Interactive Mixed Reality Rendering in a Distributed Ray Tracing Framework

Andreas Pomi and Philipp Slusallek
Computer Graphics Group, Saarland University
Im Stadtwald - Building 36.1, 66123 Saarbrücken, Germany
{pomi, slusallek}@cs.uni-sb.de

Abstract

Photo-realistic rendering methods are required to achieve a convincing combination of real and synthetic scene parts in mixed reality applications. Ray tracing is a long time proved candidate, but when it comes to interactive frame rates GPU based rendering has usually been preferred. Recent advances in distributed interactive ray tracing suggest to explore also its potential for mixed reality rendering applications.

In this paper we show how simple extensions to the OpenRT ray tracing framework enable interactive applications from different ends of the mixed reality spectrum. We point out the benefits of a ray tracer over traditional GPU approaches when it comes to flexibility and complex compositing operations.

1. Introduction

The spectrum of mixed reality applications spans from inserting real objects or actors into a virtual scene (e.g. a virtual TV studio) up to augmenting live video with rendered objects (AR). In order to provide a convincing result for the viewer, photo-realistic rendering methods are needed for taking care of visual cues like shadows and reflections.

Current GPU hardware can be used to create photo-realistic mixed reality scenes (e.g. [3]), but seems rather inflexible for this task. Recent advances in ray tracing [10] provide a new basis for photo-realistic rendering at interactive frame rates. Ray tracing enables simple implementation of complex compositing methods ([4]) that are hard to achieve with GPU based approaches.

In the following we give a brief introduction to interactive ray tracing, and some basic extensions for mixed reality. We show application examples from different ends of the mixed reality spectrum and point out benefits of ray tracing over other approaches.

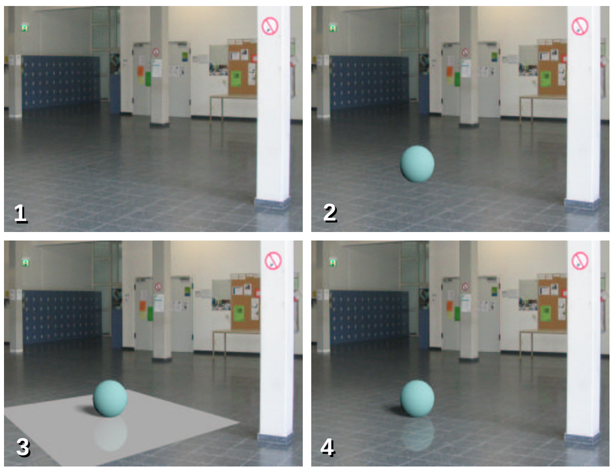

Figure 1. Simple AR compositing example with OpenRT. 1) Background view. 2) Inserted object, without a shadow it seems to float. 3) A stand-in object for the floor to generate a soft shadow and reflection from real captured illumination. 4) Final view after applying a differential rendering method.

2. Distributed Interactive Ray Tracing

There were a number of attempts to perform ray tracing at interactive frame rates e.g. on GPUs [9]. A pure software (CPU) based distributed approach [10] has proven its potential and provides a much more flexible framework than GPU approaches in general.

Ray tracing is capable of arbitrary complex geometry and advanced shading methods e.g. using interactive global illumination [2]. It is an ideal candidate for photo-realistic mixed reality rendering. An OpenGL-like API (OpenRT [5]) ensures easy implementation of applications. Compared to GPU based approaches pure software ray tracing does not suffer from hardware resource conflicts.

