Eurographics Symposium on Parallel Graphics and Visualization (2006)
Alan Heirich, Bruno Raffin, and Luis Paulo dos Santos (Editors)

An Application of Scalable Massive Model Interaction using Shared-Memory Systems

Abe Stephens, [1] Solomon Boulos, [2] James Bigler, [1] Ingo Wald, [1] Steven Parker [1]

[1] Scientific Computing and Imaging Institute, University of Utah.
[2] School of Computing, University of Utah.

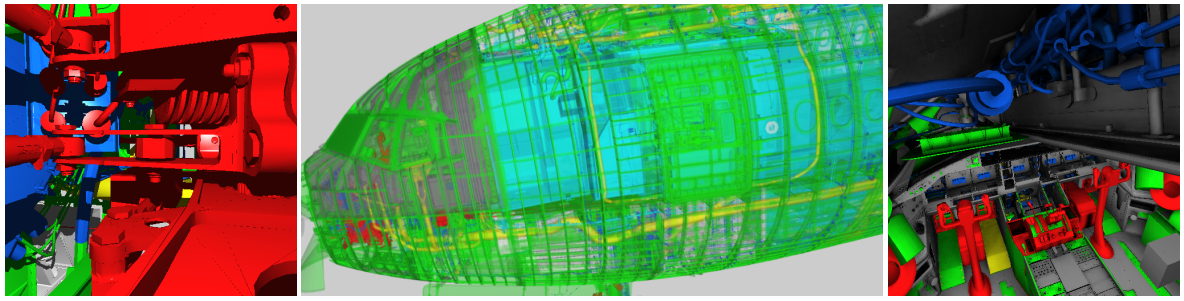


Figure 1: *Left: Phong shaded mechanical component with shadows. Center: Transparent massive model rendering which allows occluded structures to be examined in context. Right: Ambient occlusion shader inside the cockpit. (3D data provided by The Boeing Company.)*

Abstract

During the end-to-end digital design of a commerical airliner, a massive amount of geometric data is produced. This data can be used for inspection or maintenance throughout the life of the aircraft. Massive model interactive ray tracing can provide maintenance personnel with the capability to easily visualize the entire aircraft at once. This paper describes the design of the renderer used to demonstrate the feasibility of integrating interactive ray tracing in a commerical aircraft inspection and maintenance scenario. We describe the feasibility demonstration involving actual personnel performing real-world tasks and the scalable architecture of the parallel shared memory renderer.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Ray tracing I.3.2 [Computer Graphics]: Distributed/network graphics

1. Introduction

The quantity of data produced by today's engineering design and applications often exceeds the rendering capabilities of conventional interactive computer graphics. Engineers may produce tens of gigabytes of geometric model data while designing a complete product such as an airplane, but in many cases they are unable to visualize all of the data at once. This restriction may be acceptable while designing individual components of separate systems where context isn't important–however for tasks that follow fine-grained design, looking at the whole dataset at one time may be more useful.

The Boeing 777-200 dataset was used as the geometric model for this demonstration. The 3D model data consists of approximately 350 million triangles and is made available by the Boeing Company although geometry within the dataset is slightly altered. The dataset has appeared several times in architectural walk-through or massive model visualization literature [DWS05, GM05, WDS04, DWS04]. This paper, in contrast to the others, describes the first feasibility demonstration involving actual Boeing personnel using an interactive ray tracer as a tool to perform tasks from a real-world job scenario.

Designing an interactive ray tracer around a shared mem-

ory system with enough main memory to avoid the need for out-of-core techniques greatly simplifies implementation. It also eliminates the need to implement a custom high-performance communication layer in software that is necessary for most cluster based implementations [WBS02, DPH*03a].

1.1. Engineering Task

The scenario used for the demonstration was adopted from a real-world engineering process. The participating personnel include a quality analyst who discovers a problem with an aircraft's forward landing gear and a liaison engineer who is responsible for directing a solution to the problem.

First the quality assurance analyst on site inspects the potential problem and begins a remote collaborative session liaison engineer. This session includes both the ray traced visualization of the aircraft and audio and video conferencing. The liaison engineer further directs the analyst's inspection and determines the best course of action to correct the problem (see Figure 1.1). The scenario requires both remote visualization and remote collaboration using the visualization. The engineers might be on opposite ends of a production facility or could be geographically separated if the problem is detected in the field.

The interactive 3D model provided by the ray tracer serves the same purpose as conventional schematic manuals. It provides both participants with greater context of what they are looking at as well as a shared view of the correct assembly. During the demonstration the renderer was run in a collaborative workspace on a large shared memory system and was displayed both for the liaison engineer on a display wall and remotely on a tablet computer for the quality analyst. Without whole model visualization the liaison engineer and quality analyst must carefully direct each other to the same schematic diagrams and use them to separately identify and resolve the problem.

