Interactive Ray Tracing for Virtual TV Studio Applications

Andreas Pomi and Philipp Slusallek
Computer Graphics Lab
Saarland University
Im Stadtwald 36.1
D-66123 Saarbrücken, Germany
email: {apomi,slusallek}@graphics.cs.uni-sb.de
www: http://graphics.cs.uni-sb.de/MR

Abstract

In the last years, the well known ray tracing algorithm gained new popularity with the introduction of interactive ray tracing methods. The high modularity and the ability to produce highly realistic images make ray tracing an attractive alternative to raster graphics hardware.

Interactive ray tracing also proved its potential in the field of Mixed Reality rendering and provides novel methods for seamless integration of real and virtual content. Actor insertion methods, a subdomain of Mixed Reality and closely related to virtual television studio techniques, can use ray tracing for achieving high output quality in conjunction with appropriate visual cues like shadows and reflections at interactive frame rates.

In this paper, we show how interactive ray tracing techniques can provide new ways of implementing future virtual studio applications.

Keywords: Mixed Reality, Rendering, Interactive Ray Tracing, OpenRT, Streaming Video Textures, In-Shader Rendering, Actor Insertion, 3D Compositing, Video Billboards, Image-Based Visual Hull, Virtual TV Studio

1 Introduction

Novel approaches to the classic ray tracing algorithm made it possible to achieve interactive frame rates [Wal04, WPS+03]. Interactive ray tracing provides new ways of implementing interactive applications that gain from its benefits like modularity and rendering quality. In application areas like interactive Mixed Reality (MR) rendering, ray tracing can be an alternative to traditional approaches [PS04].

For convincing rendering results in virtual television studio applications, an appropriate rendering method is needed. It should provide the possibility to easily enhance the composite scene by visual cues. Visual cues, reflecting the visual interaction of the real actors with the artificial scene, can be provided by shadows, reflection, and occlusion effects.

Interactive ray tracing ideally suits this demands. Its modularity allows for a completely independent implementation of object representation and visual effects. GPU (Graphics Processing Unit) based approaches often lack this modularity and suffer from restrictions when a combination of rendering methods (e.g. hardware-supported 3D reconstruction of an actor combined with a shadow casting method and reflections) is needed.

Ray tracing also facilitates the combination process of the actor and the virtual scene (*compositing*). The demand-driven concept of ray tracing in combination with the unrestricted shading possibilities allows new ways for implementing compositing.

The concept of *in-shader compositing* moves compositing from a post-process directly into the shading calculations and thus allows the closest visual interaction between actor and scene.

In the remainder of this paper, we give an overview

urn:nbn:de:0009-6-2480, ISSN 1860-2037

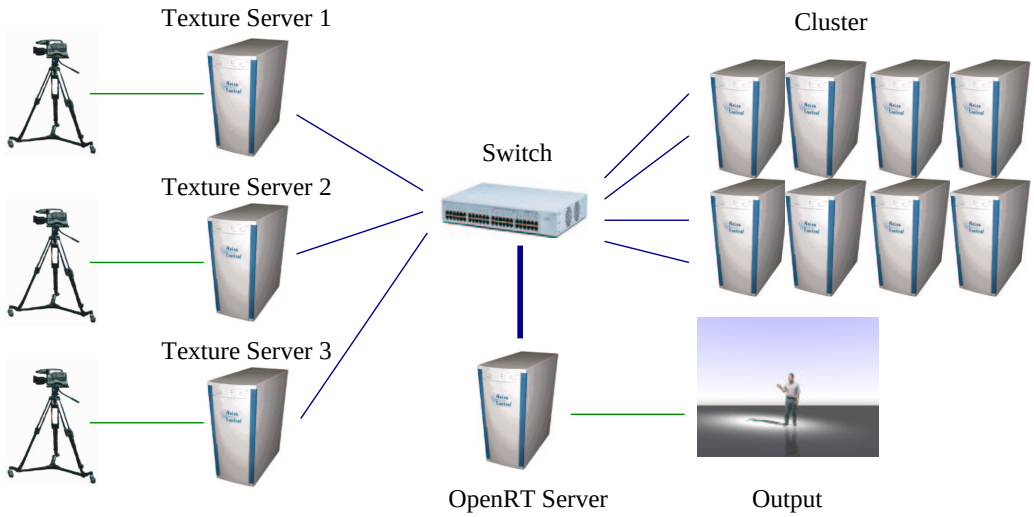

Figure 1: A typical setup for an OpenRT interactive ray tracing system with streaming video textures. The video texture servers stream the video content via multicast networking to the rendering client cluster. The OpenRT server runs the rendering application, accomplishes texture synchronization and provides the video output.

of the state-of-the-art in interactive, distributed ray tracing within the field of MR rendering. The advantages of using ray tracing for virtual studio applications are pointed out and ray tracing based methods (video billboards and a 3D reconstruction shader based on visual hull rendering) for seamless actor insertion into virtual scenes are described.

