Improving Interaction Performance for Ray Tracing

Daniel Kurz Christopher Lux Jan P. Springer Bernd Fröhlich

Bauhaus-Universität Weimar

Abstract

We have developed an approach for improving the performance of object manipulation in ray tracing systems. We assume that users alternate between navigating the scene and manipulating objects in the scene. We divide the scene in objects currently manipulated by the user and the non-interactive rest. Once a user stops navigating, we compute and store the first order reflections for the non-interactive objects. In a composition step only the manipulated objects need to be fully ray traced, while the stored reflections of the rest of the scene have to be tested only against the manipulated objects. In an initial evaluation we found that this approach significantly improves frame rates during object manipulation—and thus increases interaction performance.

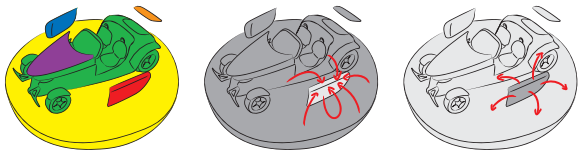
Our approach directly extends to refraction and shadow rays. It could also be used for further ray generations beyond the first order effects, but the speedup would strongly depend on the actual scene and it would probably be less significant. Our approach is independent of the underlying spatial data structure and it neither reduces visual quality nor does it introduce visual artifacts—within its constraints.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Display Algorithms; I.3.7 [Computer Graphics]: Ray Tracing; I.3.7 [Computer Graphics]: Virtual Reality

1. Introduction

Real-time ray tracing has recently become a feasible rendering method for virtual reality applications. While this rendering method can generate excellent image quality considering a number of optical effects (e. g. reflections, refractions, and shadows), the interactivity is often quite limited. However, virtual reality applications often require the interactive manipulation of objects, which may change their position and orientation from frame to frame depending on user input. This is still a challenge for ray tracing systems—in particular if they are supposed to run on a single computer.

Springer et al. [SBW*07] observed that users of projection-based virtual reality applications can often cope with lower frame rates for navigation tasks than for object manipulation tasks. As a consequence they separated the scene into a set of currently manipulated objects and the rest. Each set of objects is rendered on a different GPU, the partial images are digitally composed in a Sort-Last manner, and the result is displayed. The set of interactive objects is typically rendered at much higher frame rates than the remainder of the scene. Thus, user interaction is incorporated with much lower latency than with a conventional single-frame rate rendering approach.



(a) (b) (c)

Figure 1: Example scene with multiple objects. The left door of the car is currently manipulated (a). The reflections are decomposed into reflection of the scene in the left door (b) and the reflection of the left door in the rest of the scene (c).

Their original multi-frame rate approach was developed for rasterization-based rendering. It extends also to ray casting, but for ray tracing it is not immediately applicable due to the light interaction between interactive objects and the rest of the scene. Here we report on an initial idea to use an approach similar to multi-frame rate rendering for the first order reflections in a limited ray tracing system. The scene is also divided into two sets: the set of currently manipulated objects O and the set R containing the rest of the scene (cf. figure 1). The specular reflections between these two sets can be decomposed into reflections of R in O and vice versa (cf. figure 1b

and 1c). We recompute the reflections within the objects of R whenever the view point changes. As long as no view point change occurs, we use these pre-computed reflections as the basis for interactively ray tracing the scene. Because typically only a small subset of the scene is manipulated, the interactivity of our ray tracing system is significantly improved during object manipulation.

2. Related Work

In a user study Springer et al. [SBW*07] showed that 3D task performance was similar for single-frame rate rendering of the whole scene at 30 Hz compared to multi-frame rate rendering at 10/30 Hz, i. e. 10 Hz for rendering the static scene parts while rendering manipulated objects at 30 Hz. In [SLRF08] Springer et al. showed how to employ deferred shading [DWS*88] based on G-buffers [ST90], containing per-pixel information of geometric properties such as depth values or normal vectors, to achieve interactive frame rates for effects affecting the entire scene (e. g. manipulation of lights).