1.2. Rendering Techniques

The Boeing 777-200 dataset may be rendered on a single workstation using out-of-core, simplification, and visibility precomputation techniques [GM05]. The exploration and rendering methods used in this paper, including transparent surfaces, cutting planes, object identification and removal, and ambient occlusion (which is a special case of obscurances technique [ZIK98] and described in studio production by Landis [Lan02]) are not easily implemented with GPU techniques on such large models. In the demonstration, participants used transparent rendering and cutting planes very frequently. Usually they moved the camera to a point of interest, used cutting planes or object hiding to remove large structures in front of their point of interest, and then adjusted surface opacity to see all necessary structures in the background. This sequence of operations cannot be easily performed using schematic manuals or by rendering techniques

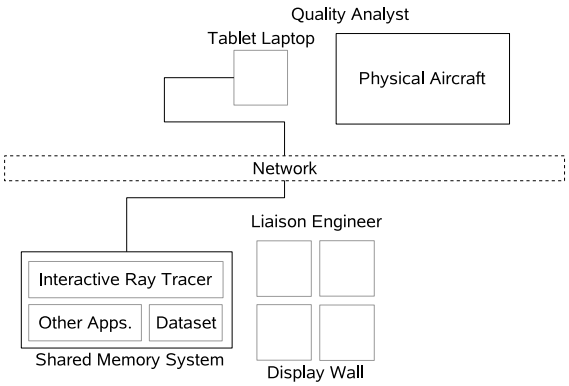


Figure 2: *Demonstration Scenario: Quality analyst inspects aircraft landing gear, discovers problem. Then uses a remote visualization of the aircraft to communicate with liaison engineer. Both use collaborative visualization to address the problem.*

that rely on pre-computed visibility since the user alters occluding surfaces as they explore the model.

2. Related Work

Large model interactive ray tracing has been applied to several scientific visualization and engineering design problems. In this section, we briefly review previous work related to the design of our system for large model visualization.

2.1. Interactive Ray Tracing

The past decade of advances in computing resources has allowed interactive ray tracing to progress from simple, sub-sampled images on 256 processors [KH95] to approximate global illumination on small clusters [WKB*02]. Earlier work involving massively parallel ray tracing [GP90, Muu95] has accurately predicted the possibility of high performance of single workstation ray tracing today [RSH05]. We believe this work will act in a similar manner: what we achieve now with supercomputers will be available in 5-10 years on a single workstation.

Interactive ray tracing for large volume visualization was first demonstrated by Parker et. al [PPL*99]. The system achieved linear scaling up to 128 processors and near-linear scaling for 1024 processors. This system was extended to clusters and allowed an 8x increase in the size of the volume data [DPH*03b].

Ray tracing of very large scenes have been demonstrated by several authors. In [PKGH97], Pharr et al. presented a system for ray tracing scenes over an order of magnitude larger than the available memory of their workstation using an out-of-core geometry caching approach. Reinhard et al. provided a system to distribute rendering of large scenes to several processors [RJ97]. In [PMS*99], a scene with 35

million spheres was rendered interactively using an SGI Origin 2000. Wald et al. obtained interactive performance for the Boeing 777 dataset of 350 million polygons on a single PC using out-of-core techniques and a high performance kd–tree implementation [WDS04].

2.2. GPU Large Model Visualization

Ray tracing is not the only viable method for large model visualization. The first system to render the Boeing dataset interactively, Boeing's FlyThru application, does not use ray tracing but instead relied on fast hardware from SGI and used model simplification [ABM96]. Correa et al. demonstrated an interactive out-of-core rendering system based on visibility preprocessing and prefetching that allowed a user to control the approximation error [CKS03]. The UNC GAMMA group has repeatedly demonstrated the feasibility of large scale model visualization using view dependent culling and mesh simplification [WVBSGM02, YSGM04]. The floating point performance of the GPU has also been used to interactively render the Boeing 777 dataset at impressive frame rates using precomputation of visibility and LOD techniques [GM05].

The majority of these methods for rendering such large models on the GPU either require several hours of preprocessing, remove occluded surfaces or are optimized for particular viewing conditions. Exploring the model in an engineering setting requires the ability to look inside of parts and behind surfaces. This requires modifying the visibility information, which makes visibility precomputation infeasible.

3. Manta Interactive Ray Tracer

The Manta Interactive Ray Tracer is an open source, highly portable renderer for shared memory supercomputers and high performance multi-core workstations. Both the configuration of the rendering pipeline and the scene graph are highly extensible. Manta is designed to be more general purpose than renderers created by Wald et al. or Reshetov et al. [RSH05] which are more highly optimized for specific situations. In addition to massive triangle rendering, Manta may be configured to render large volumes or sphere datasets.

Manta achieves interactive performance on small (8–12 processor) Itanium 2 shared memory system configurations and has scaled up to 512 processors for high resolution rendering. Because the renderer only targets shared memory systems the extra overhead associated with porting cluster based software architectures to a shared memory system are avoided. This additional overhead is likely the cause of scalability problems with large shared memory systems with ports of cluster based renderers.

