Joint Virtual Reality Conference of EuroVR - EGVE (2011)
S. Coquillart, A. Steed, and G. Welch (Editors)

Realistic Lighting Simulation for Interactive VR Applications

Alexander Löffler[1], Lukas Marsalek[2], Hilko Hoffmann[3], and Philipp Slusallek[1,2,3]

[1]Saarland University, Saarbrücken, Germany
[2]Intel Visual Computing Institute, Saarbrücken, Germany
[3]German Research Center for Artificial Intelligence GmbH (DFKI), Saarbrücken, Germany

Abstract

In the field of aircraft design, interior illumination increasingly becomes an important design element. Different illumination scenarios inside an aircraft cabin are considered to influence the mood of air passengers, help passengers to be better prepared for time lags and to create an overall positive environment. Consequently, a physically correct and realistic lighting simulation becomes essential during the design process. Available tools are producing videos or still images of illumination settings. The main reason for this is that realistic lighting simulation is believed to require heavy offline processing and unfeasible to do from within a real-time system. On the other hand, interactive Virtual Reality (VR) applications are an appropriate tool to experience an aircraft cabin under different illuminations. The ability to integrate lighting simulations into VR applications would simplify the design process remarkably by skipping time-consuming context and tool switches.
In this paper, we present a solution for integrating realistic lighting simulation with interactive performance into a single VR application. We explain our integration of real-time ray tracing, interactive global illumination, and measured point lights in a VR system, and its combination with classic rasterization techniques. We describe suitable interaction metaphors to enable realistic lighting simulation, high interactivity and intuitive interaction in an application for light design inside an aircraft cabin.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing, J.2 [Computer Applications]: Physical Sciences and Engineering—Aerospace

1. Introduction

In industrial contexts, Virtual Reality (VR) and respective virtual 3D models are commonly used for immersive review of construction and design in the product development phase. For construction reviews and inspections the inherent interactivity and immersion of VR systems and applications are the most important features whereas for design review an accurate material and surface representation is required. Especially the latter requirement is not fulfilled by current VR systems. Due to the strong need for interactivity, VR systems are optimized for rendering and interaction performance and far less for image quality. Furthermore, for regular review sessions users typically do not have the time for extensive data optimization. Data sets need to be available for VR-based examinations more or less instantly, so that a VR system must be able to handle more or less arbitrary data. It is commonly accepted that the lack of optimization costs performance, rendering and image quality. Consequently, VR systems are used for reviews of and decision making on geometric properties, shapes, visibility issues, clarity of car or aircraft interiors, etc., but not for decisions on materials, surface properties and colors. Nevertheless it is highly desirable to have VR systems that combine immersion, interactivity, and realistic representation of surface properties under certain lighting conditions.

Current VR systems are using the well-known rasterization algorithm [Shr09] for rendering. Rasterization is implemented in a highly optimized fashion on modern graphics hardware and delivers even with complex scenes real-time frame rates. On the other hand, rendering using the ray tracing algorithm produces acknowledged physically correct renderings but usually with way lower frame rates. Even if modern ray tracing renderers can achieve interactive performance by making use of all available hardware like SIMD

DOI: 10.2312/EGVE/JVRC11/001-008

CPU extensions, many-core highly parallel GPUs or even computing clusters, they are still too slow for real-time VR applications. Depending on scene complexity, typical frame rates in a ray tracer reach around 2–10 frames per second. For the targeted high image quality, rendering features such as soft shadows, global illumination with multiple light bounces, or antialiasing are necessary, which again lowers rendering performance such that frame rates may go down to less than one frame per second.

1.1. Related Work

A multitude of previous work has done some integration of advanced rendering algorithms in Virtual Reality systems. Schöffel has incorporated classical diffuse radiosity in a VR system to enable soft shadows with on-demand update at runtime [Sch97]. In order to create physically correct lighting, required for our work, the radiosity algorithm is however unsuitable. Dmitriev et al. used High Dynamic Range (HDR) environment mapping for Global Illumination calculated through a Pre-Computed Radiance Transfer (PRT) method, and combine the results in a CAVE setup [DAK*04]. PRT has limited applicability for dynamic light changes and hard shadows (e.g., as produced by spotlights), which both are required for our use cases.

