processing power on the mobile client and has the overhead of transferring all this data. If we adopted a mobile-only render, even with the next generation of hardware it is unlikely that reasonable frame rates will be achievable.

Both these systems have advantages and disadvantages. Our proposed solution is to make optimal use of both the client and server and the associated interconnect to reduce overall rendering time. It is important to take into account the "cost" of the transmission of the data. Not only is the bandwidth of mobile connections highly variable, but so too are the associated financial costs. We suggest that the output of such a system balances rendering on the client and server whilst minimising the overall costs of rendering and transmitting.

Although it still may not be possible using this framework to render at a reasonable rate, we can make additional use of the techniques demonstrated by Debattista [4]. By taking into account that the results would be viewed by a human, we can select which regions require greater rendering effort and intelligently distribute our rendering budget. In addition to these techniques, through analysis of the perceptual characteristics of SFFs it is possible to calculate the thresholds beyond which any further rendering effort will cease to produce greater perceptual gains [1].

In the following sub-sections we will detail our initial steps towards realising such a framework.

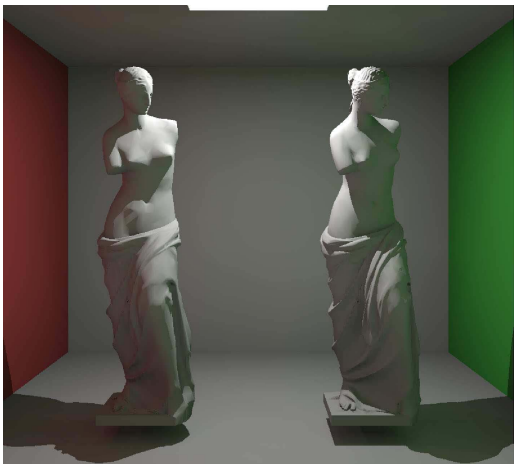

Figure 2. Global Illumination of Cornell Box with Venus de Milo models.

3.1 High-Fidelity Renderer

In order to demonstrate the potential of our framework we have developed a portable high-fidelity renderer based on methods developed for fast ray tracing [30].

Portability has been one of the major design goals of our renderer and our ray tracing kernel although tight, is adaptable. It takes advantage of current graphics hardware and can exploit multiple cores. Most current interactive ray tracers use SIMD operations to speedup performance and organise their data structures into "structures of arrays" as opposed to "arrays of structures". Since our design is targeted to different hardware, and modular code which can be easily interchanged and/or updated when newer algorithms are developed, we have opted for the more standard "arrays of structures" approach. However, we ensure that our data structures are memory coherent by keeping frequently accessed data closer in physical memory to maintain coher-

ence across all platforms.

The current implementation uses ray tracing for the computation. The first intersection is accelerated via shaders on systems that support fast graphics hardware. These shaders return depth and object properties at the intersection point. Successive intersections for whitted-style ray tracing, for determining specular and transparent components of materials are computed using recursive ray tracing. When there is no hardware acceleration, recursive ray tracing is used for the entire computation. Direct lighting is computed at the moment using Phong shading, but support for other shaders such as Ward [31] could easily be added. Only hard shadows are computed and area light sources are simplified as a number of point light sources. The indirect diffuse computation is computed using our implementation of an accelerated version of indirect radiosity [14] which had been presented in [28]. Currently, the acceleration structure uses a dynamic BVH based on [27].

Our implementation has worked successfully on IBM PCs and compatibles running Linux, various versions of Windows and MacOS X, as well as the GP2X, see Figure 1. A screenshot of the renderer performing global illumination can be seen in Figure 2.

3.2 Client-Server application

Communication occurs between the server and client using a TCP/IP connection which is established via a socket connection. In order to reduce the amount of transmitted data, data is compressed before being sent to the client where it is decompressed and combined with a portion of the image data that has already been rendered by the client. The final image is then sent to the frame buffer to be displayed.

3.3 Cost Function

The processing power of devices varies significantly, so too does the bandwidth of mobile networks. As discussed in Section 3, we have proposed a system to minimise overall rendering time by using a client-server architecture. In this section we introduce our metric for optimising the distribution of rendering.