3.1. Acceleration Structure

The Boeing 777 aircraft model consists of approximately 350 million polygons. Manta uses a kd–tree constructed using a surface area heuristic [MB89] to accelerate ray tracing performance. With a target audience of shared memory supercomputers, we use a parallel build algorithm to reduce the time required to build the acceleration structure. The quality of the structure can be controlled through user selected parameters and influences the total build time and final kd–tree file size.

The kd–tree used in the experiments for this paper and the interactive sessions with quality engineers was built over night using a 64 processor system and is approximately 29 GB in size. The Boeing 777 data itself is 28 GB for triangle data including smooth vertex normals and 1.3 GB for part model numbers and group identification information. We have found that a build requiring just two hours produces a marginally acceptable tree (in terms of rendering performance), but we use the higher quality tree for its improved performance during rendering. The 64 GB runtime size of the renderer with the model and kd–tree loaded is within the main memory size of most shared memory supercomputer and high end multi-core server configurations produced today.

3.2. Ray Packets

Manta's highly modular architecture relies on virtual function calls to pass control between routines. Statically sized ray packets are used to pass ray tracing data up and down the call stack and help amortized the overhead of virtual function lookups. Ray packets in Manta are considerably larger then those used by other renderers. Each element in the ray packet contains a ray direction and origin and information about the minimum intersection point along the ray such as pointer to a hit primitive and material shader.

Each ray packet includes a set of flags with general information about all of the rays it contains. These flags are used in the code to apply special case optimizations. For example if all of the rays in the packet have a common origin, an certain optimized intersection or shading routine may be used. Flags are also set after specific properties are computed, such as inverse ray direction, so that the properties are only computed once during the life of a packet.

In addition to amortizing the cost of virtual function lookups, ray packets increase opportunities for instruction level parallelism. On in-order processors with several floating point units, like the Intel Itanium 2, code must be written such that the compiler produces software pipelined loops in order to increase instruction level parallelism. Additionally, ray packets are laid out in memory as structures-of-arrays. This vertical organization of member fields (so that an array of three component vectors would be stored as a separate array of each component) enables simple loading into SIMD registers and further increases the opportunity for instruction level parallelism.

The default ray packet size in Manta is 32 elements. Our experiments show that eight elements are sufficient to amortize virtual function looks for most situations. Larger packets

are benefitial when special case optimizations using packet flags are available or vertical code is used to increase instruction level parallelism. When Manta is run on systems with SSE units, large ray packets are broken into smaller four element wide packets during traversal or shading. Although with two floating point scalars per operand are available on the Itanium 2 their use through SSE intrinsics is not as beneficial relative to the SSE units on other processors.

Lastly, ray packets increase memory access coherence during traversal, triangle intersection, and write-back to the frame buffer. Primary rays are sent coherently by the image traverser and pixel sampler (described below), some care must be used to send secondary rays or shadow rays coherently.

3.3. Pipeline and Rendering Stack

Manta's modular components are organized into a rendering call stack of routines executed asynchronously within each thread and a multi-stage pipeline which contains points for synchronization. The simplest rendering pipeline contains two stages; image display and rendering. This configuration is shown in Figure 3.

One frame buffer is operated on by each stage in the pipeline. Threads synchronize for state changes between each stage. In the example pipeline in Figure 3(a) image display is the first stage, executed by only one thread, and the second stage is rendering. There is a one stage lag between rendering and image display so as the current frame is displayed, the next frame is rendered. Additional stages can be added to the pipeline and the display component may be changed depending on the application. During feasibility demonstration a special display component was used to write each frame to a SHM segment. In most cases a GLX display component is used.

During the rendering stage of the pipeline each thread asynchronously traverses the rendering stack. This call stack of modular components consists of an Image Traverser, Pixel Sampler, Renderer, and lastly scene graph (see Figure 3(b)). Inside the Image Traverser each thread operates on a tile of the image which is assigned by a load balancer. The Image Traverser determines integer pixel coordinates within a tile and invokes the pixel sampler. The pixel sampler assigns coordinates within each pixel to individual rays. Ray packets are statically allocated in the pixel sampler and then sent up the rendering stack with input data (the pixel sampler assigns sub-pixel image space coordinates, the camera assigns each ray packet element an origin and direction, etc). The Renderer component of the rendering stack is responsible for invoking the camera, sending the ray packet to the scene graph for intersection, and dispatching ray packets to material shaders. Ambient occlusion shading employs a slightly different rendering stack configuration described in section 4.3.