Dietrich et al. used interactive ray tracing for the visualization of an entire aircraft geometry [DWS*06], but limit themselves to a navigational interaction through the scene, and do not offer graphical user interfaces (GUIs) or similar, which would require higher interactivity (i.e., frame rates) than a ray-tracing-only approach can currently provide. While the previously cited work is mostly targeted to specific application scenarios, Odom et al. investigated the general applicability of interactive ray tracing as an alternative rendering algorithm for driving Virtual Environments (VEs), which would combine realtime performance with physical correctness [OSR09]. Their results showed that using ray tracing in a distributed VR setup generally works, but is currently limited in terms of interactivity due to high delays and limited resolutions. This is why we chose a hybrid approach for our application scenarios.

1.2. Contributions

In this paper, we are presenting an approach to combining interactivity and physically correct light simulation in one VR application. We propose a rendering approach that combines rasterization and ray tracing in a hybrid setup. The rasterizer is intended to be used for all tasks that require interactivity, application handling, and scene manipulations, with the ray tracer as an on-demand add-on for producing the precise light simulation for the current area of interest, taking into account all direct and indirect lighting contributing to the currently rendered view. Users can switch to the ray tracer at any time, thus preserving actual view position, visible objects, as well as object properties.

We show the applicability of our hybrid approach in an application for interactive lighting design for the interior of an aircraft cabin. Light designers can actively position light sources, manipulate light properties, color profiles, etc. Furthermore, an interactive touch panel for cabin light operations is simulated to perform virtual training for cabin personnel.

On the ray-tracing end, we implement a highly accurate, yet interactive global illumination algorithm working on measured real-world luminaires, and high quality shaders for different cabin materials.

This paper is structured as follows: Section 2 explains the extensions done to our ray tracing engine in order to use standard IES light source profiles and apply a progressive global illumination algorithm to converge to the physically correct solution. In Section 3, we show how we combine this ray tracing engine with the rasterization renderer of our VR system. Section 4 then gives details on the VR interaction metaphors we provide to efficiently use such a hybrid system. An example application scenario making use of the developed technology is presented in Section 5. Section 6 summarizes results and provides an outlook to future work.

2. Light Simulation in a Realtime Ray Tracing Engine

In light design scenarios a faithful and accurate representation of the virtual scene is an essential requirement. It is necessary that the rendering system delivers results in such a quality that sound decisions about real-world appearance can be based on the virtual prototype. Of particular importance are the abilities 1) to capture as many light-object interactions as possible including indirect illumination, 2) to utilize well-specified realistic light sources, and 3) to easily incorporate advanced physically-based material models. Current rasterization engines–though delivering interactive speed with visually convincing results–cannot guarantee this predictive quality of the rendering, as the underlying algorithms often rely on approximations, simplifying assumptions, or restrictive pre-computation, which do not allow for solid estimates of the rendering accuracy.

Realtime ray tracing engines are principally much better equipped to deal with such scenarios as their properties can be more easily characterized with respect to the Rendering Equation [Kaj86] that models the physics of light transport. However, to achieve interactive rates, these engines also compromise the quality of the lighting simulation by considering only a subset of possible light-object interactions, typically limiting their scope to direct illumination with perfect reflection or refraction.

In our work, we have started off with state-of-art realtime ray tracing library RTfact [GS08] but extended it with a comprehensive light transport simulation based on progressive Monte-Carlo integration to be able to capture also indirect illumination, measured light sources to simulate real-world luminaries, and the AnySL compiler infrastructure [KRSH10] to allow for simple and efficient specification of physically-based materials. Our goal was to improve the rendering capabilities to allow for accurate simulation, yet stay interactive for applications in a VR context.

2.1. RTfact Real-Time Ray Tracing Library

RTfact [GS08] is a real-time ray tracing C++ template library that implements state of the art ray shooting algorithms. It delivers high performance, while still allowing for integration of advanced illumination simulation algorithms.

The key feature is that RTfact is architected not as a monolithic framework but rather as a library designed to simplify the building of custom ray tracing applications. The separation of the ray shooting and traversal algorithms from data structures allows us to implement the required support for simulation of indirect illumination, as described below.

2.2. Interactive Simulation of Indirect Illumination

In the scenario of the aircraft cabin we are primarily considering in this work (see Section 5) an indirect, multiple-times reflected light plays the major role in the illumination of the cabin interior. In such conditions an algorithm that would performs some form of light tracing, that is, follows particles from the light sources towards the scene, is absolutely necessary to reach acceptable performance and image quality. An algorithm based purely on path tracing [Kaj86] would be unsuitable, as the chances of hitting a light source with camera path, even with next event estimation are quite slim.