In our system, the total rendering time of the system can be written as:

$$T_{mobile} = mT_m$$

where T_{mobile} is the total cost of running on the mobile, T_m is the average cost of rendering a pixel on the mobile and m is the fraction of pixels to be rendered on the SFF device. Similarly:

$$T_{server} = s(T_s + T_t)$$

where T_{server} is the total cost of running on the server, T_s is the average cost of rendering a pixel on the server, T_t is the cost to transfer one pixel to the mobile device and s is the fraction of the number of pixels to be rendered on the server. Note that T_t could be a function in itself that takes into account latency and maximum bandwidth.

If we were using one mobile device and one server, ideally we would spend the same amount of time rendering on each to maximise computation, such that $T_{mobile} = T_{server} = T_{total}$. The solution is made simpler when one considers that $m + s = 1$, and all that remains is to solve for m for the minimum T_{total}.

The client-server architecture computes as a quick overture, once per connection, the ideal value of m for computation on the mobile device and transmits this value to the mobile device. The mobile device computes m amount of the pixels in terms of scanlines. The server computes the rest and transmits them upon completion. The computation can be extended to take into account the general case of multiple servers or clients.

4. RESULTS

In order to analyse the rendering speed of our client-server system, we ported our renderer (described in Section 3.1) onto a Tablet PC. The client was powered by a 1Ghz Intel Pentium M with 502MB RAM running Windows XP Tablet Edition. This specification is comparable to current UMPCs (Ultra Mobile PCs), and we believe is a decent representation of other next generation mobile devices. As our target resolution, we chose 320×240 which is average for current high end SFFs.

For our server we used a MacPro with two 3 GHz quad core Intel Xeons with 4GB DDR2 memory. Data from the server was transmitted using wireless standard 802.11g which provided an average throughput of 19Mbit/s.

We tested our system using two different scenes, the Buddha (67,244 polygons) in Figure 4 and a Cornell Box with Venus de Milo and a reflective sphere (45,109 polygons) in Figure 3. In the calculation to optimally balance rendering it was assumed that the cost of receiving network data on the mobile and the cost of compression/decompression was constant per pixel, this can be integrated into the cost function. We assume a compression ratio of 4:1, based on compression rates observed using ZLIB. The average cost of transmission for the Tablet per ray using a wireless connection came out at 400ns. We compared how our system dealt with varying setups, the results of which can be seen in Table 1. Based on the costs of rendering on client and the server, our cost function was used to calculate the optimal balance of rendering effort between the client and server, providing the minimum possible rendering cost and the associated distribution.

Using the system as described above, we found that to minimise the rendering time for the Tablet, about 10% of the scene is required to be rendered on the mobile. By applying our cost function, we were able to demonstrate a decrease in rendering time per pixel of magnitude 8-13, delivering an approximate frame of between 10-23 FPS, providing near interactive rates. From the rates in Table 1, it may seem a reasonable question to ask why one would not render the entire content on the server and make use of the significant resources. It is however, important to note that these results are somewhat idealised; although the servers' frame rates are impressive, it is an unreasonable assumption to believe that clients will have the luxury of a dedicated server. If these

Table 1. Rendering time per pixel in nanoseconds on server, client and speed-up achieved using our cost function optimisation.

Scene	No. server processors	Server Time per pixel (ns)	Client Time per pixel (ns)	Cost function optimisations	
				Client share	Time per pixel (ns)
Cornell Box	2	989	8841	13.58%	1200
	4	585	8841	10.02%	886
	8	325	8841	7.58%	669
Buddha	2	520	6688	12.09%	809
	4	390	6688	10.56%	706
	8	208	6688	8.33%	557

resources were to be distributed among more clients, the balance of rendering would be altered significantly.

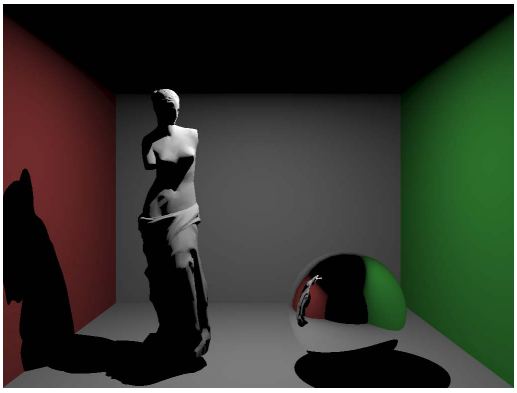

Figure 3. Ray traced Cornell Box with Venus de Milo model and reflective sphere.