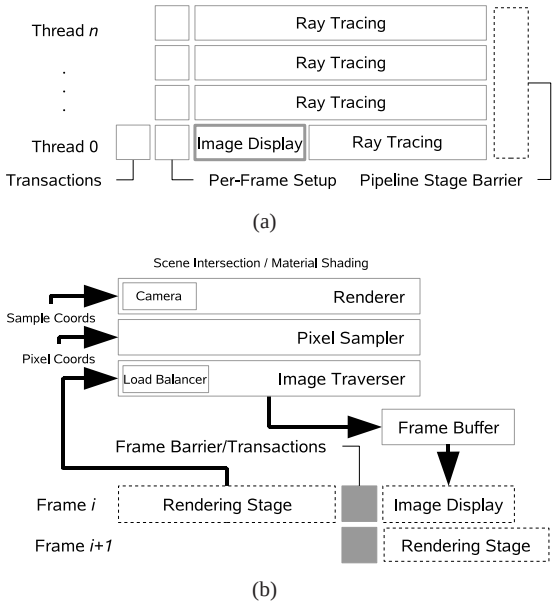


Figure 3: *Two stage rendering pipeline (a) executed by all threads and rendering call stack (b) executed asynchronously in each thread.*

3.4. Synchronization and Transactions

While ray tracing is traditionally considered embarassingly parallel, the design of an interactive ray tracer quickly reveals practical difficulties. For example, although each ray's computation is independent, the scene state must be consistent throughout the rendering of the frame. This requires that there are several points during rendering that require all the threads to synchronize.

Synchronization of the Manta pipeline occurs in two places; reconfiguration (changing the number of available processors or modifying the pipeline) and at the conclusion of rendering a frame for state changes. How to synchronize for state changes is an important consideration in the design of an interactive ray tracer. While double buffering all state is one possibility, Manta maintains a FIFO queue of incremental state changes called Transactions.

Transactions are extremely short callbacks executed by a single thread before each pipeline stage. Camera movement and object selection are common examples. These callbacks can safely modify any shared state in the renderer. Transactions allow the renderer to run completely asynchronously from a graphical user interface. The application sends transactions to Manta and Manta displays frames directly through GLX. Manta supports other forms of callbacks for animation that may be executed either by one thread or in parallel.

Balanced load is maintained by labelling each task as inherently load balanced, dynamically load balanced or load

imbalanced. Dynamically load balanced tasks, such as rendering are invoked last in order to mask the overhead of load imbalanced tasks like image display. Ultimately the load balancer is responsible for ensuring good scaling as the number of processors increases. The barriers and counters used for synchronization and work assignment are implemented using high performance atomic operations and do not create a bottle neck.

4. Visualization Techniques

In this section, we describe the different shading and user inspection features that were implemented in Manta to improve user exploration of the Boeing 777.

4.1. Cutting Planes and Hiding Objects

The Boeing 777 dataset is incredibly complex. Within most visible portions of the model lies a complex network of wiring, cabling and other small structures. To allow users to see these otherwise occluded objects, Manta allows the user to place and manipluate cutting planes while rendering (See Figure 4).

The cutting plane is placed on top of the scene graph. This allows the cutting plane to modify each ray intersection interval so that objects on the culled side of the plane will not return valid intersections.

In addition to removing geometry with cutting planes, Manta allows users to click on and hide individual objects in the dataset. As with cutting planes, objects that match hidden serial numbers simply do not return intersections with rays. Figure 4 demonstrates the use of a cutting plane to remove the top of the aircraft and the removal of several objects in the forward galley to reveal structure below.

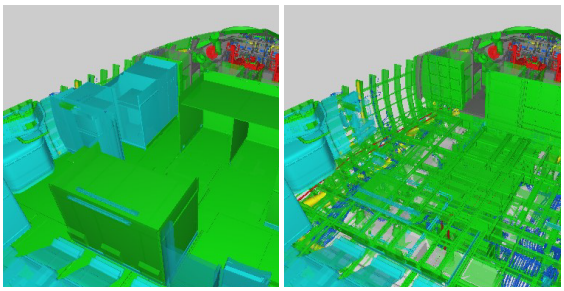


Figure 4: *Using a cutting plane to examine the forward galley and then selecting and hiding individual objects to reveal the structure below.*

4.2. Transparent Rendering

While cutting planes and object hiding allows for a wide range of model inspection, it is sometimes necessary for a user to maintain their current context while investigating occluded surfaces. Through the use of a user controlled global transparency value, Manta allows users to see occluded structures in a less binary manner. Transparent rendering of the dataset provides users with a very different inspection ability then a traditional architectural walk through provides (See Figure 5).