2 Interactive Ray Tracing

There are several approaches to interactive ray tracing: Software ray tracing has been accelerated by new algorithmic approaches by more than a factor of 30 [Wal04] on a single CPU. Ray tracing easily parallelizes linearly on shared-memory machines as well as on PC clusters. The OpenRT system [WD03] is an example of such an interactive ray tracing framework.

Another approach would be to use the programmability of modern raster graphics chips (GPUs). However, this approach is restricted by the limited programming model [PBMH02]. A better alternative is provided by a graphics hardware architecture especially designed to perform ray tracing, like the SaarCOR [SWS02, WSS05] system.

Software ray tracing offers a number of advantages over traditional, GPU based rendering techniques [MH99]. The high modularity of a ray tracer allows to combine rendering effects in a very straightforward way. Resource conflicts and limitations as experienced with GPUs do not occur. Since shading is performed in software, arbitrary complex shading algorithms (e.g. global illumination methods [BWS03]) can be used. The modularity also allows to use other geometry representations than triangles like spline surfaces (e.g. [BWS04]) or volumetric representations (e.g. [MFK+04]). The image-based visual hull shader presented below is an example for a non-triangle data representation.

Ray tracing is *output-sensitive*, i.e. the computing power that is necessary to render an output frame depends mainly on the frame content. Expensive shading calculations need only be computed for the visible parts of a scene. Ray tracing is thus a *demand-driven* algorithm.

Another advantage of using ray tracing is its ability to handle far more complex models than are possible with raster graphics methods. Compared to the linear GPU approach, ray tracing scales logarithmically in the number of triangles [Gla95, Hav00]. Highly detailed models with more than 10^9 triangles can be rendered even with soft shadows [DWS04].

2.1 The OpenRT Framework

Since the power of a typical PC is often not sufficient for high-performance applications, it is necessary to combine the processing capabilities of several hosts.

OpenRT [DWBS03, WD03] provides a framework for distributed interactive ray tracing. The OpenRT library features an interface very similar to OpenGL and hides the distribution and networking aspects from the application.

The host with the rendering application is referred to as the OpenRT server. A number of rendering clients (a cluster) provide the necessary computing power. Beside the application API, OpenRT also features a shader programming API for surface, light-source, and camera shaders.

In [PMWS03], we introduced two extensions to OpenRT to support interactive Mixed Reality rendering methods. *Streaming video textures* allow for using live video in the shading process. Since shading is performed on the clients, the video content is streamed by dedicated texture servers (see Figure 1). Multicast networking [Ste98] and a synchronization mechanism ensure scalability in the number of hosts. Figure 2 shows how a virtual scene can be enhanced by live video.

Figure 2: A streaming video texture used on the screen of a TV set. The video content is also used for dynamic lighting of the room.

The second extension is an *Augmented Reality view compositing* method [PS04]. Here, a live video background is streamed to the rendering clients and included into the rendering process (*in-shader compositing*, [Pom05]). This enables the easy implementation of *differential rendering* methods [Deb98] for achieving convincing visual cues. Both extensions are implemented as plugins to OpenRT without changes to the core system.

2.2 Interactive Ray Tracing for Virtual Studio Applications

Interactive ray tracing can provide a number of advantages over traditional GPU-based rendering for virtual studio applications. The high visual quality can be used to provide visual cues of the positions of the inserted persons. Lighting effects can be used to cast shadows and reflections of the real actor onto the virtual objects. Live video can be included into the scene for implementing virtual video displays.

There are also a number of technical advantages: Ray tracing inherently supports video field rendering, i.e. the necessary computing power is exactly the same for rendering 25 frames per second or 50 interlaced fields [Poy03].

The software frame-buffer of a ray tracer allows easy connection to professional video frame-buffer boards that support the demands of a video studio infrastructure like video synchronization and digital video (SDI) outputs.

Ray tracing also supports easy implementation of arbitrary camera models (e.g. via the OpenRT camera shader). This makes the matching of studio cameras to the virtual camera easily possible. Camera effects like depth-of-field can be correctly rendered and need not be approximated [WGDD98].

2.3 The Concept of In-Shader Compositing

The technique of combining two images (or video streams) into a single image is referred to as *compositing* [Bri99, Kel00]. Traditional compositing methods are based on the formal 2D image compositing algebra introduced by Porter and Duff [PD84] and are implemented as a post-process. The compositing process is controlled by auxiliary information, for example an alpha-channel or depth information.

A traditional video compositing system for virtual studio purposes would thus consist of a live video input from the studio, showing the actor in front of a background that can be used to generate the auxiliary information, e.g. by chroma keying. The second image is provided by a renderer, and in a 2D post-process the compositing of both images is performed by special hardware (a video keyer [Poy03]).

