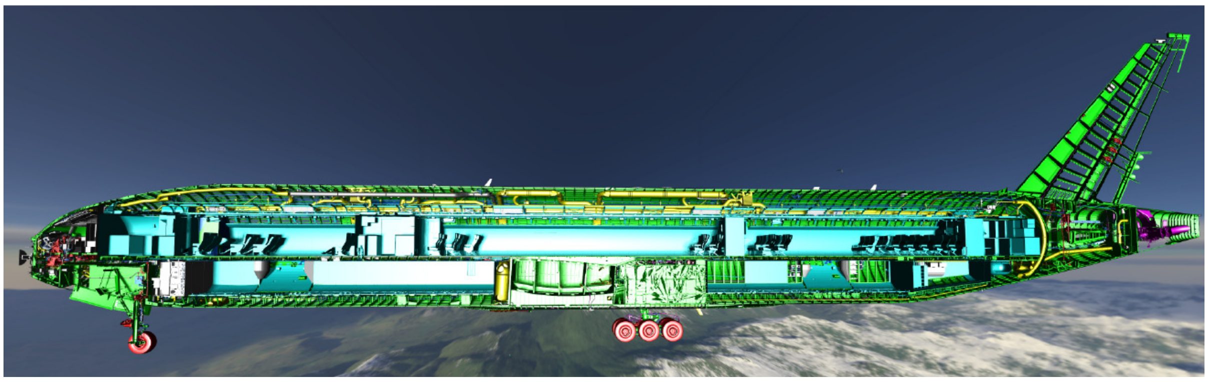
Realtime Ray Tracing for Advanced Visualization in the Aerospace Industry

Andreas Dietrich[*] *Ingo Wald*[+] *Holger Schmidt*[◊]
Kristian Sons[◊] *Philipp Slusallek*[*]



Abstract

One of the most pervasive problems in large-scale engineering projects is the difficulty in realistically visualizing models for evaluating the design and its visual appearance. The prohibitively high investment of using physical mockups has led to pre-assembly being performed almost entirely digital. Unfortunately, the vast complexity of full CAD datasets and the required realism can not be handled by available high-end graphics hardware. In this article we present a ray tracing based software system running on a scalable shared-memory architecture that allows for interactive high-quality visualization and evaluation of huge CAD models. Special features like cutting planes, model interrogation, sophisticated shading, lighting simulation previews, and collaborative remote visualization are also supported. The capabilities of our framework will be demonstrated using several practical examples from the collaborative design review of a Boeing 777 plus lighting visualization and evaluation of industrial design concepts from EADS.

1 Introduction

One of the most pervasive problems in large-scale engineering projects is the difficulty in properly fitting all individual parts together. The prohibitively high in-

[*] Computer Graphics Group, Saarland University, {dietrich, slusallek}@cs.uni-sb.de
[+] SCI Institute, University of Utah, wald@sci.utah.edu
[◊] EADS Corporate Research, {Holger.Schmidt, Kristian.Sons}@eads.net

vestment of using physical mockups has led to pre-assembly being performed almost entirely digital. Unfortunately, the vast complexity of full CAD datasets and the required realism can not be handled by available high-end graphics hardware. In this article we present a ray tracing based software system running on a scalable shared-memory architecture, which allows for interactive high-quality visualization and evaluation of huge CAD models.

For real-time display of highly complex models, *ray tracing* provides a highly interesting alternative. Ray tracing algorithms [Glassner 1989] closely model physical light transport by shooting rays into the virtual scene. This allows for accurately simulating the visual appearance and global optical effects including shadows, reflections, and others. By employing spatial index structures, ray-object intersections can be found efficiently, resulting in a logarithmic time complexity with respect to scene size. Additionally, because of the algorithm's *output sensitivity*, only data that is actually visible is eventually accessed.

Since the colors of different pixels can be calculated independently of each other, ray tracing offers an extremely high degree of parallelism. By assigning different pixels to different processing units, it is therefore possible to reach even real-time performance. This was first shown by Muuss [Muuss 1995] and Parker et al. [Parker 1999], who demonstrated interactive ray tracing using massively parallel shared-memory supercomputers. More recently Wald et al. [Wald 2003, Wald 2004a] have shown that interactive frame rates can also be achieved on clusters of low-cost commodity PCs. Although the use of PC clusters enables linear scaling in performance, memory scalability still remains a problem. Because every cluster node might potentially need to access the complete model, the scene database has to be replicated on each PC. For complex industrial CAD models of dozens or hundreds of gigabytes in size, this is not feasible. Special PC-based out-of-core variants for ray tracing massively complex models exist as well [Wald 2004b], but cannot yet deliver the performance and quality demanded by industrial application scenarios.

In this paper we present a ray tracing based interactive visualization system suited for the realistic display and design evaluation of extremely large CAD models *without* approximations, simplifications, or rendering artifacts. By efficiently combining a highly optimized ray tracing engine with a shared-memory multiprocessor architecture, it is possible to do real-time walkthroughs in large-scale highly detailed scenes, which is demonstrated at the example of a complete Boeing 777, consisting of more than 350 million individual polygons. Additionally, our system incorporates several features required for design review, such as distance measurement between arbitrary points, interactive identification and movement of individual model components, and sophisticated shading (including soft shadows, massive textures, and highlights) that may be programmed by the user.

With the help of the OpenGL Vizserver frame buffer streaming system or similar products, there is even the possibility to do the compute intensive image generation on a centralized visualization server, while the walkthrough can be controlled from lightweight remote clients, even over wide-area Internet connections.

The remainder of the paper is structured as follows: Section 2 starts with a brief overview over some existing massive model walkthrough systems. Section 3 will then provide some insight into our ray tracing software, the underlying shared-memory multiprocessor architecture, and the remote visualization features of the system. We demonstrate two capabilities and features in two application scenarios: The visualization of an entire Boeing 777 aircraft in Section 4 and high-quality visualization of aircraft cabins and helicopters at EADS in Section 5. We conclude and discuss future extensions in Section 6.