A bi-directional path tracer [LW93], though theoretically fitting, is however hard to parallelize within a high-performance packet ray tracing context, mainly due to low ray coherence. We have thus extended RTfact with Progressive Instant Radiosity [Kel97] enhanced with robust importance sampling [GS10]. Instant Radiosity (IR) is not to be confused with classic diffuse radiosity, but is an illumination simulation algorithm based on Monte-Carlo integration techniques. In a first phase, the algorithm traces light particles from the light sources and creates so-called Virtual Point Lights (VPLs) at the places where the particles hit a surface. As these particles are independent, they can be efficiently traced in parallel. In the main rendering phase the generated VPLs are used as conventional direct point light sources and thus their evaluation can be performed with the existing packet ray tracing infrastructure of RTfact.

Due to the complex illumination conditions, extending the system with an importance sampling is required to achieve a good performance. Without it, too many VPLs end up being irrelevant for the camera viewpoint, significantly increasing the rendering times. In our system we use the robust approach of Georgiev [GS10], adapted for use in parallel packet-based system. Still, to reach the desired rendering quality, several thousands or more VPLs have to be traced and evaluated, even including the importance sampling. This amount is however too high to stay interactive and thus a progressive approach is required. Each frame, we create only a small set of VPLs, which can be rendered interactively. Successive frames are then properly weighted and accumulated, continuously improving the quality and converging to a correct solution. So when the user is interacting with the scene, only a small number of VPLs is rendered, creating an approximation of the final solution. As soon as she stops, accumulation is automatically enabled and the image gradually improves towards a final correct solution within dozens of seconds.

We have also considered newer approaches to the many-lights algorithm, like Lightcuts, Matrix Row-Column Sampling, or Virtual Spherical Lights (VSLs) [WFA*05, HKWB09] but we have decided to use the Progressive Instant Radiosity with importance sampling and clamping reduction scheme for several reasons. First, though these methods do reduce certain artifacts caused by IR, they introduce another ad-hoc scene-dependent parameters (like VSL radius) that have not been properly investigated in the progressive setting. They also usually require extensive scene-dependent tuning, which is not desirable in the VR context (see Section 1). Moreover, the progressive variant of IR does remove the core disadvantage of pure IR, since the clamping is reduced over time and thus correct solution is achieved. Finally, it is as of yet unclear how to incorporate these advanced methods efficiently into a high-performance packet ray tracer, partly also due to their high memory requirements.

2.3. Realistic Luminaries

To assess the real-world feasibility of a given light design it is necessary to be able to perform the above described simulation with actual light sources that exhibit real-world characteristics, especially physically-measured emission and its angular distribution. Two main industry standards have been proposed to capture the characteristics of real-world luminaries, IESNA-95 (or newly IESNA-02) [Ame] and EULUMDAT [Lig]. In our system we have used the most popular and widespread IESNA-95 format.

IESNA-95 encodes luminaries as point light sources with measured non-homogeneous luminous intensity. This allows to create realistic non-uniform light sources like flashlights, street-lamps, emergency exit lights, etc. The intensity distribution of an IESNA-95 is described in an *.ies* file, which contains tabulated luminous intensity values sampled in several outgoing directions. Besides the emission characteristics, the format can also encode the geometry of the light fixture. Our system incorporates the IESNA-95 light sources as point light sources with automatic conversion of the photometric units to radiometric equivalents used by the illumination simulation. In the particle generation phase, we directly sample the converted tabulated data to get a radiance in a given direction. We have found out that linear interpolation in the tabulated data provides sufficiently accurate results.

2.4. Advanced Programmable Materials

Realistic behavior of materials plays a major role in the accuracy of the rendered images. RTfact library provides only a basic set of materials that are not suitable for highly-accurate rendering. Moreover, writing new materials is quite tedious, as it requires coding in a packet-based, templated C++ framework, where the shader author must manually deal with ray parallelization and correct SIMD packet utilization. To this end, we have used the AnySL compiler infrastructure [KRSH10], which extends RTfact with the possibility to write materials in RenderMan and other high-level languages. This allows us to use physically-based material models, but still keep high-performance due to the integrated SIMD vectorization infrastructure of AnySL.

3. Integration of Rasterization and Ray Tracing in VR

Our overall software architecture of the VR system integrated with ray tracing resembles the one suggested by Hoffmann et al. [HRL*09]. We discuss changes with respect to this original architecture below.

3.1. System Overview

As a basis for integration we use the VR system *Lightning* [BGB08], which is strongly modularized and implements a data flow concept between its independent modules. The native rendering back-end of Lightning is implemented on top of the OpenGL-based *OpenSceneGraph* [BO04]. We extend Lightning with two rendering back-ends for (a) direct rendering using the RTfact ray tracing engine and (b) distributed rendering using the DRONE distribution framework (formerly called URay) [RLRS09].