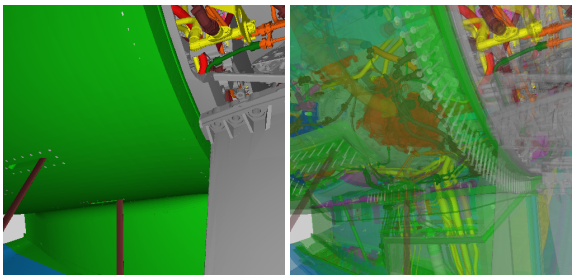


Figure 5: *Comparison between Lambertian shading with shadows and transparent rendering in the engine nacelle. Transparent rendering reveals the intricate details of the assembly and preserves context in the area, while standard Lambertian shading hides much of the detail.*

Transparency is implemented by modifying the kd–tree traversal to blend colors between sorted subsequent triangle intersections along a ray. This approach is similar to ray marching through a volume dataset. The termination criteria of the traversal was altered to allow rays to attenuate as they passed through the model instead of terminating the ray after it passed through a leaf node containing a valid intersection. In most cases, users adjusted opacity so that they could see through a small number of surfaces but still maintain context within the model. As is common for volume rendering, once the opacity reaches a cutoff value (0.95 was used for the images in this paper) the ray traversal is terminated for efficiency.

Varying the transparency value increases the cost of rendering. As the global transparency value is varied, rays will need to intersect more triangles and kd–tree nodes. We have measured performance to be linear as opacity increases from moderate values with the Boeing 777 dataset on a 16 processor Itanium 2 system.

4.3. Ambient Occlusion

While the previous visualization methods allow a user to interactively inspect the Boeing 777 model, the renderings are far from an accurate depiction of the real parts the quality assurance analyst will see on site. Providing the liason engineer with a full interactive global illumination may allow for more meaningful interaction with the model. This approach is complicated due to the computational burden of global illumination and lack of lighting and material information for the model.

The computational burden of global illumination stems from the large number of secondary rays required to compute an acceptable solution. For each primary ray, several

secondary rays are required. The material information included with the dataset is a simple color coding; an advanced global illumination solution would desire identification of metallic, glass and plastic objects. More importantly, however, an accurate lighting simulation of the aircraft would require the placement of light sources which do not come with the model. In the case of field work, these light sources would need to be defined to match the analyst's environment.

Despite this lack of information, we can use ambient occlusion to provide a result that is qualitatively similar to a diffuse global illumination solution without relying on placement of light sources. Ambient occlusion approximates the amount of ambient light that would reach a surface by sampling a hemisphere about the intersection point with secondary rays. This results in renderings with increased contrast around small geometric details due to local occlusions (See Figure 6). The ratio of occluded to total samples is calculated and used as an ambient term. Secondary rays are considered occluded if they intersect another surface with a user controlled cutoff distance.

To reduce the computational overhead involved in using many secondary rays, ambient occlusion was implemented in Manta following existing instant global illumination techniques [WKB*02]. Interleaved sampling [KH01] was used to select a small subset of precomputed ray directions in a hemisphere. This subset was transformed to the orientation of each primary ray intersection point hemisphere and secondary rays were produced for each direction. The hit ratio on the hemisphere for each primary ray intersection point was filtered together using a kernel similar to that used by Ward for irradiance caching [WRC88].

Manta's rendering stack was reconfigured to more closely follow the direction of Wald et al.ś instant global illumination. The Image Traverser and Pixel Sampler component functionality was combined so that filtering could be performed on a discontinuity buffer spanning many pixels. Each rendering thread maintained an individual discontinuity buffer. Additionally the Renderer component, at the top of Manta's rendering stack (see 3(b)) was modified to store hit and material information in the discontinuity buffer.

4.4. Remote Visualization

Remote rendering and collaboration is an essential component of the demonstration scenario (see Figure 1.1) and is an important consideration for any large shared memory visualization system. Larger configurations are often accessed in machine rooms from users' work areas over local area networks with insufficient bandwidth and may be shared resources within an organization. Remote visualization also reduces the need to transfer massive, possibly proprietary, data sets to collaborators.

Compression utilities such as OpenGL Vizserver read back frame buffers from the graphics driver, apply a variety of compression algorithms to the image stream and then transmit it to a remote client. The compression utility is transparent to the rendering application and provides for some collaboration – users may interact with the same application running on the host system and will see each others' mouse pointers. Using this type of utility it is possible to control an interactive ray tracing session with modest resolution from a laptop across the Internet.

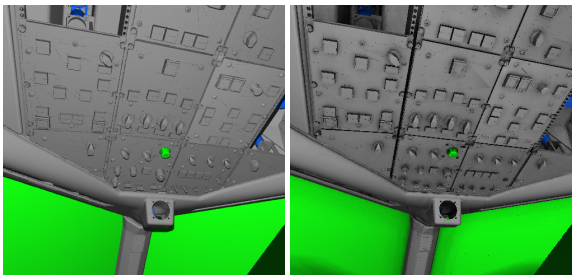


Figure 6: *Comparison between Lambertian shading with shadows and ambient occlusion (5x5 filter, four secondary rays) of the overhead control panel in the cockpit. Ambient occlusion enhances the contrast between the panel and individual buttons, switches and dials.*

5. Results

The Boeing 777 dataset has been rendered using Manta on a variety of systems with different configurations. On most configurations memory is the greatest limiting factor. The standard small system to render the dataset is a 12 processor Itanium 2 SGI Prism. This configuration was used at a demonstration of Manta at SIGGRAPH 2005. The renderer is commonly run at higher resolutions on a 64 processor system and has been scaled up to 128 and 512 processor configurations.