2 Related Work

Due to its practical relevance, the problem of realistically visualizing massively complex models has received a lot of attention, which we will briefly discuss.

2.1 Rasterization Based Systems

The UNC GigaWalk system [Baxter2002] runs on an SGI Onyx workstation (300 MHz MIPS R12000 CPUs, 16 GByte RAM) with Infinite Reality graphics, and makes use of two rasterization pipes and three processes running in parallel on individual CPUs. The visible geometry of each frame is treated as potential occluders for successive frames. Using occlusion culling based on these occluders in combination with a Hierarchical Z-Buffer [Greene 1993] the system is reported to render scenes with up to 82 million triangles at 11-50 frames per second.

Another recently proposed framework is iWalk [Correa 2003]. It can handle models consisting of up to 13 million triangles at 9 frames per second on a single commodity PC (2.8 GHz Intel Pentium 4 CPU, 512 MByte RAM) with an NVIDIA Quadro 980 XGL card. However, the system relies on approximated visibility, and uses an object-space algorithm [Klosowski 2000] to estimate a potentially visible geometry set, which can result in visible polygons being omitted.

In contrast to the above mentioned applications that are primarily meant for visualization only, the Boeing FlyThru [Arbarbanel 1996] system, a proprietary in-house application originally conceived for the 777 twin-engine airliner program (see Section 4) comprises a great number of features aiding collaborative CAD. Apart from displaying thousands of parts at one time, it facilitates detection of motion anomalies and interference between structures, interactive design reviews across a network, modeling, kinematics, and remote control by other applications.

Unfortunately, no detailed information about its interactive rendering capabilities is available. It can, however, not display the full 777 dataset at real-time rates without geometric simplifications [Kasik 2005].

None of these systems provides realistically complex shading or global lighting effects like shadows or accurate reflections necessary for faithfully evaluating a virtual model as possible with current physical mockups.

2.2 Ray Tracing Based Systems

Ray tracing technology efficiently supports interactive visualization of large non-simplified datasets. The OpenRT real-time ray tracing engine [Wald 2003, Wald 2004a] has been shown to be capable of handling scenes with up to several million triangles in real-time. On a setup of 24 commodity dual-processor PCs (AMD AthlonMP 1800+ CPUs, 512 MByte RAM) this system has been reported to achieve up to 23 frames per second. Additionally, it incorporates physically correct and global lighting simulations [Benthin 2003] and features interactive placement of geometric parts. It relies, however, on the fact that each cluster node can keep the complete scene in main memory.

Wald et al. [Wald 2001] have also presented an out-of-core rendering variant of the OpenRT system that combines explicit memory management, demand-loading of missing parts, and computation reordering. While this system has been shown to render scenes that are much larger than main memory it can only handle scenes where only a small fraction of data has to be loaded between successive frames, and does not easily scale to scenes of a more realistic complexity.

In a more recent publication [Wald 2004b] it was demonstrated that even on a single desktop PC (dual 1.8 GHz AMD Opteron 246, 6 GByte RAM), out-of-core ray tracing can be used for interactively visualizing a complete Boeing 777 CAD dataset containing more than 350 million individual surface polygons. Even including the calculation of pixel-accurate shadows and highlights, the system reaches up to 5 frames per second. Due to the out-of-core nature of the approach, model parts are only loaded on demand, and — as not all missing data can be loaded within the same frame — an approximation scheme has to be employed to represent data not loaded yet. This frequently leads to rendering artifacts that are not tolerable for practical applications. Additionally, the framework does not easily parallelize due to the need to synchronize all memory operations on all client machines, and thus cannot deliver sufficient performance.[1]

[1] Design reviews are usually considered to require 10-20 frames per second.

3 Visualization System Outline

The presented rendering architecture basically builds on the system of Wald et al. [Wald 2004b] with OpenRT as ray tracing core. The rendering artifacts introduced through the out-of-core mechanism required on a PC platform made this system not applicable for practical applications. In contrast, the eventual end users of our visualization system explicitly demanded display of an entire complex dataset at *any time*, without *any* kind of approximations, demand-loading stalls, or rendering artifacts.

To meet these demands, it was decided to port the initial system to a scalable shared-memory multiprocessor architecture, and thereby couple the performance scalability of the OpenRT system with the memory scalability of this platform.

3.1 Hardware Architecture

Our early experiments were conducted on an SGI Altix 350 mid-range server [SGI Altix 2004] composed of 8 dual-processor nodes. Each of the nodes is equipped with two Intel Itanium 2 CPUs clocked at 1.4 GHz, and contains 4 GByte local memory. In this setup, the memory banks of the nodes form a system-wide 32 GByte large, shared-memory address space. A fast interconnect provides a low-latency, high-bandwidth interconnect between the distinct nodes, gaining peak transfer rates of up to 6.4 GByte per second. As this is completely transparent to the application, each CPU can directly access every desired part of the model in the global memory space. The geometric database is actually distributed over the nodes' physical memory banks without any replication of data.

Similar shared memory systems based on commodity PC technology have recently become available including multi-processor AMD Opteron servers with up to 8 dual-core processor (16 CPUs) and up to 128 GB of memory connected by fast Hypertransport links.

3.2 OpenRT

The OpenRT real-time ray tracing core [Wald 2004a] serves as a high-performance rendering back-end for our 3D CAD browser application. It supports physically correct lighting simulation, plug-and-play shading by means of dynamically loaded shader libraries (i.e. custom programs that perform the actual light propagation calculations), and handling of dynamic and complex 3D environments.