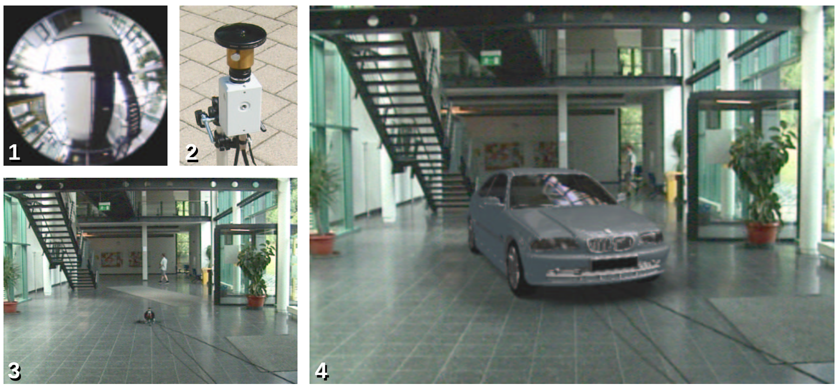

Figure 2. Car model inserted into a live captured background. The car is lit by the incident light captured with a HDR video light-probe at the car position. 1) A light-probe HDR frame. 2) HDR camera with 180 degree fish-eye lens. 3) A background frame. 4) Final video output with soft shadows cast onto the real floor. Note the transparent car windows.

A typical software implementation of OpenRT consists of a number of computers connected by a commodity network. The application runs on a server on top of the OpenRT API, which hides the distribution issues from the user. The rendered image is split into tiles (e.g. 32x32 pixels), which are scheduled for rendering on the clients.

Each clients runs a highly optimized ray tracer implementation that is roughly a factor 30 faster than traditional ray tracing systems [10]. The rendering performance can be easily scaled by adding more clients, and is only limited by the available network bandwidth.

2.1. Mixed Reality Extensions to OpenRT

A simple approach to mixed reality is to include the real world in the rendering process by using live video streams. In [8] we came up with two extensions to the OpenRT framework: *Streaming video textures* and *in-shader view compositing*.

Video textures allow for synchronized streaming of live video from dedicated video texture servers with low latency. A multicast networking approach ensures scalability in the number of rendering clients.

In-shader view compositing is a variation to video textures for inserting a video background behind rendered objects. Compared to a video texture, we only stream the colors of the appropriate tile from the background instead of the whole image. This yields lower network bandwidth and latency. The background color for each primary ray can be accessed in the shader for performing *differential rendering* methods [4].

3. Application Examples

3.1. Virtual Object Insertion into Live Video

Figure 1 shows a simple AR compositing example: A sphere is rendered in front of a video background. A stand-in object for the local floor is used to create a soft shadow and a reflection. Differential rendering [4] integrates the effect into the background video.

A more sophisticated example using image-based lighting is shown in Figure 2. A virtual car is placed into a live video background. A HDR video camera with a 180 degree fish-eye lens is used to capture the incident light. The light-probe image is streamed as a video texture, and is used for both, lighting the car and creating a soft shadow on the real floor.

3.2. Actor Insertion into Synthetic Scenes

To insert live actors into a virtual scene, billboards can be used. A video texture streams the video of the actor captured in front of a green or blue background. Chroma-keying is performed inside the billboard shader (*in-shader*, see Figure 3.1).

However billboards have a number of drawbacks due to their 2D nature (see e.g. [7]). Our framework enables another approach: A number of video textures with different views of the actor, combined with a 3D image-based visual hull [6] reconstruction algorithm provides correct shadows and reflections compared to 2D billboards (see Figure 3.2). A visual hull shader assigned to a box provides a simple integration into the scene. Shadows and reflections are computed automatically in a ray tracing framework [7].