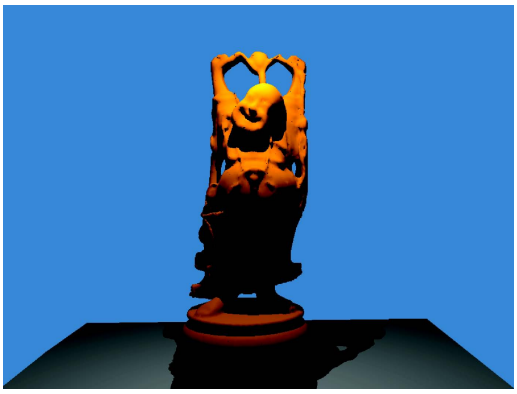

Figure 4. Ray traced Buddha model.

5. CONCLUSIONS AND FUTURE WORK

The results from our client-server rendering system have demonstrated a method of delivering high-fidelity content at near interactive rates. Although our results were produced on a Tablet PC, it is important to take into account the trends of improvements to mobile hardware and network speeds which would significantly reduce the load placed on the server. From our results we can conclude that due to the low resolution of SFF devices, ray tracing based rendering may be possible at a reasonable rate within the next few years, opening up the use of such mobile devices for many applications which require high-fidelity physics-based imagery. Our initial implementation of a client-server solution to minimise rendering time demonstrates how adaptive balancing of rendering load can be used to reduce this further.

Although it is a matter of significant debate whether pure ray tracing based rendering delivers any further quality over rasterisation, it is trivial to extend ray tracing using methods such as path tracing or distributed ray tracing to deliver a full global illumination solution capable of producing a complete physically-based solution including indirect diffuse illumination and caustics.

As part of the overall view of the framework discussed in Section 3, in the future we intend to investigate how our cost function can be extended to multiple servers/clients. It is also possible to improve our cost function to take into account additional variables such as the financial cost of transmitting the data such that the minimal rendering time can be produced whilst meeting financial constraints.

A further step to reducing rendering costs is to take into account the effect of human visual perception, allowing us to reduce rendering in regions of a scene without introducing perceptible errors [33, 3, 24]. By selectively targeting our rendering effort using perceptual models [33, 3, 24, 16] we can minimise any potentially redundant computation without reducing perceptual quality.

In addition to employing selective rendering techniques, we will take into account analysis of the unique perceptual thresholds on SFF devices which have be proven to differ from other display devices [1].

6. REFERENCES

[1] M. Aranha, K. Debattista, A. Chalmers, and S. Hill. Perceived rendering thresholds for high-fidelity graphics on small screen devices. In *Theory and Practice of Computer Graphics 2006*, pages 133–140. EG, July 2006.

[2] Broadcom. 802.11n: Next-generation wireless lan technology. Technical report, Broadcom, apr 2006.

[3] K. Cater, A. Chalmers, and G. Ward. Detail to attention: exploiting visual tasks for selective rendering. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 270–280, Aire-la-Ville, Switzerland, Switzerland, 2003.

Eurographics Association.

[4] K. Debattista. *Selective Rendering for High Fidelity Graphics*. PhD in Computer science, University of Bristol, 2006.

[5] J. Diepstraten, M. Gorke, and T. Ertl. Remote line rendering for mobile devices. *Computer Graphics International, 2004. Proceedings*, pages 454–461, 16-19 June 2004.

[6] F. Duguet and G. Drettakis. Flexible point-based rendering on mobile devices. *IEEE Computer Graphics and Applications*, 24(4):57–63, 2004.

[7] K. Engel, O. Sommer, and T. Ertl. A framework for interactive hardware-accelerated remote 3d-visualization. In *In Proceedings of EG/IEEE TCVG Symposium on Visualization VisSym '00*, pages 167–177, 291, 2000.

[8] U. Forum. 3g/umts evolution: towards a new generation of broadband mobile services. Technical report, UMTS Forum, dec 2006.

[9] T. A. Funkhouser. RING: A client-server system for multi-user virtual environments. In *Symposium on Interactive 3D Graphics*, pages 85–92, 209, 1995.

[10] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile. Modeling the interaction of light between diffuse surfaces. In *SIGGRAPH '84*, pages 213–222. ACM Press, 1984.