Highly optimized code and distributing computation among several CPUs working in parallel allows the OpenRT engine to reach interactive and even real-time frame rates. In this client-server approach a single master process centrally manages a number of client processes: The image is decomposed into a number of disjoint regions that are asynchronously assigned as tasks to the clients *on demand*. After a client has finished computation of an assigned image-tile, it sends back the respective pixel color values to the master, which composes them into the resulting image. Although, the system has been specifically designed to run on a cluster of PCs, the setup on the shared-memory systems is practically the same. All client processes are started on the same machine as the master process, while the operating system takes care of distributing the processes among the available CPUs.

Memory management of large CAD databases can be done in a very straightforward manner. Since the shared memory systems provide enough RAM to keep the full model in memory, the whole dataset (including all spatial index structures) it simply mapped from disk into the global address space, using Linux memory mapping facilities. This can be independently done by each client process because the operating system takes care that no part is paged into physical memory more than once. Although OpenRT incorporates a memory management subsystem that can deal with scenes larger than main memory in an out-of-core fashion (see [Wald 2004b] for details), this is not required here.

3.3 Remote Visualization

For the purpose of collaborative design reviews, the system can also act as a centralized visual server for multiple clients in geographically diverse locations. To this end the system make use of the OpenGL Vizserver [Vizserver 2004] or similar technology: A frame rendered on the visualization server is captured and the compressed pixel data is sent to the clients over standard local as well as wide-area networks. Each client then decompresses the pixel stream, displays the uncompressed image, and directs back all user interaction to the server. As only the final image is transmitted, the clients themselves do not need any high-performance graphics capabilities at all, and can thus be lightweight clients such as desktop PCs or laptops.

Although a simple video streaming approach could have been more efficient, especially since we do not require support for hardware accelerated rendering, we opted for the Vizerserver because it provides a stable, mature, and widely used industrial remote visualization framework.

4 Application: Design Review of a Complete Airplane

One of the main objectives we targeted was the interactive walkthrough of a fully-detailed 3D model of a Boeing 777 twin-engine airplane. There should not only be the possibility to directly render every single part of the original CAD data without any kind of geometric simplification, visual approximation, or artifacts. The system should also be suited for engineering design review sessions.

4.1 The Boeing 777 Model

The Boeing 777 model used in our experiments results from a direct export of the original construction CAD data out of the CATIA CAD/CAM system. Although some components are missing, the model already consists of more than 350 million individual surface triangles. Organized in over 13,000 compressed files, all components, including cables, screws, valves etc., have been modeled at extremely high accuracy. Without any additional spatial index structures, the raw model requires more than 12 GByte of hard disk space. Because the polygons were provided without any mesh connectivity information (i.e. coming as a "soup of triangles"), and with all vertices being randomly displaced to prevent data theft, the model is difficult to handle for surface simplification algorithms found in most large model rendering systems.

4.2 Visualization Workflow

For the purpose of efficient model access during ray traversal calculations, spatial index structures are needed in addition to the geometric triangular surface information. In a first step, the original files are decompressed, parsed, and transformed into an unordered triangle stream. This stream gets then sorted into a k-d tree [Bentley 1975], which is stored in binary form. Like the ray tracing engine, the preprocessing tool chain can make use of multiple processors. Thus, it is able to preprocess the entire data in a parallel manner during less than 2 hours.

Including all additional index data, the resulting binary data files cover roughly 20 GByte of hard disk space. Since the files fit completely into main memory, they can then be copied into a RAM disk, from where they are directly mapped into main memory. This enables the ray tracing engine to virtually start in an instant, and to provide the first images after less than 10 seconds.

Upon startup of the 3D browser application all the binary files are mapped into the global shared memory space and are therefore immediately visible in the address space of each client process. A user can now freely browse the fully-detailed model without having to wait for data being fetched from disk and without encountering visual artifacts caused by not yet loaded data.

4.3 Design Review Functionality

The prime requisite for our system to be useful for design reviews is to deliver high-quality real-time rendering performance. In particular, these goals were specified by the users as: (i) The system should achieve at least 10 frames per second at a resolution of 640 x 480 pixels, even for complex views and during interaction. (ii) It should not generate *any* visual artifacts during rendering – especially it should not generate any approximate views (as done in [Wald2004b]). (iii) It should have maximum startup times of only a few seconds. Using the aforementioned visualization system, where the distributed ray tracing engine delivers real-time performance, and the global shared memory system allows for keeping the entire model data in main memory, these demands can be fulfilled.