5.1. Renderer Performance

The feasibility demonstration in this paper was targeted at a 128 processor system with 256 gigabytes of main memory. The system was used to render the aircraft model, compress the image stream for remote vizserver connections, and handle several additional interaction streams from video conferencing. 112 of the 128 processors were used for ray tracing, other applications used the remaining processors.

All of the benchmarks were conducted using a program that sends camera update transactions from a pre-recorded path to Manta at a prescribed rate and approximates the behavior of a real user. In the benchmark plots the x axis indicates the transaction number when each data point was recorded.

Figure 7(a) contains a processor scaling plot of the renderer's performance using a simple shadowed Phong material on the demonstration system. Resolution 1024x768 was used for the benchmark. The software scales well through 64

processors (average 22 fps). On 126 processors Manta ran at 82 percent of linear with an average speed of 40.0 fps.

The results in both Figure 7(b) and Figure 7(c) were recorded on an older 62p 1.5 GHz Itanium 2 SGI. The alpha value 0.3 was used in Figure 7(b) this is nearly the minimum useful alpha value and consititues a lower bound on transparent performance. Alpha 0.3 is marginally fast enough to be useful on 60 processors. It is likely that a user would need to navigate using less tranparency and then decrease the value after finding the location of interest.

The 800x600 resolution is too high to render using the ambient occlusion shader with reasonably sized systems. Unfortunately a 128 processor configuration was not available to test the shader on. (see figure 7(c)) Decreasing the resolution is necessary to obtain more interactivity, but not real-time results. Currently the shader would best be used to produce a high quality image after the user stops moving the camera quickly.

5.2. Discussion

One limitation of our system is that the objects can only be hidden, not moved with the model. While being able to hide or see through an object may be a useful mechanism for inspection, it may be more intuitive to allow a user to simply move objects out of the way. This limitation is inherited from the static nature of the acceleration structure used and is a topic for future work.

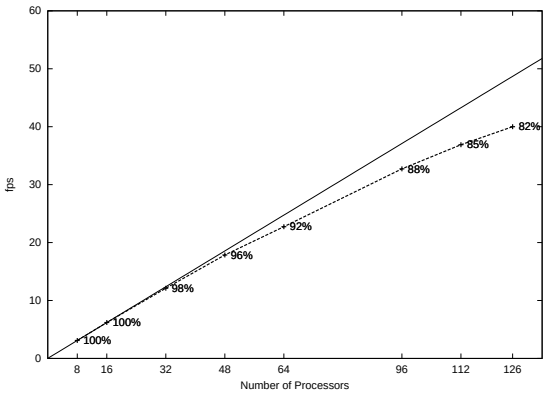
The high cost of the ambient occlusion shader, relative to transparent or shaded surface rendering prohibits it from being feasible to use in the aircraft maintenance and quality scenario. Never-the-less, five frames per second is still interactive and will contine to improve with overall system performance.

It is important to make the distinction between technologies which are feasible to integrate in a real-world process, and those that are practical to widely deploy. Today of course, not every member of an engineering organinzation can have a 128 processor system beside their desk – collaborative/remote rendering must be used instead.

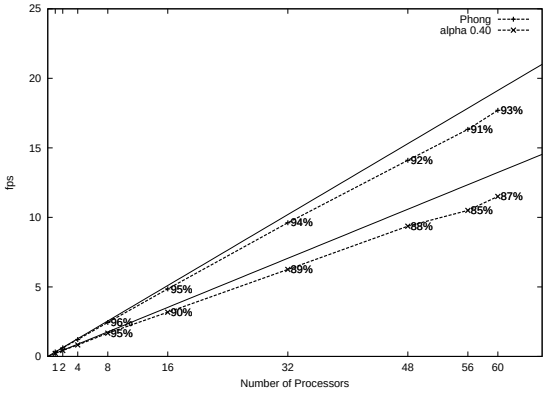
The visualization objective of the feasibility demonstration was to build a system that would integrate massive model rendering with a real-world engineering process–and obtain an acceptable level of performance. Manta's general purpose modular design permitted straight forward integration in a work flow with other applications. The renderer's scalable architecture enabled it to be deployed on a sufficiently large system.