[11] V. Havran. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[12] J. Huang, B. Bue, A. Pattath, D. S. Ebert, and K. M. Thomas. Interactive illustrative rendering on mobile devices. *IEEE Comput. Graph. Appl.*, 27(3):48–56, 2007.

[13] S. Iyer, L. Luo, R. Mayo, and P. Ranganathan. Energy-adaptive display system designs for future mobile environments. In *In Proceedings of The First International Conference on Mobile Systems, Applications, and Services*, pages 245–258, San Francisco, CA, USA, 2003.

[14] A. Keller. Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[15] P. J. Lohrmann. Energy-efficient interactive ray tracing of static scenes on programmable mobile gpus. Master's thesis, Worcester Polytechnic Institute, 2007.

[16] P. Longhurst, K. Debattista, and A. Chalmers. Snapshot: A rapid technique for driving a selective global illumination renderer. In *WSCG 2005*, February 2005.

[17] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian. Integrated power management for video streaming to mobile handheld devices. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 582–591, New York, NY, USA, 2003. ACM Press.

[18] S. Pasricha, M. Luthra, S. Mohapatra, N. Dutt, and N. Venkatasubramanian. Dynamic backlight adaptation for low-power handheld devices. *IEEE Design and Test of Computers*, 21(5):398–405, 2004.

[19] K. Pulli, T. Aarnio, K. Roimela, and I. Vaarala. Designing graphics programming interfaces for mobile devices. *Computer Graphics and Applications*, 25:66–75, nov 2005.

[20] J.-C. Quillet, G. Thomas, X. Granier, P. Guitton, and J.-E. Marvie. Using expressive rendering for remote visualization of large city models. In *Web3D '06: Proceedings of the eleventh international conference on 3D web technology*, pages 27–35, New York, NY, USA, 2006. ACM.

[21] A. Reshetov, A. Soupikov, and J. Hurley. Multi-level ray tracing algorithm. *ACM Trans. Graph.*, 24(3):1176–1185, 2005.

[22] B.-O. Schneider and I. Martin. An adaptive framework for 3d graphics over networks. *Computers and Graphics*, 23:867–874(8), December 1999.

[23] M. Shevtsov, A. Soupikov, and A. Kapustin. Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes. *ACM Trans. Graph.*, 26(3), 2007.

[24] V. Sundstedt, K. Debattista, and A. Chalmers. Perceived aliasing thresholds in high-fidelity rendering. In *APGV 2005 - Second Symposium on Applied Perception in Graphics and Visualization (poster)*. ACM, August 2005.

[25] E. Teler and D. Lischinski. Streaming of complex 3D scenes for remote walkthroughs. In A. Chalmers and T.-M. Rhyne, editors, *EG 2001 Proceedings*, volume 20(3), pages 17–25. Blackwell Publishing, 2001.

[26] I. Wald. Realtime Ray Tracing and Interactive Global Illumination. *PhD thesis, Saarland University*, 2004.

[27] I. Wald, T. Ize, A. Kensler, A. Knoll, and S. G. Parker. Ray Tracing Animated Scenes using Coherent Grid Traversal. *ACM Transactions on Graphics*, 2006. (Proceedings of ACM SIGGRAPH 2006).

[28] I. Wald, T. Kollig, C. Benthin, A. Keller, and P. Slusallek. Interactive Global Illumination using Fast Ray Tracing. In *13th EUROGRAPHICS Workshop on Rendering*, Pisa, Italy, 2002.

[29] I. Wald, T. J. Purcell, J. Schmittler, C. Benthin, and P. Slusallek. Realtime Ray Tracing and its use for Interactive Global Illumination. In *Eurographics State of the Art Reports*, 2003.

[30] I. Wald, P. Slusallek, and C. Benthin. Interactive Distributed Ray Tracing of Highly Complex Models. In *12th EUROGRAPHICS Workshop on Rendering*, pages 274–285, London, United Kingdom, June 2001.

[31] G. Ward. Measuring and Modeling Anisotropic Reflection. In *SIGGRAPH'92 - 19th International Conference on Computer Graphics and Interactive Techniques*, pages 266–272. ACM Press, 1992.

[32] T. Whitted. An improved illumination model for shaded display. In *SIGGRAPH '80*, page 14. ACM Press, 1980.

[33] H. Yee, S. Pattanaik, and D. P. Greenberg. Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. *ACM Trans. Graph.*, 20(1):39–65, 2001.