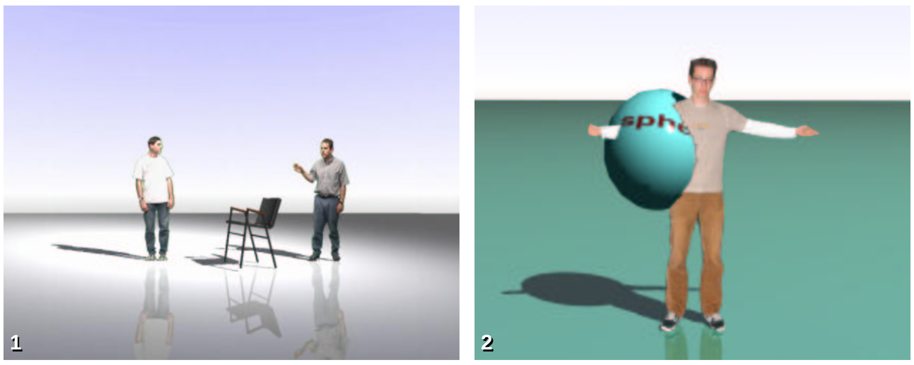

Figure 3. 1) 2D video texture billboards with in-shader compositing. 2) In-shader 3D visual hull reconstruction with exact occlusion by a sphere, lighting effects and reflection.

4. Results

Table 1 shows the frame rates we currently achieve for our examples at a video resolution of 640x480. Note the number of samples used for lighting and soft shadow generation. The rendering clients were dual Athlon MP 1800+ PCs connected via 100Mbit/s Ethernet to a central switch. The rendering server is connected via Gigabit. The video delay is about 3-4 frames.

5. Conclusion and Future Work

We have shown that interactive ray tracing is indeed suitable for mixed reality rendering, and there still remains a large space for further exploration.

Compared to rasterization based approaches we still achieve lower frame rates, but a pure software ray tracer allows for much simpler implementation and combination of rendering effects than current GPUs since no hardware resource conflicts can occur and arbitrary complex rendering algorithms can be implemented.

For the future, we aim for providing an ARToolkit [1] version for OpenRT. We plan farther research on interactive image-based lighting (using stand-in geometry [3]), and to integrate the lighting process in our OpenRT based interactive global illumination system [2].

Example	Samples	#CPUs	fps
Sphere, soft shadow (Fig. 1)	40	8	9.1
Sphere, hard shadow	1	8	20.5
Car, 208259 tris (Fig. 2)	40	24	4.5
Billboards (Fig. 3.1)	1	8	16
Visual hull shader (Fig. 3.2)	1	16	15.5

Table 1. Frame rates

References

[1] ARToolKit. http://www.hitl.washington.edu/artoolkit/.

[2] C. Benthin, I. Wald, and P. Slusallek. A Scalable Approach to Interactive Global Illumination. *Computer Graphics Forum*, 22(3):621–630, 2003. (Proceedings of Eurographics).

[3] J. Cook, S. Gibson, T. L. Howard, and R. J. Hubbold. Realtime photorealistic augmented reality for interior design. In *ACM Siggraph 2003 Conference Abstracts and Applications*. ACM Press, July 2003.

[4] P. Debevec. Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-based Graphics with Global Illumination and High Dynamic Range Photography. *Computer Graphics (Proceedings of SIGGRAPH 98)*, 32(4):189–198, 1998.

[5] A. Dietrich, I. Wald, C. Benthin, and P. Slusallek. The OpenRT Application Programming Interface – Towards A Common API for Interactive Ray Tracing. In *Proceedings of the 2003 OpenSG Symposium*, pages 23–31, Darmstadt, Germany, 2003. Eurographics Association.

[6] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, and L. McMillan. Image-based visual hulls. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 369–374. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[7] A. Pomi, S. Hoffmann, and P. Slusallek. Interactive In-Shader Image-Based Visual Hull Reconstruction and Compositing of Actors in a Distributed Ray Tracing Framework. In *1. Workshop VR/AR, Chemnitz, Germany*, 2004.

[8] A. Pomi, G. Marmitt, I. Wald, and P. Slusallek. Streaming Video Textures for Mixed Reality Applications in Interactive Ray Tracing Environments. In *Proceedings of Virtual Reality, Modelling and Visualization (VMV)*, 2003.

[9] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan. Ray Tracing on Programmable Graphics Hardware. *ACM Transactions on Graphics*, 21(3):703–712, 2002. (Proceedings of SIGGRAPH 2002).

[10] 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*. Eurographics, 2003.