The concept of *in-shader compositing* [Pom05] provides an alternative to the traditional approach: the compositing is performed directly inside the shading process. Here, the full 3D information is still available (e.g. occlusion/depth, surface normals, etc.). There is no need to carry additional auxiliary information to a post-process. This ideally suits the demands of a ray tracing framework. As a byproduct, the compositing is performed on demand since a ray tracer is demand-driven. The actor insertion methods presented in the following section can be seen as applications of in-shader compositing.

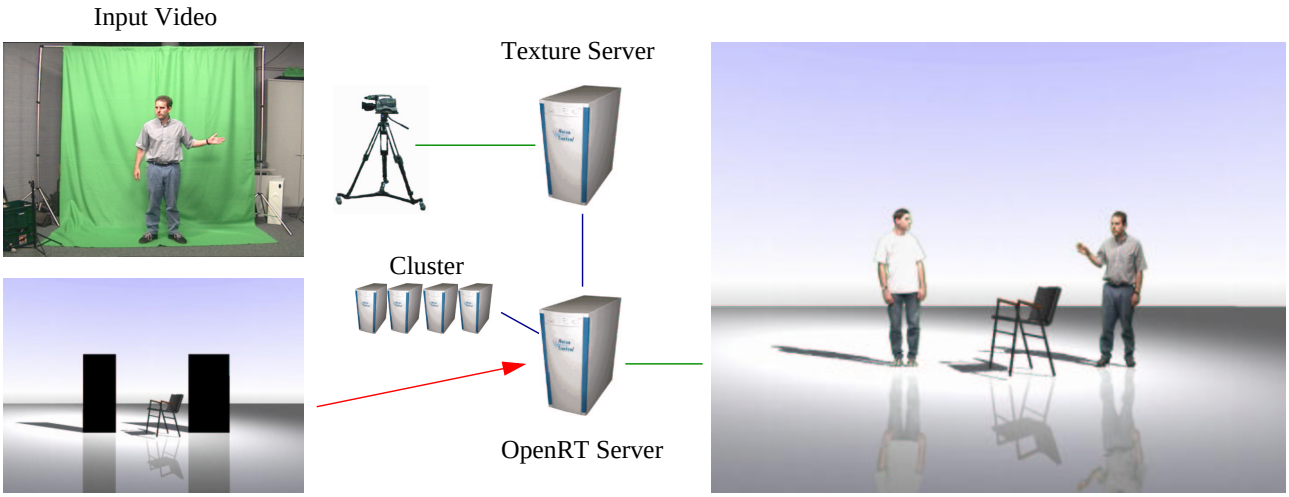

Figure 3: An OpenRT video billboard application. The video image of a person in front of a greenscreen is streamed as a video texture. The scene description contains two billboard rectangles with an associated video billboard shader. Chroma keying is performed *in-shader*. The resulting video output shows two persons inserted into the scene. Lighting effects and reflections provide the necessary visual cues.

3 The Actor Insertion Problem

The basic rendering problem of virtual studios can be referred to as the *actor insertion problem*: one ore more persons (actors) are placed into a virtual scenario. This is a special case of the general compositing problem of combining virtual and synthetic parts.

For a convincing result, it is necessary to provide the feeling of *interaction* between the virtual and synthetic components. The composite image is not accepted by the audience if it lacks *visual cues* of interaction. These cues can for example be shadows, cast by the actor onto the virtual scene or a shiny floor, showing a glossy reflection of the person. Also occlusion (depth) of the actor by scene parts plays an important role, especially when walking around.

These effects are hard to achieve with the traditional approach. Tricks, like generating the effects in the real world (e.g. using a shiny floor in the studio) and using sophisticated keying hardware to extract them together with the actor, are very limited. Think of the distortion of the actors' shadow when it is cast onto a complex synthetic object. For special cases, the use of auxiliary cameras can provide a solution for this problem [WGO00].

A ray tracing framework using the in-shader approach to compositing provides an alternative solution to the actor insertion problem. In-shader compositing allows for a high visual interaction between the actor and the scene. In the following sections, we discuss two in-shader compositing methods that can be used for actor insertion.

3.1 Video Billboards

A simple method to include an actor into a virtual scene is to provide an image of the actor by using a *billboard*. The billboard concept is well known from computer games [MH99]. It features a flat (2D) rectangle perpendicular to the view axis that is textured with an image. The image contains transparent parts, e.g. when it shows a person, the parts outside the silhouette are transparent. Billboards can be easily extended for video applications by using the above described streaming video textures.

Figure 3 shows a video billboard system implemented with OpenRT. The actors are shot in front of a greenscreen in the studio. The camera output is streamed as a video texture. The billboard shader associated with the rectangles performs a video texture lookup at the position specified by the rectangles texture coordinates. A *chroma keying* [Bri99, SB96] criterion is used to segment the texture content into foreground and background pixels. In case of a foreground pixel, the texture color is returned. For a (green) background pixel, a transparency ray is traced further

into the background scene as if the billboard would not exist. Our billboard shader uses a simple principle component approach to chroma keying inspired by [vdBL99].