For combining multiple renderers in a single application, we implement a generic window module (*ltWindow*) in Lightning, which is able to display not only the results of a single rendering back-end, but those of many instantiated renderers one at a time. This generic window is implemented on top of OpenSceneGraph and displays the scene graph below a root *Switch* node, whose function is to limit the scene graph traversal to exactly one of its children at a time. If an OpenSceneGraph-based renderer is attached to the window, its internal scene graph is attached below the Switch node, in all other cases renderers directly write pixels to a full-screen textured quad attached as respective children below the root Switch.

To summarize, in the assembled architecture, hardware-accelerated rendering takes place in OpenSceneGraph (GPU-based) or in RTfact (CPU-based). Pixel generation of the latter might again be distributed on multiple physical machines through DRONE. The display of pixels generated by a local or distributed ray tracer is always handled by OpenSceneGraph. Figure 2 shows the assembled software stack again graphically.

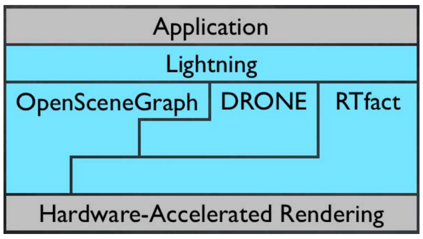

Figure 2: *Software stack of the assembled VR system.*

3.2. Synchronization

In order to switch seamlessly between different renderers processing the same scene, application state needs to be maintained at each physical host that takes part in the targeted VR setup. We perform synchronization on two levels: First, on the Lightning level, where the internal state of modules needs to be synchronized between different hosts, and second, on the DRONE level, where distributed rendering of parts of the same frame has to be performed. As synchronization inside DRONE is covered in detail in the respective work by Repplinger et al. [RLRS09], we will focus on the former kind here.

The Lightning *cluster protocol* [BGB08] allows to synchronize the state of single Lightning modules across network boundaries by serializing state structures and sending them from a master machine that receives and processes input to all slaves that need to react to changes. In our VR setup, we use this protocol to synchronize the state of the current view position and all dependent single cameras that make up the eye transformations (and their respective display surfaces) of one renderer. To synchronize the cameras in-between *all* renderers, we simply replicate any incoming input events changing the view position to all renderers attached to a window, which then perform the distribution to participating machines internally. Besides viewing information, we synchronize light transformations and properties between renderers using the very same mechanisms.

3.3. Scene Data Handling

In contrast to the original architecture [HRL*09], our software stack does not contain a dedicated scene graph for ray tracing. The reasons for this are threefold:

Firstly, it turned out that we do not need the fine-grained control over the exact scene hierarchy when we do ray tracing. On the contrary, flattening the inherent scene graph of

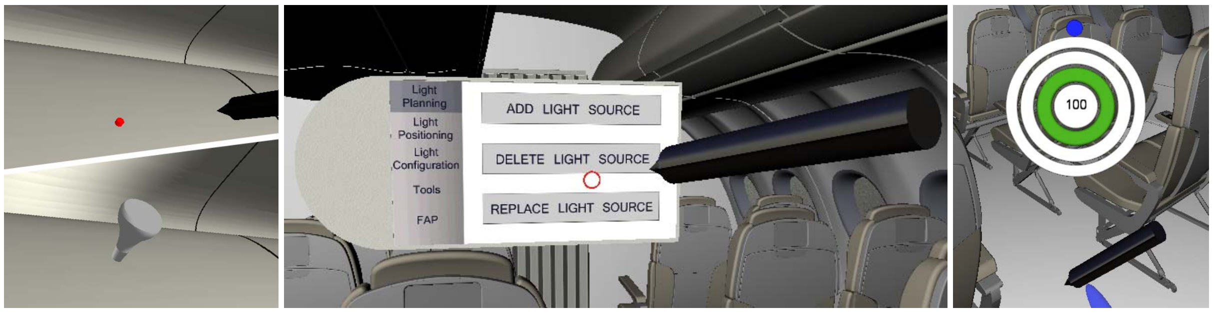

Figure 1: *Immersive user interfaces for object positioning and data input: (a) surface positioning before and after creating a light source; (b) 3D input panel with grasping region, tabs, and main interaction area; (c) number input wheel.*

a loaded scene file to nothing but a single triangle mesh before using it in the ray tracing engine turns out to yield a performance gain of 3-5x in frame rate, because the compute-intense traversal of unnecessary hierarchical data structures for each ray is completely omitted. As flattening results in an entirely static scene, we can choose to omit this optimization and thus enable dynamic objects as well.