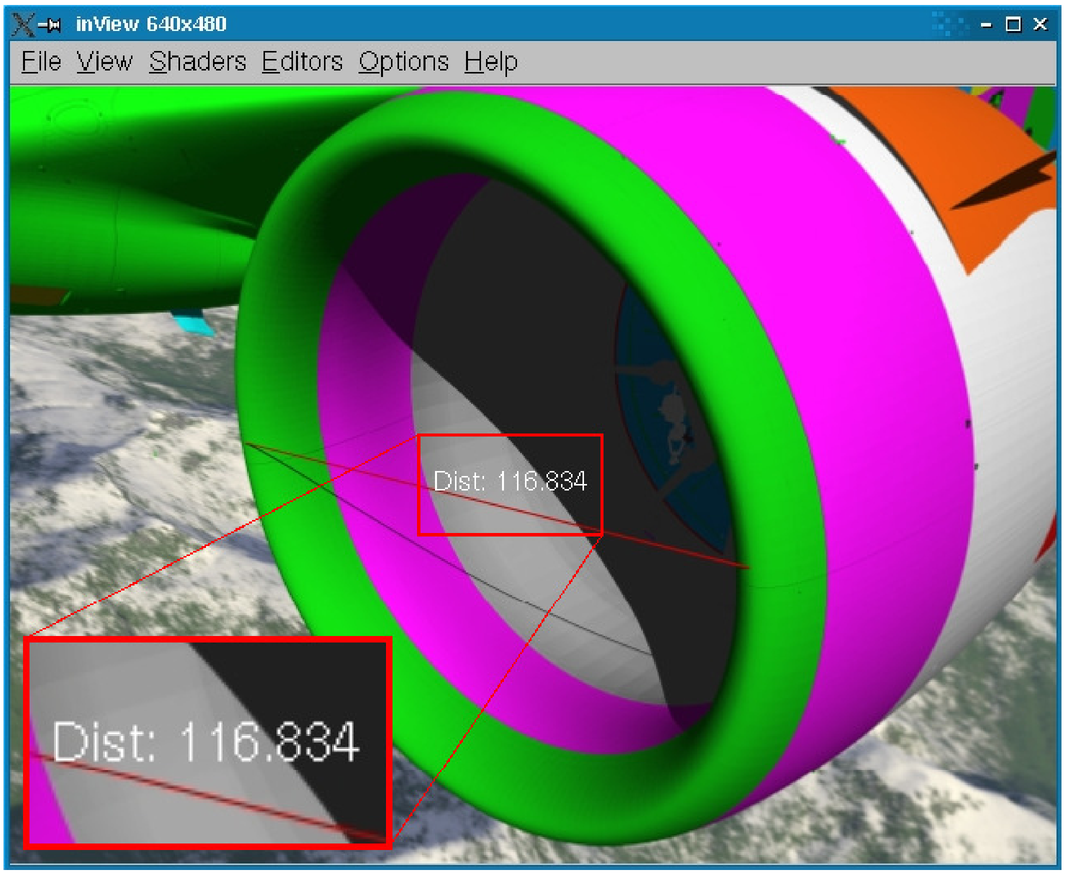


Figure 1: Interactively measuring the diameter of a Boeing 777.

Apart from the capability of interactively displaying arbitrary parts of the Boeing 777 model, our visualization framework offers a number of additional functions required for collaborative CAD evaluation.

A very important feature that eases fitting together a model's components, is the ability to measure the exact distance between arbitrary points in the dataset. The user simply has to click at two different points in the browser window. By shooting rays from the projection center through the respective pixels into the scene, the ray tracer can easily find the distance between the visible surface points and the observer. The application can optionally insert a line object into the scene that helps visualizing the connection between the two points in question. The distance value is also shown besides the line (see Figure 1). Because this line object (actually a slim box) behaves like any other geometric object, it too can cast shadows that provide important visual cues on the exact location of that measuring line.

By applying the same technique, i.e. firing a ray through the pixel the mouse pointer is currently hovering above, not only the distance to a surface point can be determined. Because the ray tracing core can provide the front-end application with an identification of the object being hit by the ray, arbitrary information regarding this part can be looked up and displayed (see Figure 2). Little application-specific code (except for display) was required for that feature, as the usually complex picking-operation could easily be realized by using the ray tracer.

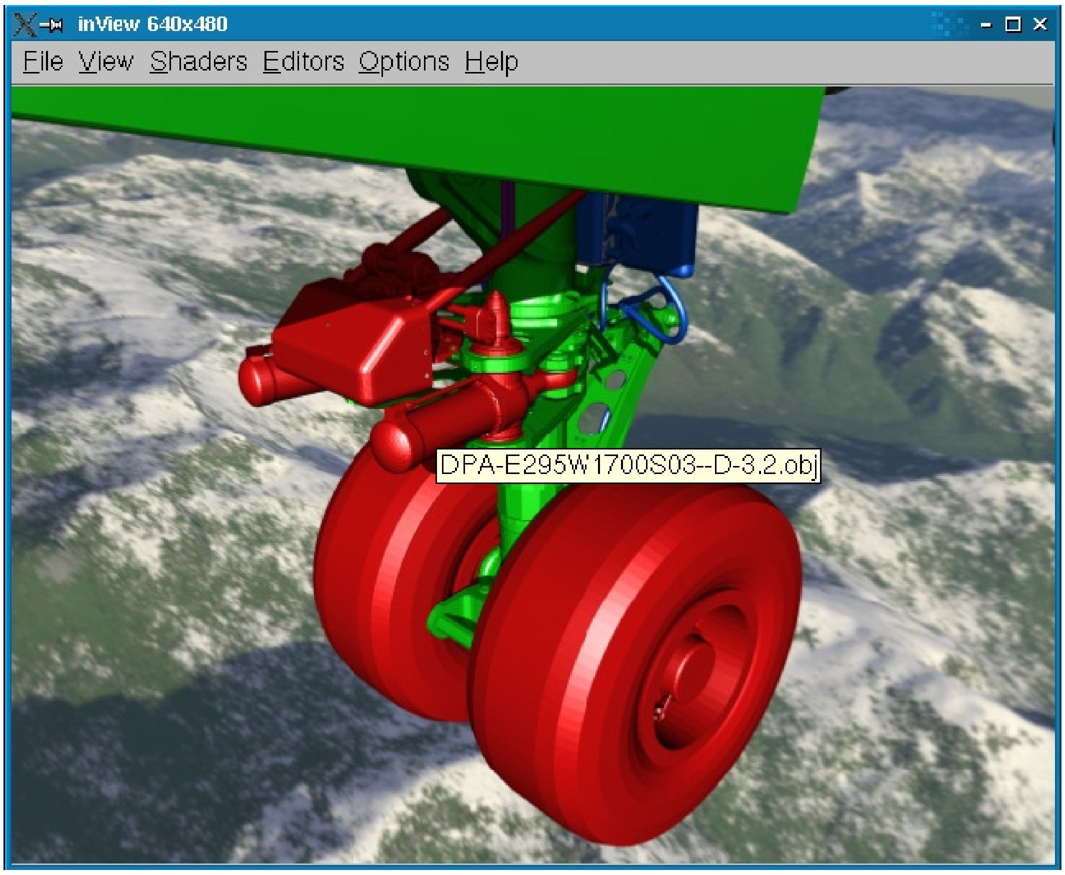

inView 640x480
File View Shaders Editors Options Help
DPA-E295W1700S03--D-3.2.obj

Figure 2: All components can be pixel-accurately identified by simply moving the mouse pointer over them.