6. Acknowledgements

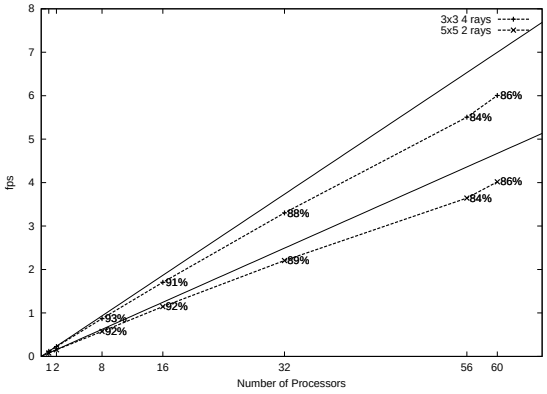
This work was funded by the DOE ASC Center for the Simulation of Accidental Fires and Explosions and the Utah Center of Excellence for Interactive Ray-Tracing and Photo Realistic Visualization. The feasibility demonstration was



(a) Shadowed Phong shading 128 processor 1.6 GHz (Demonstration machine). 1024x768 pixels



(b) Shadowed Phong and very low alpha value. 62 processor 1.5 GHz. 800x600 pixels



(c) Ambient Occlusion. 3x3 and 5x5 filters. 62 processor 1.5 GHz. 512x512 pixels

Figure 7: *Scaling performance on a variety of machines. Each plot indicates the average performance along a camera path consisting of several thousand camera updates. Considerable variation in scene complexity is experienced along the paths. The solid line drawn with each plot indicates linear performance. The ratio between measured and linear performance is indicated at each data point.*

supported by Silicon Graphics Incorporated, The Boeing Company, and Intel Corporation. The authors would like to thank Hansong Zhang, Kenny Hoff, Dan McLachlan, Jimmy Wang, and Rocky Rhodes for their assistance from SGI as well as David Kasik and Jim Troy from Boeing.

References

[ABM96] ARBABANEL R. M., BRECHNER E., MCNEELY W.: FlyThru the Boeing 777. In *ACM SIGGRAPH, Visual Proceedings* (1996).

[CKS03] CORREA W. T., KLOSOWSKI J. T., SILVA C. T.: Visibility-based prefetching for interactive out-of-core rendering. In *PVG '03: Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (Washington, DC, USA, 2003), IEEE Computer Society, p. 2.

[DPH*03a] DEMARLE D., PARKER S. G., HARTNER M., GRIBBLE C., HANSEN C.: Distributed Interactive Ray Tracing for Large Volume Visualization. In *Proceedings of the IEEE PVG* (2003), pp. 87–94.

[DPH*03b] DEMARLE D. E., PARKER S., HARTNER M., GRIBBLE C., HANSEN C.: Distributed interactive ray tracing for large volume visualization. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (Oct. 2003), pp. 87–94.

[DWS04] DIETRICH A., WALD I., SLUSALLEK P.: Interactive Visualization of Exceptionally Complex Industrial Datasets. In *ACM SIGGRAPH 2004, Sketches and Applications* (August 2004).

[DWS05] DIETRICH A., WALD I., SLUSALLEK P.: Large-Scale CAD Model Visualization on a Scalable Shared-Memory Architecture. In *Vision, Modeling, and Visualization* (July 2005), pp. 303–310.

[GM05] GOBBETTI E., MARTON F.: Far Voxels – a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. *ACM Transactions on Graphics 24*, 3 (August 2005), 878–885. Proc. SIGGRAPH 2005.

[GP90] GREEN S. A., PADDON D. J.: A Highly Flexible Multiprocessor Solution for Ray Tracing. *The Visual Computer 6*, 2 (1990), 62–73.

[KH95] KEATES M. J., HUBBOLD R. J.: Interactive ray tracing on a virtual shared-memory parallel computer. *Computer Graphics Forum 14*, 4 (Oct. 1995), 189–202.

[KH01] KELLER A., HEIDRICH W.: Interleaved Sampling. *Rendering Techniques* (2001), 269–276. (Proceedings of the 12th Eurographics Workshop on Rendering).

[Lan02] LANDIS H.: Production-Ready Global Illumination. In *SIGGRAPH 2002 RenderMan in Production Course Notes*. July 2002.

[MB89] MACDONALD J. D., BOOTH K. S.: Heuristics for ray tracing using space subdivision. In *Proceedings of Graphics Interface* (1989), pp. 152–63.

[Muu95] MUUSS M.: Towards real-time ray-tracing of combinatorial solid geometric models. In *Proceedings of BRL-CAD Symposium* (1995).

[PKGH97] PHARR M., KOLB C., GERSHBEIN R., HANRAHAN P.: Rendering Complex Scenes with Memory-Coherent Ray Tracing. *Computer Graphics 31*, Annual Conference Series (Aug. 1997), 101–108.

[PMS*99] PARKER S. G., MARTIN W., SLOAN P.-P. J., SHIRLEY P., SMITS B. E., HANSEN C. D.: Interactive ray tracing. In *Proceedings of Interactive 3D Graphics* (1999), pp. 119–126.