The video billboard shader is evaluated for each individual ray. This works not only for primary rays but also for arbitrary secondary rays like shadow or reflection rays. The rendering effects shown in Figure 3 thus need no additional rendering passes. Beside the normal shading call, OpenRT shaders support a second function that allows to speed up the handling of shadow rays by dropping unnecessary shading calculations.

The foreground color can be modulated by incident light in the shading process. This allows e.g. to implement a spotlight effect: the parts of the actor at the border of the light cone get darker. The top row images in Figure 4 illustrate the importance of the visual cues generated by the rendering effects.

The images in the bottom row of Figure 4 show the refractive effects that occur when the actor walks behind a glass sphere. These effects are hard to achieve when the billboard approach is used on raster graphics hardware.

Size and position of the billboard rectangle in the scene determine the available acting area. Automatic placement can be achieved using auxiliary camera views (see e.g. [GPT00]).

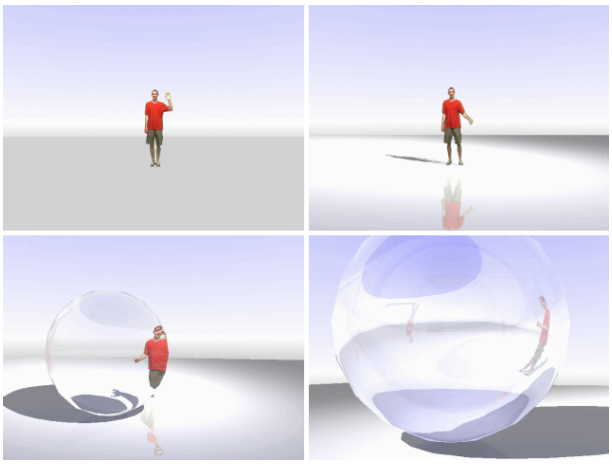

Figure 4: Rendering effects using a video billboard shader. Without visual cues like shadows and reflection, the actor seems to float (top left). The lower images show the actor walking behind a refractive glass sphere. The refraction is physically correct and hard to achieve with GPU based approaches.

Video billboards have a number of drawbacks due to their 2D nature, e.g. a mirror behind the actor would show the front view since there is no information of how the actor appears as seen from behind. Also there is no depth information available and the billboard rectangle limits the acting area. For some lighting angles the shadow cast from the billboard gets too small or disappears [Pom05].

These problems suggest to use a real 3D approach instead of the flat billboards.

3.2 A 3D approach

The billboard approach is appealing by its simplicity, but how can it be extended to 3D? The solution seems simple: extend the flat rectangle to a (3D) box and provide more than one view of the actor.

The shader associated with the box must implement a 3D reconstruction algorithm. Many approaches to 3D reconstruction methods have been published in the last years (see e.g. for a survey [SCMS01]. Most of the use *shape-from-silhouette* methods.

We opted for a modified *image-based visual hull* reconstruction algorithm [MBR+00] for our 3D reconstruction shader. The visual hull reconstruction principle can be described as intersecting the cones of the projected silhouettes of the actor. The view cameras need to be calibrated for a proper reprojection. The intersection set of the cones is called the *visual hull* [Lau94] of the actor and is a close approximation to the real shape, as long as enough views from different directions are used. Figure 5 illustrates the intersection process.

Rays that intersect the box are projected into the 2D view images and rasterized using a line drawing algorithm. Hence the method is called *image-based*. All operations are performed in 2D, which speeds up the computation. The methods suits also the needs of a ray tracing since the visual hull can be evaluated for single rays. We improved the method by [MBR+00] in several details to fit it into the (demand-driven) ray tracing framework.

A clipping process of the projected rays with the bounding box of the actors' silhouette accelerates the 2D intersection computation and allows the use of large acting areas without loss of performance. The acting area in the studio is directly related to the box object in the scene description. The visual hull shader also takes synthetic geometry inside the box object into account, which is important for exact occlusion