4.4 Cutting Planes

One of the advanced features of the ray tracing core is its ability to instantly cut away large parts of a model by specifying a number of freely orientable clipping planes. This works most efficiently for a ray tracer since it simply has to clip rays, whereas rasterization techniques need to clip all potentially visible polygons. Although inserting a cutting plane can completely change the set of visible triangles, this is not a problem at all for our system: Since all scene data completely resides in memory, even such a drastic change of the visible set does not introduce any loading cost.

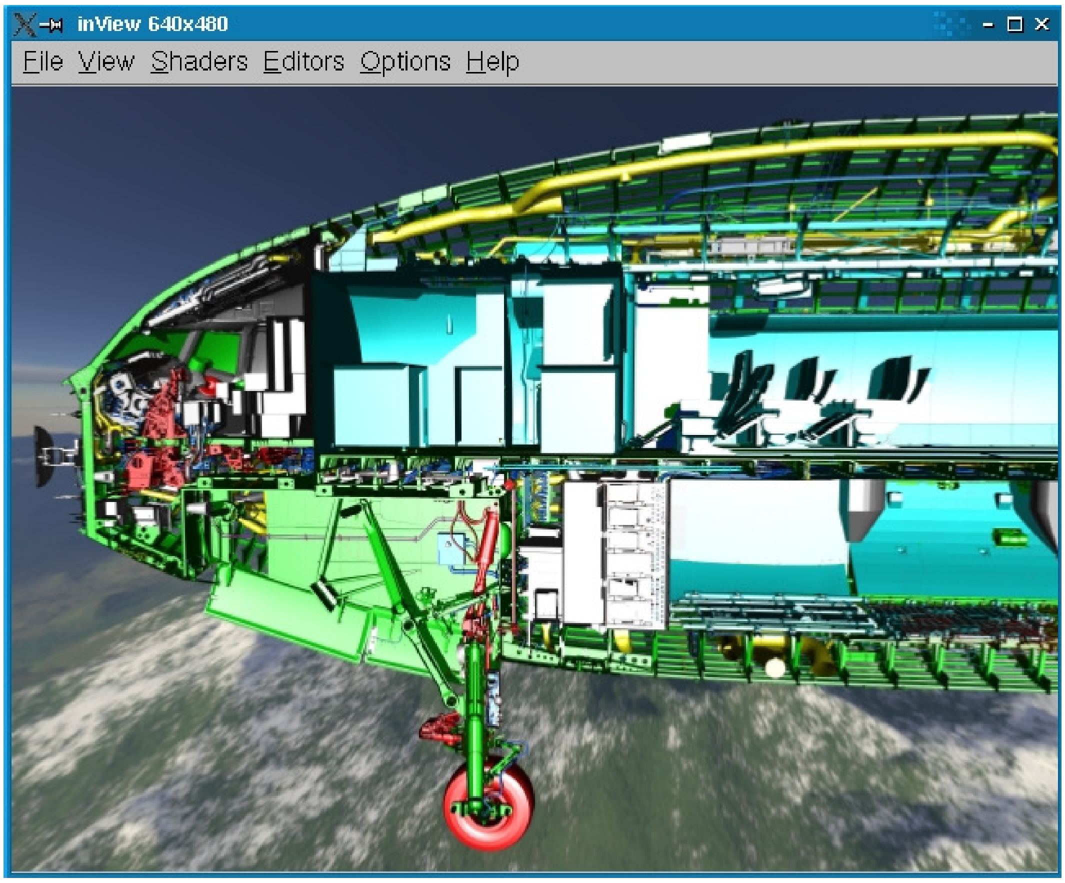


Figure 3: An interactively placed axis-aligned cutting plane slicing the airplane in half. The resulting cross-section view gives a much better insight into the model's overall structure. Multiple freely orientable cutting planes can be placed interactively by the user.

Cutting planes are particularly useful for structural analysis. For example, they easily allow for producing cross-sectional views (see Figure 3) that may, for example, serve as technical illustrations. Note that these cutting planes do not simply cut away the geometry, but can be configured to only affect viewing rays, therefore making it possible to look into the airplane without influencing e.g. the shadow computations inside.