Secondly, a flat scene structure perfectly matches the concept of the Lightning VR system, which incorporates the concept of *visual objects* representing a batch of geometry that is only to be manipulated as a whole. So if one needs geometry that is to be transformed differently than a second geometry, those two need to be separate visual objects in Lightning.

Finally, all the fine-grained control and manipulation in our application scenarios that would require access to the scene graph structure happens in the rasterization back-end anyways. In the OpenSceneGraph-based rendering back-end, the full scene graph can be accessed, sub-graphs transformed and searched, etc.. What we ultimately need in the ray tracing back-end is just the entire static scene geometry, very few dynamic visual objects, and all light sources to be manipulated separately.

4. Interaction Concepts

Interaction for our targeted light design scenarios focuses on the spatial manipulation of light sources, configuration of their parameters (e.g., color or intensity), travel within the scene, and switching between rasterization and ray tracing. All of these will be covered in the following subsections.

4.1. Facultative User Tracking

In VR, tracking of the user's input device and head transformation are very important prerequisites for efficient user interfaces and stereo vision, respectively. To have a user perceive an immediate reaction of the VR system to her tracking input, constraints apply to the time needed for completing a feedback loop between user input and system reaction. Due to the computational cost of ray tracing and the resulting frame rate of the overall system, our internal tests showed that using head tracking while operating the ray tracing back-end is not beneficial for the overall immersion of the user, or might even cause simulator sickness due to the perceived delays between head movement and adaptation of the field of view. As a result of these observations, we offer a full tracking of head and interaction device only in the rasterization back-end, where no perceivable delays occur for the user. The switch to the raytracing back-end results in a freeze of the current field of view. Navigation and head tracking is enabled whenever the switch back to rasterization is made. This very well fits our targeted workflows of first navigating and manipulating lights, then finding a desired viewing position (all in rasterization), and afterwards inspecting the results of the changed light situation under global illumination (in ray tracing). We use the tracked input device in rasterization to realize a simple travel technique using point-and-fly gestures, which are well-known in literature [BKLP04].

4.2. Light Source Positioning

A further important aspect of light design scenarios is the positioning of virtual light sources within the scene. Depending on the type of light source (point, spot, area, directional) the parameters position and orientation are available and can be changed during runtime.

In the rasterization back-end of the visualization, all mentioned light source types have *representation objects* to visualize the current spatial properties of the respective light source for the user (cf. Figure 1a). The most simple way of changing the position or orientation of a light source is available through *direct manipulation*; that is, by grasping and dragging the representation object of interest. This however turns out to be difficult whenever a user wants to position light sources precisely on surfaces, taking into account the exact surface geometry for position and normal orientation. To circumvent inaccuracies in these cases, we use the surface positioning technique proposed by Rentzos et al. [RPA*11].

Here, the user receives immediate feedback of how the surface is oriented, and the light source can be positioned accurately aligned to the surface and oriented towards its normal vector (cf. Figure 1a).

4.3. Alphanumeric Data Input

For specifying exact figures used as light source parameters, or selecting IES specification files from the filesystem, alphanumeric input is needed from within the VR application. Again, we use the concepts proposed by Rentzos et al. [RPA*11], namely the *3D input panel* and the *3D input wheel*.

The 3D input panel (cf. Figure 1b) is a flat piece of geometry, which can be directly manipulated by the user, that is, freely repositioned within the virtual world. Distinct sections on its surface separate the areas for grasping and moving the entire panel, for displaying the main user interface that features classic 2D widgets, and for switching between several tabs of that main user interface. The panel follows classic pen-and-tablet interaction techniques [BKLP04], with the difference of the tablet being a fully virtual object without any physical representation in the real world. We use the 3D input panel for direct input of illumination parameters, saving and loading of light profiles, and selection from within a set of IES specification files for assignment to single light sources.

The 3D input wheel (cf. Figure 1c) is a 3D user interface for the input of numeric data. The interface consists of three concentric rings that surround a number field located in the center. Each of the rings represents a different order of magnitude for changing the value of the central parameter. Whenever a user selects one of the rings, she is presented a blue guide marker orthogonal to the ring plane. Tilting the interaction device with respect to that guide enables her to add to or subtract from the value field, taking into account the order of magnitude of the change given by the currently selected ring. We use the 3D input wheel as a follow-up interface, which appears upon choosing to change a numeric light parameter in the 3D input panel.