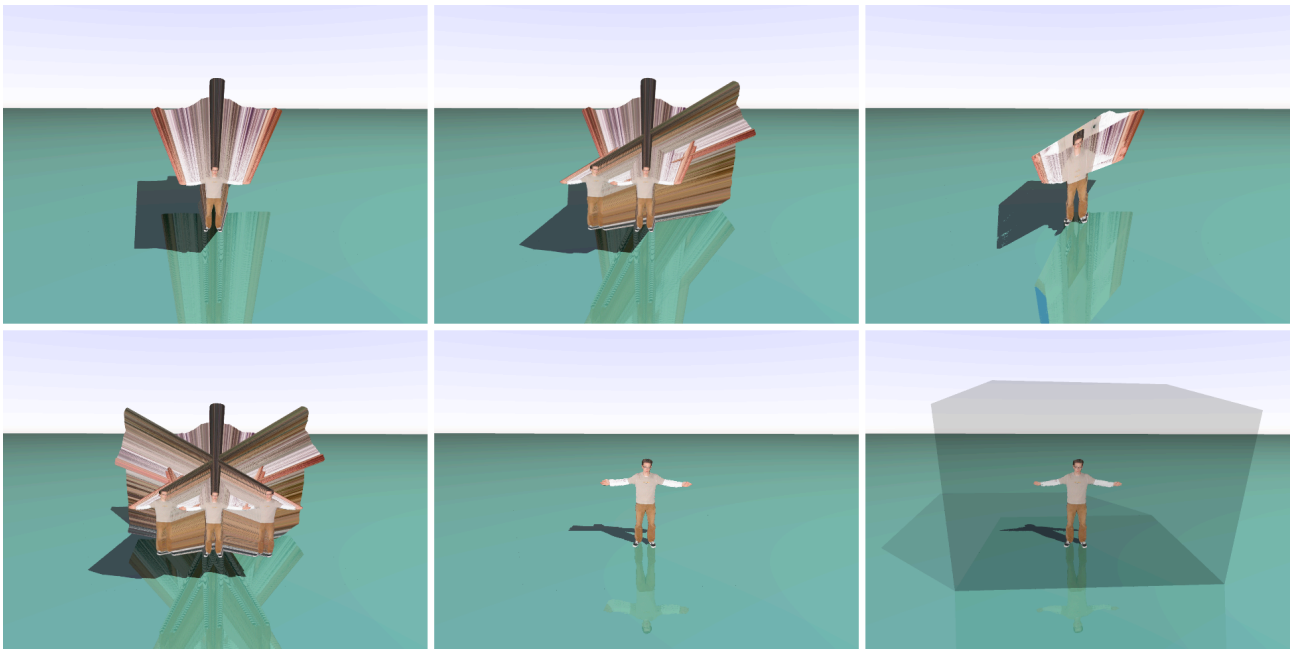

Figure 5: Illustration of the visual hull reconstruction algorithm. Two view cones of an actor are projected and intersected (top row). After adding a third view, the intersection tightly approximates the actor already (bottom mid). The bottom right image shows the box with the associated visual hull shader. Note, that visual cues (shadow and reflection) appear automatically.

effects (see Figure 6). A detailed discussion of the visual hull shader can be found in [PHS04, Pom05]. Note that it is also possible to implement visual hull reconstruction methods on graphics hardware (e.g. [LMS04]), but the restricted hardware resources make the integration with other geometry and multi-pass rendering methods for visual effects difficult.

To texture the visual hull of an actor, a *view-dependent texturing* method [DYB98] is used. Textures of the different views are blended to provide a smooth texture.

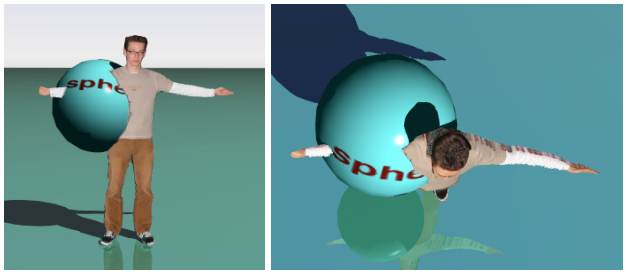

Figure 6: The visual hull shader achieves pixel-exact mutual occlusion between the real person and synthetic geometry.

Occlusion of a camera view by parts of the actor himself (e.g. by an arm) has to be taken into account when selecting the appropriate views for blending. This occlusion can be determined using the visual hull intersection method recursively and causes the main computational bottleneck of the visual hull shader [PHS04].

Multiple instances of the visual hull shader can be used to reconstruct different actors (Figure 7 right). Also a single shader can reconstruct several persons sharing the acting area as long as the visual hull reconstruction properties [Lau94] are not violated, i.e. all silhouettes can be distinguished in at least one view image.

Compared to the billboard approach, the visual hull shader allows the implementation of a real free-viewpoint virtual studio. The virtual camera(s) can arbitrarily move around in the virtual scene. Figure 7 shows an example scene.

Several other approaches to actor insertion based on 3-D reconstruction methods are published. They often employ volumetric methods and benefit from GPU hardware acceleration (e.g. [GPT04, HLS04]). The modular environment of a ray tracer and the presented visual hull shader for ray tracing provide an interest-

Figure 7: An example application using the image-based visual hull shader for seamless real 3D integration of a person into a synthetic scene. Note the correct reflection of the actor visible on the car side. The right image shows how three simultaneous instances of the visual hull shader can be used to reconstruct several persons independently. Nine camera views were used for reconstruction.

ing alternative to these methods and allow an easier and more straightforward integration with rendering effects and other AR/MR techniques.

3.3 Combination with AR Techniques

The described actor insertion methods can also be used to compose real actors over a real (live) video background. The rendering techniques are similar to those used with Augmented Reality applications [PS04]. *Differential rendering* methods [Deb98] allow for enhancing the result with visual cues like shadows and reflections of the inserted objects onto the real video background (Figure 8). For further possibilities and application examples please refer to [Pom05].