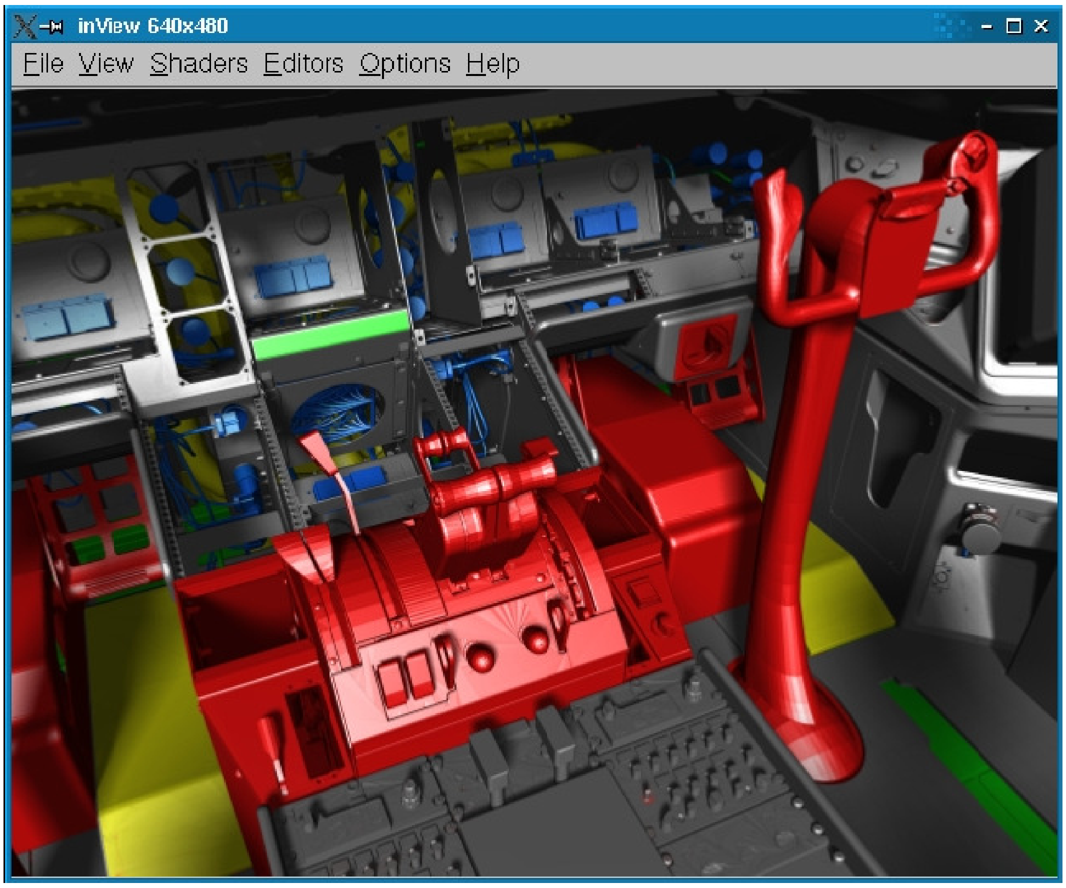


Figure 4: Soft shadow effects in the cockpit providing a better impression of the relative placement of components.

Due to accurate simulation of physical light transport, sophisticated shading and lighting (e.g. pixel-accurate shadows, highlights, or reflections off of curved surfaces) can easily be incorporated in a plug-and-play fashion. In particular for complex geometry, the projection of 3D data onto a 2D display often incurs an undesired loss of depth impression. In that case, shadows often significantly help in the perception of the relative position of objects. Especially *soft* shadows help in judging the distance between shadow caster and receiver. Figure 4 shows the impact of rendering soft shadows that can significantly enhance the impression of shape and depth (see also Figure 1).

5 Application: Advanced Visualization at EADS

A similar project evaluating realtime ray tracing is under ways also at the EADS corporate research center (CRC) as part of several internally and externally funded research projects in the context of virtual and augmented reality. EADS was particularly interested in the transfer and evaluation of new technologies for use within different corporate EADS departments. The main focus for realtime ray tracing is its use for design reviews using high-quality visualizations of large data sets. In addition realtime ray tracing has been integrated with a tracking system and split-screen approach for stereo display. For a number of different internal purposes the ray tracing system has also been extended by advanced shaders. Two example applications from this use at EADS are discussed below.

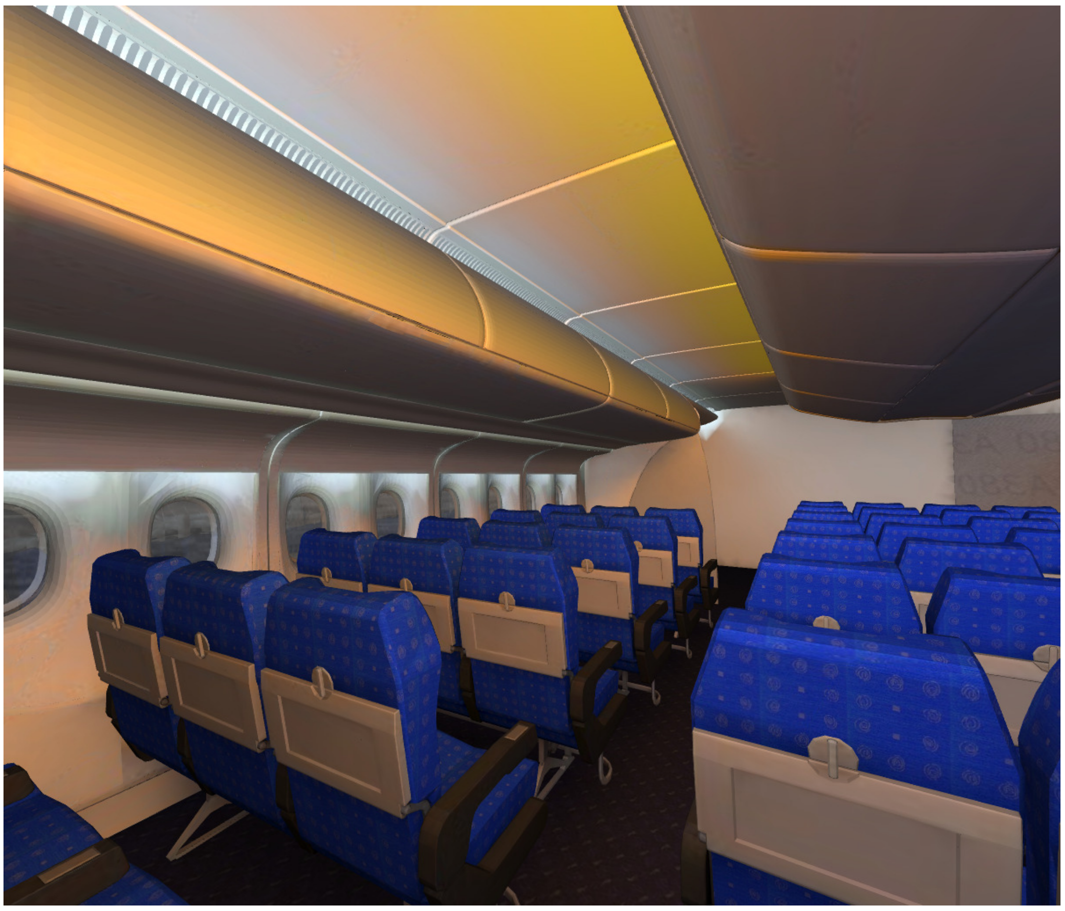


Figure 5: Low resolution geometric model with results from high-quality lighting simulation. More than 600 MB of texture memory are required to visualize these results interactively.