5. Aircraft Cabin Use-Case

We combine all the above concepts of rendering and interaction in a single use case application that visualizes the interior of a passenger aircraft.

5.1. Data Preparation

Input data for the use case visualization is a partial model of a passenger aircraft cabin and IES light specifications for all the cabin lighting. The scene is exported from a DCC application in standard VRML 2.0. We could load the very same scene in rasterization and ray tracing, but as the scene has to be kept in memory twice anyways (once for each renderer), we use an enhanced scene for the ray tracer. To do so, we extend the VRML specification by custom entities to specify AnySL shader usage (e.g., brushed metal or glass) directly in the file (cf. Section 2.4). The extended VRML format is then interpreted by the RTfact scene loader.

In order to be able to interactively manipulate lights and their properties from within the VR application, they have to be dedicated objects in the Lightning system. Thus, as established in Section 3.3, lights may not be part of the loaded VRML file, because the entire file is treated as one inseparable entity. Instead, we export *light lists* directly from DCC. When Lightning loads these lists, the light specifications are translated to the respective interactive representations in the OpenSceneGraph- and RTfact-based rendering back-ends. Currently, in case of distributed rendering, both scene files and IES files have to be present on each machine performing a part of the ray tracing. Light lists are forwarded as scene setup events to each render node and then loaded in the present RTfact instance, identical to the local case.

5.2. Cabin Lighting

Besides any lighting that can be added by the user during runtime, the following light sources are placed in the scene from the beginning: six long area lights at the ceiling, four of them fitted into the openings above the luggage compartments, two in the rear cabin; 42 spotlights as reading lights, one above each seat; and a directional light on the outside of the cabin, simulating the sunlight entering through the windows. For performance reasons, we limit the overall number of actual OpenGL light sources in the scene, but have some of them available as representation objects only. In ray racing, however, all the lights are present and contribute to the GI simulation.

Each light, can be moved and configured through direct manipulation of their representation objects or the 3D input panel, respectively (cf. Sections 4.2 and 4.3). In addition to the controls on the 3D input panel, an additional interactive panel (Flight Attendant Panel, FAP) is attached to the aft cabin wall, in the position where it is also located in real-world counterparts of the aircraft cabin. It offers the same 2D user interface for light control as parts of the 3D input panel do, the FAP is however modeled after the real-world UI design and thus more suitable for virtual training of actual cabin personnel, who can then have a VR simulation of their real viewpoint when changing the light settings.

5.3. Rendering

The main input device is equipped with a dedicated button to switch from rasterization to ray tracing at any time the user likes to see the actual light conditions from the current point of view. Upon switching to ray tracing, as the application guarantees no more camera movement after the switch, the renderer directly starts accumulating the illumination across

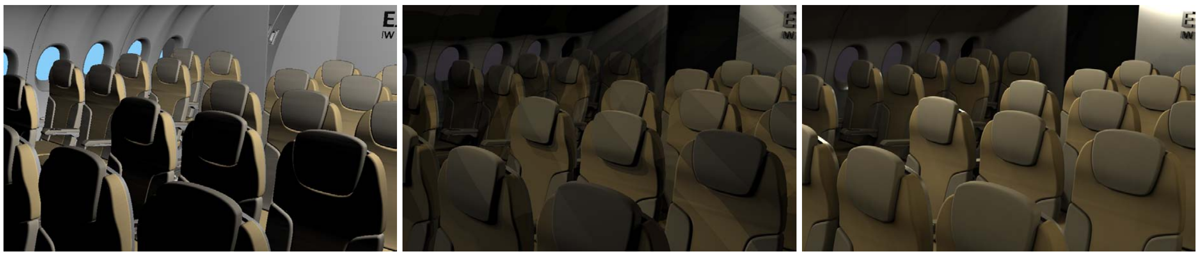

Figure 3: *Screenshots showing the aircraft cabin scene (a) in the rasterization back-end, (b) in the ray tracing back-end directly after the switch from rasterization, starting the accumulation, and (c) with global illumination after 15 seconds of convergence.*

all light sources and multiple frames using progressive refinement, at the same time descending deeper in the tree representing the hierarchy of reflected rays. Users are told in advance that a switch to ray tracing entails a disabling of any head tracking, such that major head movements would lead to a loss of immersion due to a wrong perspective with respect to their current point of view.

Over a short time of a few seconds (cf. Section 6) after switching to ray tracing, the image converges towards the physically correct solution. Figure 3 shows an example of the same field of view in rasterization and different stages of convergence when using ray tracing. The only interaction option a user has while in ray tracing mode, is switching back to interactive rasterization using the same button switch as before.