Scene	Figure	#Triangles	#CPUs	fps
TVRoom w/o lighting	-	7237	24	21
TVRoom w. lighting	2	7237	24	8.4
Billboards	3	424	16	15.2
Billboards	3	424	24	21.1
Billboards, glassball	4	824	16	12.7
VH shader w. car	7 left	208259	16	1.6
VH shader w. car	7 left	208259	24	3.3
3 VH shader w. car	7 right	208283	24	3.6
VH shader w. sphere	6	412	16	15.5
VH shader, closeup	-	-	16	2.1

Table 1: Achieved frame rates for the example applications for a varying number of CPUs at an output video resolution of 640x480 pixel.

4 Results

Resolution and frame rate of the video textures are limited only by the network bandwidth. The video delay is currently about 3–4 frames due to network latency. Due to the multicast approach, texture dropout caused by lost network packets can occur.

For this proof-of-concept, the visual hull shader is implemented using still photograph as input. Nine photographs, eight horizontally covering all sides, and one from the top, were used (Figure 9). The foreground was segmented manually. The camera pose and the calibration matrix was determined using [Bou04] and a checkerboard target.

The frame rates of the example scenes are given in Table 1. Rendering was done on dual Athlon MP 1800+ clients, connected via FastEthernet (100Mbit/s) to a central switch. The OpenRT server was connected by a Gigabit uplink.

Since rendering in a ray tracing framework is demand-driven, the frame rates vary with the current view. A closeup on the visual hull decreases the frame rate noticeably. The used OpenRT implementation was limited to a maximum of $\sim$21fps at a video resolution of 640x480 due to network bandwidth.

Note, that many of these limitations will soon vanish as high-end shared-memory PCs with up to 16 CPUs (8x dual-core) become available. Each of these CPUs has more than twice the compute power of our clients. New hardware architectures (like the Cell

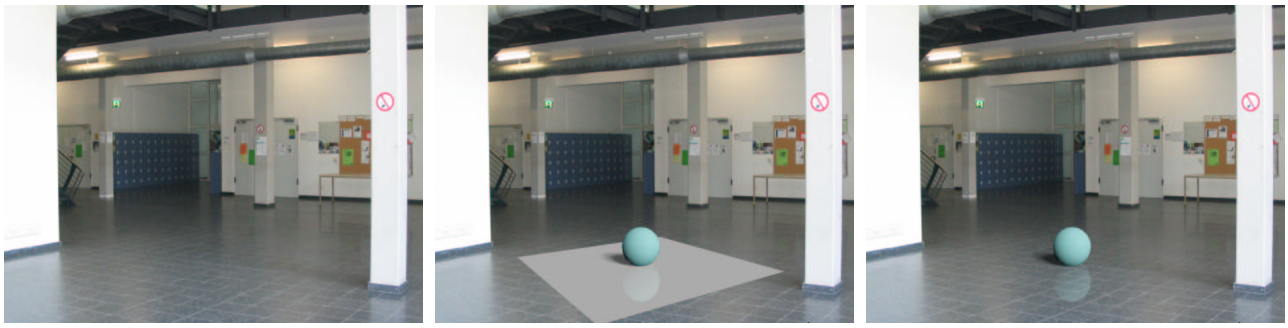

Figure 8: An in-shader differential rendering example. A sphere is rendered into a video background (left) using the AR view compositing method from [PMWS03, PS04]. Stand-in geometry (mid) is used to render the visual cues. The differential rendering method allows to blend the effects with the real background (right).

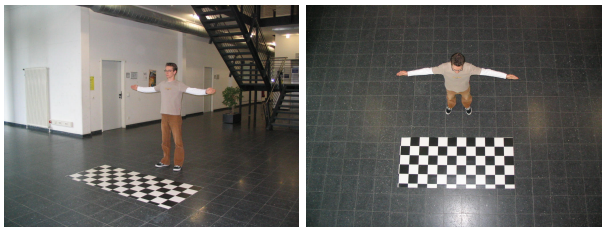

Figure 9: Two of the nine view photographs that were used as input for the visual hull shader. The checkerboard target is used for determining the camera calibration parameters. An image resolution of 1024x768 pixel was used.

processor) or dedicated ray tracing hardware [SWS02, WSS05] may also be used in the near future.

5 Conclusion

Though the results have to be seen still in an ongoing research context, they seem already very promising. The main drawback is still the rather slow frame rate which does not meet the hard real-time requirements of live virtual studio applications yet.

The above examples show the achievable rendering quality and the high amount of integration of the actors into the virtual scene using the in-shader compositing method.

The 3-D reconstruction method does not suffer from problems apparent with 2-D billboards. The concept of a free viewpoint virtual studio with real 3D actor reconstruction based on the presented visual hull shader can provide a large potential for future virtual studio system designs. Interactive ray tracing technology can provide here an alternative implementation platform with appealing benefits compared to complicated graphics hardware based systems (e.g. [HLS04]).