Real-time ray tracing approaches for the visualization of complex static scenes employ PC clusters [WSB01] and recently single workstations [DWS04]. Recent work on ray tracing of animated and interactive scenes focuses on fast to build acceleration structures [WMG*07]. Wald [Wal07] shows how to rebuild a bounding volume hierarchy (BVH) for the entire scene on a per-frame basis. Yoon et al. [YCM07] present a technique for locally restructuring parts of a BVH instead of rebuilding the entire structure, which works well if only small portions of the scene are manipulated. In contrast, our approach reduces the number of rays and the number of objects that need to be considered during object manipulation.

3. Description of Rendering Approach

Multi-frame rate rendering is a parallel rendering technique which decomposes a scene into user-manipulated elements and the rest of the scene. Using digital composition the results of the two frame buffer images are merged into the final rendering. The manipulated objects may change throughout the use of the application (e. g. users may first manipulate the door of a car, then a car seat, and later the hood). We have developed a ray tracing approach that allows us to separately render the currently manipulated objects and the rest of the scene and to efficiently combine the partial images including correct inter-reflections between the elements in both images.

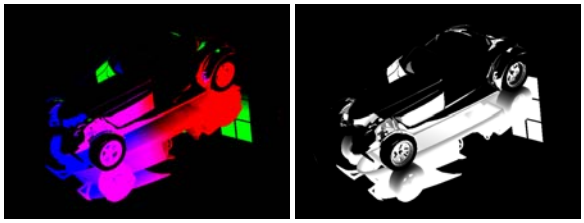
The proposed algorithm is outlined for first order reflections. The scene S consists of a set of objects, which is decomposed into the set of currently manipulated objects O and the set R containing the rest of the scene. The algorithm splits the rendering process into three logical stages: pre-computing reflections within the set R, updating reflections and active objects with respect to the set O, and final compositing. We assume that users alternate between navigating the scene and manipulating objects. During navigation, we use regular ray tracing. Once navigation stops the pre-computation stage is performed once while the other stages are updated each frame.

The final composition stage combines two frame buffer images $\mathrm{I}(R)$ and $\mathrm{I}(O)$ of the set R and the set of manipulated objects O. These images contain color and position information of primary ray intersections. The final color is computed by comparing the images pixel-wise, thereby determining which associated intersection is closer to the viewer. The color components of both frame buffer images must contain proper reflections from the complementary parts of the scene to generate correct visual results. This is achieved by tracing secondary rays for the whole scene while rendering the active objects into $\mathrm{I}(O)$. Because the pre-computation of the static parts of the scene only considers the scene without the active objects, the generated image $\mathrm{I}(R)$ does not show reflections of O. To generate an image containing the proper reflections we employ a technique derived from deferred shading. During pre-computation a series of G-buffers is constructed containing information about primary and secondary ray intersections such as positions, normal vectors, and material information. This information is later used to compute the final shading of $\mathrm{I}(R)$ by tracing secondary rays from the pre-computed positions for the image of the set R. The idea is that during the update stage only intersections with the active objects O need to be detected. This operation also outputs G-buffers containing information for the updated secondary ray intersections. The image $\mathrm{I}(R)$, containing proper reflections, is derived from the pre-computed and updated G-buffers by a merge operation. The positions of the secondary ray intersections are compared and, for intersections closer to the reflecting surface, the proper shading is calculated. After this operation $\mathrm{I}(R)$ and $\mathrm{I}(O)$ show correct reflections of the complementary parts of the scene and can be digitally composed into the final image.

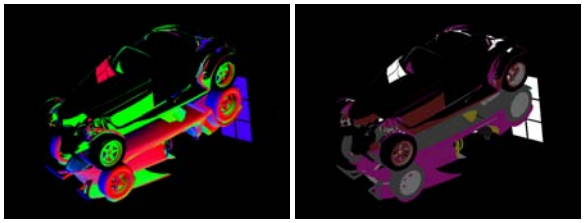
Our method also applies to refraction and shadow computations. Extending the G-buffers by additional components allows for storing intermediate results from the refraction and shadow ray traversals. During the merge operation before the compositing stage these additional components have to be taken into account for the generation of $\mathrm{I}(R)$ to include proper refractions and shadows.