5.4. Workflow

The workflow applied in preliminary tests using this use case scenario is as follows: Users start in rasterization mode, where they are supposed to make themselves comfortable with navigation in the cabin interior. Afterwards, they are supposed to manipulate existing lights in color and intensity, first using the 2D Flight Attendant Panel attached to the cabin wall, then using the 3D input panel from anywhere within the scene. After each change, they may switch to ray tracing mode to examine the changes, and back to rasterization mode for further changes. Finally, they should create and position new light sources using the 3D input panel and, if desired, the surface positioning input technique.

6. Results

We implemented and tested the described system architecture and applications on a very moderate hardware setup consisting of two Intel Core 2 Quad 2.66 GHz workstations (one for each eye of a projective stereo setup) equipped with one NVIDIA GeForce 9600 GT GPU each. Each workstation was running Ubuntu Linux and driving a passive stereo projection screen with absolute pixel dimensions of 1920x1200. The aircraft scene we tested contained about 2.8 million partly textured triangles. In rasterization mode we reached frame rates of 15-20 fps, in ray tracing mode 2-5 fps depending on the geometric complexity visible from the current viewpoint. Waiting for a reasonably converged ray-traced image took around 10-15 seconds. Our informal user tests confirmed that such a setup is comfortable to work with and suits the workflow of interactivity and light simulation in rasterization and ray tracing, respectively. Having to switch renderers did not pose a problem for our users, although a formal user test with aircraft design specialists will follow to confirm these findings. Taking into account the very moderate specifications of the testing hardware, it can be assumed that much higher frame rates can be achieved with newest hardware or compute clusters for distributed ray tracing.

Overall we have shown that the combination of realtime ray tracing and rasterization can be suitable for a multitude of VR applications. Furthermore, we have proposed ways to incorporate measured IES light specifications used throughout the illumination industry can be directly employed in a ray tracer and, through our approach, used seamlessly within VR. Our approach presented here is not at all limited to aircraft design but is suitable for many applications such as car interior design, architecture, etc., as the resulting image quality and correctness of light simulation was considered very helpful during any tested design process. Through separate renderers, the interactivity for the light design process is still given. The additional time effort for data preparation is limited, there is no need for precomputing light scenarios to be used during the VR session, everything is simulated on the fly. The system is inherently scalable across multiple rendering and display hosts by using the DRONE framework [RLRS09], which also allows synchronization of multiple render hosts in terms of a consistent global illumination across all participating machines.

Limitations of our current approach are the required hardware resources, especially main memory, due to the fact that each rendering back-end currently holds its own version of the same scene to be displayed. Ray tracing in general still requires a lot of hardware performance, thus future work will include making even better use of hardware resources than RTfact already does. Our approach can be also implemented using any high-performance ray tracer and thus further tests with recent GPU-based real-time ray tracing libraries [PBD*10] are a natural extension to our work. Further improvements in rendering speed might be achieved by exploiting frame-to-frame coherence through incorporating for example the incremental approach of Laine et. al [LSK*07], which could be suitable for supporting head tracking directly in the ray tracing mode. We leave this for future work, as an important redesign of the ray tracing framework would be needed, due to a major role of shadow maps in the technique. Finally, our tests involved only simple projective VR setups, for surround-screen CAVE-like setups much more pixels need to be rendered per frame, and the required hardware will multiply; even though longer times to wait for a converged image might be an option for some applications. In the long run, with increasing hardware capabilities, a purely ray-traced yet affordable VR system might become feasible as well.

Acknowledgements

This research was made possible in part by the European Union's Seventh Framework Programme (FP7) for Research and Technological Development, project "VISION - Immersive Interface Technologies for Life-Cycle Human-Oriented Activities in Interactive Aircraft-Related Virtual Products", and the Intel Visual Computing Institute. Thanks to EADS Innovation Works Germany for providing the aircraft cabin scene and IES light profiles; thanks to the Laboratory for Manufacturing Systems and Automation of the University of Patras and the Interactive Media Systems group of the Vienna Technical University for the fruitful collaboration regarding 3D user interfaces.