5.1 Visualization of Lighting Scenarios in an Aircraft Cabin

Significant research results already existed for the simulation of static lighting scenarios. However, previous rendering technologies posed several problems including the low performance for large data sets (beyond 10 million polygons) and the limited texture memory on traditional graphics cards. The results from the lighting simulation are represented in large textures, which require extremely high resolution that increases with the model complexity and the accuracy of the results. Even for scenes with a low geometric complexity (see Figure 5) the required amount of texture memory easily exceeds 600 MB and more.

A software based realtime ray tracing system is not restricted by the limited texture memory on a graphics card as it can directly use the textures in main memory. It is also not necessary to use texture compression with its danger of introducing artifacts. In addition ray tracing allows for combining the diffuse results from lighting simulations with other optical effects, such as mirror reflections in windows, metallic surfaces, or other specular objects of arbitrary shape. Ray tracing guarantees an accurate visualization while allowing simple and highly flexible realization of nearly arbitrary lighting models.

This ongoing project of combining the results of a static lighting simulation with realtime ray tracing is targeted to be used later by engineers, designers, and customers for evaluating the simulated lighting situation in different cabins directly in a high-performance virtual reality visualization environment.

5.2 Visualization of Different Paint Designs on a Helicopter

Helicopter often require a customer specific design that needs to be visualized with optimal image quality and high geometric accuracy based on a large number of polygons to capture the smoothly curved surfaces. Again detailed textures are used to capture the design (see Figure 5) and realtime ray tracing provides the physically realistic appearance including omnipresent reflections for external views. The system is tightly integrated with the CAD system, retaining during data export all the geometric details including even individual screws on the exterior (see lower right image in Figure 6).

Custom shaders are used to capture the special paint and glass surfaces accurately for achieving the necessary realism. These shaders are combined with high-dynamic-range environment textures in order to integrate the helicopter smoothly into the surrounding scene. This combination of appearance simulations with high-quality reflections of the environment greatly improves the visual quality of VR presentations including stereo projection and tracked head-mounted displays.

The targeted use within EADS is the evaluation of paint designs as well as the detection of error when applying the designs. The visualization quality allows for virtually presenting the design to a customer in a highly realistic environment even before the helicopter is physically built. As shown in Figure 6 the visual quality of the interactive VR presentation is far beyond that achieved with traditional rasterization based rendering technology.

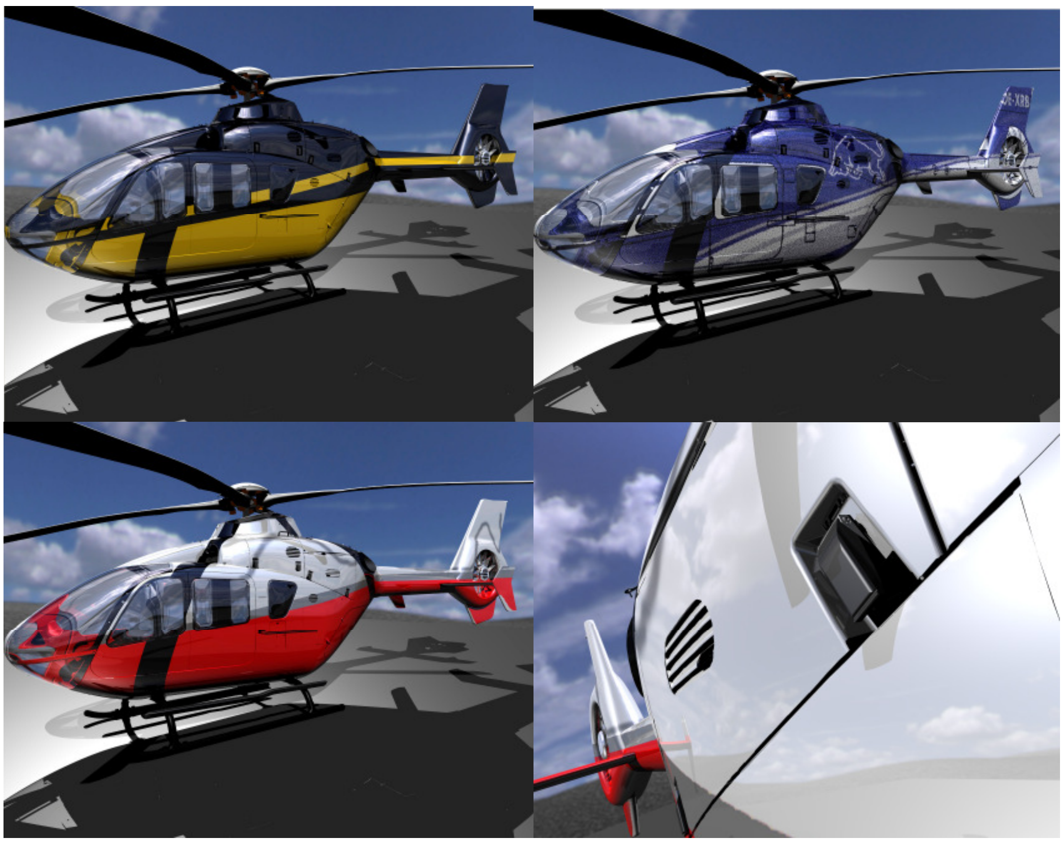


Figure 6: Different paint design for a helicopter visualized interactively with real-time ray tracing. The highly specular material results in significant reflections that need to be taken into account during the design. The close up shows the physically accurate reflection evens on curved surfaces.