6 Future Work

For a full proof-of-concept of the in-shader visual hull reconstruction, we still need to set up a live multi-camera system in the future. A live system would also incorporate real-time background segmentation of the actors' views.

Since OpenRT also supports multi-threaded rendering on large shared-memory machines, it makes sense to migrate from the network setup to a single machine.

Virtual studio applications require a guaranteed, fixed output video frame rate. Further research is needed to modify the output-sensitive ray tracing based rendering methods to provide a constant output frame rate, e.g. by adapting the rendering quality.

References

[Bou04] Jean-Yves Bouguet, *Camera Calibration Toolbox for Matlab*, Caltech, US, 2004, http://www.vision.caltech.edu/bouguetj/calib_doc/.

[Bri99] Ron Brinkmann, *The Art and Science of Digital Compositing*, first ed., Morgan Kaufmann Publishers Inc., 1999, ISBN 0-121-33960-2.

[BWS03] Carsten Benthin, Ingo Wald, and Philipp Slusallek, *A Scalable Approach to Interactive Global Illumination*, Computer Graphics

Forum **22** (2003), no. 3, pp. 621–630, (Proceedings of Eurographics).

[BWS04] Carsten Benthin, Ingo Wald, and Philipp Slusallek, *Interactive Ray Tracing of Free-Form Surfaces*, Proceedings of the 3rd international conference on Computer graphics, virtual reality, visualisation and interaction in Africa, Afrigraph 2004, November 2004, ISBN 1-58113-863-6, pp. 99–106.

[Deb98] Paul Debevec, *Rendering Synthetic Objects Into Real Scenes: Bridging Traditional and Image-Based Graphics With Global Illumination and High Dynamic Range Photography*, Proceedings of SIGGRAPH 98, Computer Graphics Proceedings, Annual Conference Series, July 1998, ISBN 0-201-30988-2, pp. 189–198.

[DWBS03] Andreas Dietrich, Ingo Wald, Carsten Benthin, and Philipp Slusallek, *The OpenRT Application Programming Interface – Towards A Common API for Interactive Ray Tracing*, Proceedings of the 2003 OpenSG Symposium (Darmstadt, Germany), Eurographics Association, 2003, pp. 23–31.

[DWS04] Andreas Dietrich, Ingo Wald, and Philipp Slusallek, *Interactive Visualization of Exceptionally Complex Industrial Datasets* , ACM SIGGRAPH 2004, Sketches and Applications, August 2004.

[DYB98] Paul E. Debevec, Yizhou Yu, and George D. Borshukov, *Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping*, Proceeding of the EG Workshop Rendering Techniques '98, June 1998, pp. 105–116.

[Gla95] Andrew S. Glassner, *Principles of digital image synthesis*, first ed., vol. 1+2, Morgan Kaufmann Publishers, Inc., 1995, ISBN 1-558-60276-3.

[GPT00] Oliver Grau, Marc Price, and Graham A. Thomas, *Use of 3-D Techniques for Virtual Production*, Proceeding of SPIE Volume 4309 Videometrics an Optical Methods of 3D Shape Measurement, 2000, pp. 40–50.

[GPT04] Oliver Grau, Tim Pullen, and Graham A. Thomas, *A combined studio production system for 3-D capturing of live action and immersive actor feedback*, IEEE Transactions on Circuits and Systems for Video Technology (2004).

[Hav00] Vlastimil Havran, *Heuristic Ray Shooting Algorithms*, Ph.D. thesis, Faculty of Electrical Engineering, Czech Technical University of Prague, 2000.

[HLS04] Jean-Marc Hasenfratz, Marc Lapierre, and Francois Sillion, *A Real-Time System for Full Body Interaction with Virtual Worlds*, Eurographics Symposium on Virtual Environments EGVE, 2004, pp. 147–156.

[Kel00] Doug Kelly, *Digital compositing in depth: The only guide to post production for visual effects in film*, Coriolis Group Books, 2000, ISBN 1-57610-431-1.

[Lau94] Aldo Laurentini, *The Visual Hull Concept for Silhouette Based Image Understanding*, IEEE Transactions on Pattern Analysis and Machine Intelligence **16** (1994), no. 2, pp. 150–162.

[LMS04] Ming Li, Marcus Magnor, and Hans-Peter Seidel, *Hardware-Accelerated Rendering of Photo Hulls*, Computer Graphics Forum **23** (2004), no. 3, 635–642.

[MBR+00] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan, *Image-Based Visual Hulls*, Proceedings of the 27th Annual Conference on Computer Graphics, SIGGRAPH 2000 (Kurt Akeley, ed.), ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000, pp. 369–374.

[MFK+04] Gerd Marmitt, Heiko Friedrich, Andreas Kleer, Ingo Wald, and Philipp Slusallek, *Fast and Accurate Ray-Voxel Intersection Techniques for Iso-Surface Ray Tracing*, Proceedings of the Vision, Modeling, and Visualization Conference VMV, November 2004, ISBN 3-89839-058-0.