[PPL*99] PARKER S. G., PARKER M., LIVNAT Y., SLOAN P.-P., HANSEN C., SHIRLEY P.: Interactive ray tracing for volume visualization. *IEEE Transactions on Computer Graphics and Visualization 5*, 3 (1999), 238–250.

[RJ97] REINHARD E., JANSEN F. W.: Rendering Large Scenes using Parallel Ray Tracing. *Parallel Computing 23*, 7 (July 1997), 873–885.

[RSH05] RESHETOV A., SOUPIKOV A., HURLEY J.: Multi-level ray tracing algorithm. In *(Proceedings of SIGGRAPH* (2005), pp. 1176–1185.

[WBS02] WALD I., BENTHIN C., SLUSALLEK P.: *OpenRT - A Flexible and Scalable Rendering Engine for Interactive 3D Graphics*. Tech. rep., Saarland University, 2002. Available at http://graphics.cs.uni-sb.de/Publications.

[WDS04] WALD I., DIETRICH A., SLUSALLEK P.: An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models. In *Rendering Techniques 2004, Proceedings of the Eurographics Symposium on Rendering* (2004), pp. 81–92.

[WKB*02] WALD I., KOLLIG T., BENTHIN C., KELLER A., SLUSALLEK P.: Interactive Global Illumination using Fast Ray Tracing. *Rendering Techniques* (2002), 15–24. (Proceedings of the 13th Eurographics Workshop on Rendering).

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. In *Computer Graphics (Proceedings of SIGGRAPH 88)* (Aug. 1988), vol. 22, pp. 85–92.

[WVBSGM02] WILLIAM V. BAXTER I., SUD A., GOVINDARAJU N. K., MANOCHA D.: Gigawalk: interactive walkthrough of complex environments. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2002), Eurographics Association, pp. 203–214.

[YSGM04] YOON S.-E., SALOMON B., GAYLE R., MANOCHA D.: Quick-VDR: Interactive view-dependent rendering of massive models. In *VIS '04: Proceedings of the conference on Visualization '04* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 131–138.

[ZIK98] ZHUKOV S., IONES A., KRONIN G.: An ambient light illumination model. In *Ninth Eurographics Workshop on Rendering* (June 1998), pp. 45–56.

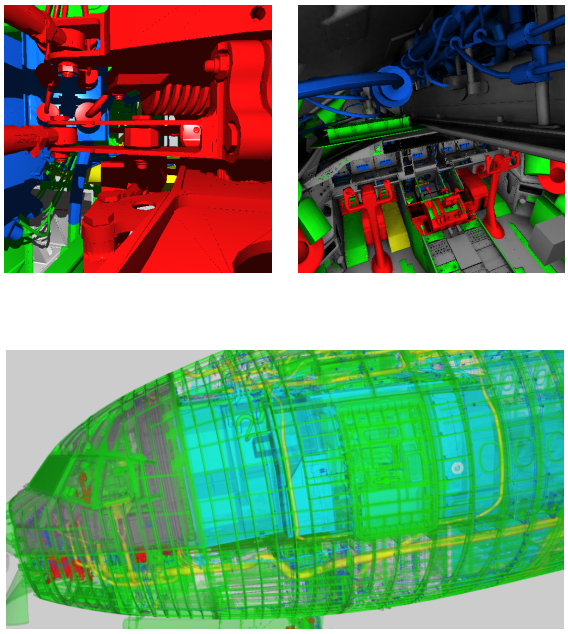


Figure 1: *Top Left: Phong shaded mechanical component with shadows. Bottom: Transparent rendering. Top Right: Ambient occlusion shader. (3D data provided by The Boeing Company.)*

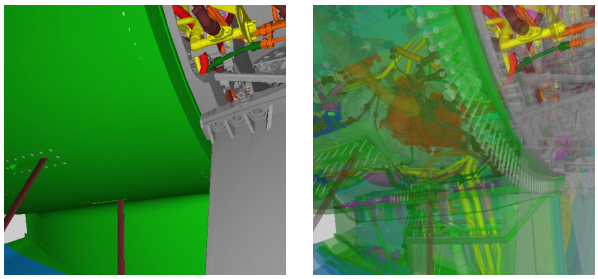


Figure 5: *Comparison between Lambertian shading with shadows and transparent rendering in the engine nacelle.*

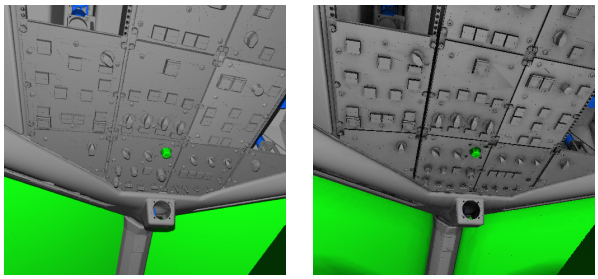


Figure 6: *Comparison between Lambertian shading with shadows and ambient occlusion (5x5 filter, four secondary rays).*