6 Conclusions and Future Work

In this paper we have shown that the combination of a shared-memory multiprocessor architecture and a high-performance ray tracing implementation can be efficiently used for highly realistic real-time visualizations of highly detailed, large-scale industrial CAD databases. We have demonstrated that even a complete model of an aircraft or helicopter can be handled without approximation, simplification, or rendering artifacts. Our system supports several features important for virtual design review sessions, like distance measurement, identification, and interactive placement of individual model components, and sophisticated shading.

The proposed system is currently being evaluated at EADS, Boeing, and other companies on how it can best be integrated into the digital design workflow of large-scale engineering projects, even more complex than those presented here.

One possible direction of further research is to enhance the current setup into a "visualization service" similar to the grid computing philosophy. For example, instead of using one fixed visualization server, it would be possible to transparently provide a visualization service onto which the clients could connect without even knowing which machine they are communicating with.

Another field is the investigation of real-time global lighting simulation algorithms that build on the current architecture. In particular commodity multi-core and multi-processor systems with shared memory architectures will provide the compute performance necessary for interactive global illumination even in extremely complex datasets.

Recently, dedicated hardware for realtime ray tracing has also been developed [Woop 2005]. This Ray Processing Unit (RPU) promises to provide the same level of integration and hardware accelerated performance as current graphics chips while offering all the benefits of realtime ray tracing.

7 Acknowledgments

This project was supported by a large number of people. In particular, we would like to thank Boeing, and Dave Kasik in particular, for providing the 777 airliner dataset, for funding the project, and for supplying valuable information on practical needs of end users. The hardware for this project, the remote visualization setup, as well as help in numerous instances was provided by Silicon Graphics, Inc. Hardware for an Opteron-based SHM system was provided by AMD. Finally, we would like to thank Krzysztof Kobus, Kai Renner, Gerd Marmitt, and inTrace GmbH.

Disclaimer: Images and source 3D data provided by and used with permission of the Boeing Company and EADS Corporate Research.

8 References

[Arbabanel 1996] Robert M. Arbabanel, Eric Brechner, and William McNeely. *FlyThru the Boeing 777*. In ACM SIGGRAPH, Visual Proceedings, 1996.

[Baxter 2002] William V. Baxter III, Avneesh Sud, Naga K Govindaraju, and Dinesh Manocha. *GigaWalk: Interactive Walkthrough of Complex Environments*. In Rendering Techniques 2002, 203–214, 2002. (Proceedings of the 13th Eurographics Workshop on Rendering).

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

[Bentley 1975] Jon Louis Bentley. *Multidimensional Binary Search Trees Used for Associative Searching*. Communications of the ACM, 18(9):509–517, 1975.

[Correia 2003] Wagner T. Correia, James T. Klosowski, and Claudio T. Silva. *Visibility-Based Prefetching for Interactive Out-Of-Core Rendering*. In Proceedings of Parallel Graphics and Visualization (PGV), 1–8, 2003.

[Glassner 1989] Andrew Glassner. *An Introduction to Ray Tracing*. Morgan Kaufmann, 1989.

[Green 1993] Ned Greene, Michael Kass, and Gavin Miller. *Hierarchical Z-Buffer Visibility*. In Computer Graphics (Proceedings of ACM SIGGRAPH), 231–238, 1993.

[Kasik 2005] David J. Kasik. Boeing Company. Personal Communication, 2005.

[Klosowski 2000] James T. Klosowski and Claudio T. Silva. *The Prioritized-Layered Projection Algorithm for Visible Set Estimation*. In IEEE Transaction on Visualization and Computer Graphics, 108–123, 2000.

[Muus 1995] Michael J. Muuss. *Towards Real-Time Ray-Tracing of Combinatorial Solid Geometric Models*. In Proceedings of BRL-CAD Symposium ’95, 1995.

[Parker 1999] Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, and Peter-Pike Sloan. *Interactive Ray Tracing*. In Proceedings of Interactive 3D Graphics, pages 119–126, 1999.

[SGI Altix 2004] Silicon Graphics, Inc. SGI Altix 350 Server. http://www.sgi.com/products/-servers/alitx/350, 2004.

[VizServer 2004] Silicon Graphics, Inc. SGI OpenGL Vizserver http://www.sgi.com/products/software/vizserver, 2004.

[Wald 2001] Ingo Wald, Philipp Slusallek, and Carsten Benthin. *Interactive Distributed Ray Tracing of Highly Complex Models*. In Rendering Techniques 2001, pages 274–285, 2001. (Proceedings of the 12th Eurographics Workshop on Rendering).

[Wald 2003] Ingo Wald, Carsten Benthin, Andreas Dietrich, and Philipp Slusallek. *Interactive Ray Tracing on Commodity PC Clusters – State of the Art and Practical Applications*. In Euro-Par 2003. Parallel Processing, 9th International Euro-Par Conference, 2003. Proceedings, volume 2790 of Lecture Notes in Computer Science, 499–508. Springer, 2003.

[Wald 2004a] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University, 2004. Available at http://www.mpi-sb.mpg.de/~wald/PhD/.

[Wald 2004b] Ingo Wald, Andreas Dietrich, and Philipp Slusallek. *An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models*. In Rendering Techniques 2004, Proceedings of the Eurographics Symposium on Rendering, pages 81–92, 2004.

[Woop 2005] Sven Woop, Jörg Schmittler, and Philipp Slusallek. *RPU: A Programmable Ray Processing Unit for Realtime Ray Tracing*. SIGGRAPH 2005, 2005.