[MH99] Tomas Moeller and Eric Haines, *Real-Time Rendering*, first ed., AK Peters, Ltd., 1999, ISBN 1-568-81101-2.

[PBMH02] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan, *Ray Tracing on Programmable Graphics Hardware*, ACM Transactions on Graphics, 2002, pp. 703–712.

[PD84] Thomas Porter and Tom Duff, *Compositing digital images*, In Proceedings of the 11th Annual International Conference on Computer Graphics and Interactive Techniques, 1984, pp. 253–259.

[PHS04] Andreas Pomi, Simon Hoffmann, and Philipp Slusallek, *Interactive In-Shader Image-Based Visual Hull Reconstruction and Compositing of Actors in a Distributed Ray Tracing Framework*, 1. Workshop VR/AR, Chemnitz, Germany, September 2004, pp. 115–125.

[PMWS03] Andreas Pomi, Gerd Marmitt, Ingo Wald, and Philipp Slusallek, *Streaming Video Textures for Mixed Reality Applications in Interactive Ray Tracing Environments*, Proceedings of Virtual Reality, Modelling and Visualization (VMV), Akademische Verlagsgesellschaft Aka GmbH, Berlin, November 2003, pp. 261–269.

[Pom05] Andreas Pomi, *Interactive Mixed Reality Rendering in a Distributed Ray Tracing Framework*, Ph.D. thesis, Saarland University, Saarbrücken, Germany, July 2005.

[Poy03] Charles Poynton, *Digital Video and HDTV. Algorithms and Interfaces*, first ed., Morgan Kaufmann, 2003, ISBN 1-55860-792-7.

[PS04] Andreas Pomi and Philipp Slusallek, *Interactive Mixed Reality Rendering in a Distributed Ray Tracing Framework*, IEEE and ACM International Symposium on Mixed and Augmented Reality ISMAR 2004, Student Colloquium, November 2004.

[SB96] Alvy Ray Smith and James F. Blinn, *Blue screen matting*, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, ACM Press, 1996, pp. 259–268.

[SCMS01] Greg Slabaugh, W. Bruce Culbertson, Thomas Malzbender, and Ron Shafer, *A Survey of Methods for Volumetric Scene Reconstruction from Photographs*, International Workshop on Volume Graphics 2001, Stony Brook, New York (2001).

[Ste98] W. Richard Stevens, *UNIX Network Programming. Networking APIs: Sockets and XTI.*, second ed., Prentice-Hall, 1998, ISBN 0-13-490012-X.

[SWS02] Jörg Schmittler, Ingo Wald, and Philipp Slusallek, *SaarCOR – A Hardware Architecture for Ray Tracing*, Proceedings of the ACM SIGGRAPH / Eurographics Conference on Graphics Hardware, 2002, ISBN 1-58113-580-7, pp. 27–36.

[vdBL99] Frans van den Bergh and Vali Lalioti, *Software Chroma Keying in an Immersive Virtual Environment*, SAICSIT'99, Annual Conference, South African Institute of Computer Scientists and Information Technologies, 1999.

[Wal04] Ingo Wald, *Realtime Ray Tracing and Interactive Global Illumination*, Ph.D. thesis, Computer Graphics Group, Saarland University, April 2004.

[WD03] Ingo Wald and Tim Dahmen, *OpenRT User Manual*, Computer Graphics Group, Saarland University, 2003, http://www.openrt.de.

[WGDD98] Andrzej Wojdala, Marek Gruszewski, K. Dudkiewicz, and M. Donotek, *Real-time depth-of-field algorithm for virtual studio*, MGV (Machine Graphics and Vision) **7** (1998), no. 1/2, pp. 5–14.

[WGO00] Andrzej Wojdala, Marek Gruszewski, and Ryszard Olech, *Real-time shadow casting in virtual studio*, MGV (Machine Graphics and Vision) **9** (2000), no. 1/2, pp. 315–329.

[WPS+03] Ingo Wald, Timothy J. Purcell, Jörg Schmittler, Carsten Benthin, and Philipp Slusallek, *Realtime Ray Tracing and its use for Interactive Global Illumination*, Eurographics State of the Art Reports, 2003.

[WSS05] Sven Woop, Jörg Schmittler, and Philipp Slusallek, *RPU: A Programmable Ray Processing Unit for Realtime Ray Tracing*, Proceedings of SIGGRAPH '05, vol. 24, ACM Transactions on Graphics, no. 3, ACM Press, August 2005, pp. 434–444.

Citation

Andreas Pomi and Philipp Slusallek, *Interactive Ray Tracing for Virtual TV Studio Applications*, Journal of Virtual Reality and Broadcasting, 2(2005), no. 1, December 2005, urn:nbn:de:0009-6-2480, ISSN 1860-2037.