References

[Ame] American National Standards Institute: IESNA LM-31-95 specification. http://webstore.ansi.org/RecordDetail.aspx?sku=IESNA+LM-31-95. 3

[BGB08] Bues M., Gleue T., Blach R.: Lightning - dataflow in motion. In *Proceedings of the First Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS@VR 2008)* (Mar 2008). 4

[BKLP04] Bowman D. A., Kruijff E., LaViola Jr. J. J., Poupyrev I.: *3D User Interfaces: Theory and Practice.* Addison-Wesley Longman, 2004. 5, 6

[BO04] Burns D., Osfield R.: OpenSceneGraph: Introduction, examples, and applications. In *VR '04: Proceedings of IEEE Virtual Reality 2004* (Mar 2004). 4

[DAK*04] Dmitriev K., Annen T., Krawczyk G., Myszkowski K., Seidel H.-P.: A CAVE system for interactive modelling of global illumination in car interior. In *VRST '04: Proc. of the ACM Symposium on Virtual Reality Software and Technology* (Nov 2004). 2

[DWS*06] Dietrich A., Wald I., Schmidt H., Sons K., Slusallek P.: Realtime ray tracing for advanced visualization in the aerospace industry. In *Proceedings of the 5th Paderborner Workshop Augmented & Virtual Reality* (Jun 2006). 2

[GS08] Georgiev I., Slusallek P.: RTfact: Generic concepts for flexible and high performance ray tracing. In *IEEE/Eurographics Symposium on Interactive Ray Tracing 2008* (Aug 2008). 2, 3

[GS10] Georgiev I., Slusallek P.: Simple and robust iterative importance sampling of virtual point lights. In *Proceedings of Eurographics 2010* (May 2010). 3

[HKWB09] Hašan M., Křivánek J., Walter B., Bala K.: Virtual spherical lights for many-light rendering of glossy scenes. *ACM Trans. Graph. 28* (December 2009), 143:1–143:6. 3

[HRL*09] Hoffmann H., Rubinstein D., Löffler A., Repplinger M., Slusallek P.: Integration of realtime ray tracing into interactive Virtual Reality systems. In *Proceedings of the 2nd Sino-German Workshop Virtual Reality & Augmented Reality in Industry* (Apr 2009). 4

[Kaj86] Kajiya J. T.: The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1986), SIGGRAPH '86, ACM, pp. 143–150. 2, 3

[Kel97] Keller A.: Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (Aug 1997). 3

[KRSH10] Karrenberg R., Rubinstein D., Slusallek P., Hack S.: AnySL: Efficient and portable shading for ray tracing. In *Proceedings of the Conference on High-Performance Graphics (HPG)* (Jun 2010). 3, 4

[Lig] Light Consult Inc.: EULUMDAT file format specification. http://www.helios32.com/Eulumdat.htm. 3

[LSK*07] Laine S., Saransaari H., Kontkanen J., Lehtinen J., Aila T.: Incremental instant radiosity for real-time indirect illumination. In *Proceedings of the Eurographics Symposium on Rendering* (Jul 2007). 8

[LW93] Lafortune E. P., Willems Y. D.: Bi-directional path tracing. In *Proceedings of the 3rd Int'l Conference On Computational Graphics And Visualization Techniques (Compugraphics '93* (1993), pp. 145–153. 3

[OSR09] Odom C. N., Shatty N. J., Reiners D.: Ray traced Virtual Reality. In *Advances in Visual Computing: 5th International Symposium, ISVC 2009* (Dec 2009). 2

[PBD*10] Parker S. G., Bigler J., Dietrich A., Friedrich H., Hoberock J., Luebke D., McAllister D., McGuire M., Morley K., Robison A., Stich M.: Optix: a general purpose ray tracing engine. *ACM Trans. Graph. 29* (July 2010), 66:1–66:13. 8

[RLRS09] Repplinger M., Löffler A., Rubinstein D., Slusallek P.: DRONE: a flexible framework for distributed rendering and display. In *Advances in Visual Computing: 5th International Symposium, ISVC 2009* (Dec 2009). 4, 7

[RPA*11] Rentzos L., Pintzos G., Alexopoulus K., Mavrikios D., Chryssolouris G.: Advanced interactions for immersive engineering applications. Submitted to Joint Virtual Reality Conference 2011, Sep 2011. 5, 6

[Sch97] Schöffel F.: Online radiosity in interactive Virtual Reality applications. In *VRST '97: Proc. of the ACM Symposium on Virtual Reality Software and Technology* (Sep 1997). 2

[Shr09] Shreiner D.: *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1.* Addison-Wesley Longman, 2009. 1

[WFA*05] Walter B., Fernandez S., Arbree A., Bala K., Donikian M., Greenberg D. P.: Lightcuts: a scalable approach to illumination. *ACM Trans. Graph. 24* (July 2005), 1098–1107. 3