4. Example Implementation

We implemented the proposed algorithm with an in-house developed GPU-based real-time ray tracing framework. The prototype is build upon regular grids as the main acceleration structure and uses OpenGL shader programs to implement the ray traversal. Each potentially active object is stored in a separate grid for ray-intersection acceleration [PBMH02]. These grids are additionally stored in a global grid, which is locally rebuild after each interaction. This allows for transforming parts of the scene without fully rebuilding global



(a) Position **(b)** Distance

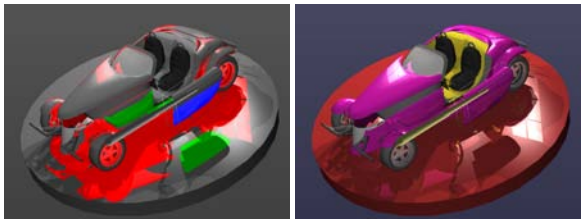


(c) Normal **(d)** Material

Figure 2: G-buffer contents from the pre-computation pass for the reflected surfaces of the static scene.

acceleration structures. Upon traversal of a particular grid rays are transformed into the local grid coordinate system for intersection testing. For efficiency reasons rasterization and shadow mapping are used to replace primary ray intersection and shadow ray traversal computations. This divides the ray tracing algorithm into multiple rendering passes: shadow map creation, primary ray rasterization, and tracing of reflection rays. The primary ray rasterization pass uses multiple render targets (MRT) to produce G-buffers for origins and directions of the reflection rays as well as for local lighting. The output buffers are used as input to the final ray tracing pass.

We integrated our algorithm using modified rasterization and ray tracing passes. Instead of directly calculating lighting coefficients we store the necessary information in additional render targets. The pre-computation stage generates the following G-buffer portions for first order reflections (as illustrated in figure 2a to 2d): position of the ray intersection with R, distance of the intersection from the reflecting surface, normal vector, and material parameters.



(a) Composite **(b)** Final

Figure 3: Reflections of the static scene and the active object are composed (a) and lead to the final image (b).

First, in the update stage the shadow maps are updated using the manipulated scene. Then the active scene objects O are rendered using the unaltered ray tracing algorithm resulting in the frame buffer image $\mathrm{I}(O)$. Simultaneously a mask is generated to distinguish between fragments which belong to R and O. Based on this mask pre-computed reflections are updated only for visible fragments corresponding to R. This update results in similar G-buffer output as the pre-computation stage but shows only potential reflections of the active objects. Based on the distances contained in the pre-computed and updated G-buffers the proper reflections and final shading are computed. The result is stored in the frame buffer image $\mathrm{I}(R)$. Finally, the composition stage merges the frame buffer images $\mathrm{I}(O)$ and $\mathrm{I}(R)$ to the final output image. Figure 3a illustrates the updated reflections for the static parts of the scene in green and the active objects in blue while the unaltered pre-computed reflections are shown in red.

5. Discussion

We conducted a brief performance evaluation to examine the impact of the proposed method. For the evaluation we used an Intel Core2 Duo 2.4 GHz workstation with 2 GiB RAM and an NVIDIA GeForce 8800 GTX graphics card driving a projection-based display installation. The test scene consisted of four objects and is shown in figure 4. Using a pick ray the user was able to select as well as translate and rotate single objects. The test was performed multiple times under different view points. Figure 4 illustrates the results measured in frames per second. The gray bar represents the performance using the original ray tracing approach. The results show that our proposed method results in a significant performance improvement during the manipulation of single objects.

The achievable speedup for scene manipulations depends on the model complexity and the size of the projection of the active objects. The screen-projection size determines the amount of rays that must be traced for the computation of $\mathrm{I}(O)$ while the geometric complexity determines the overall

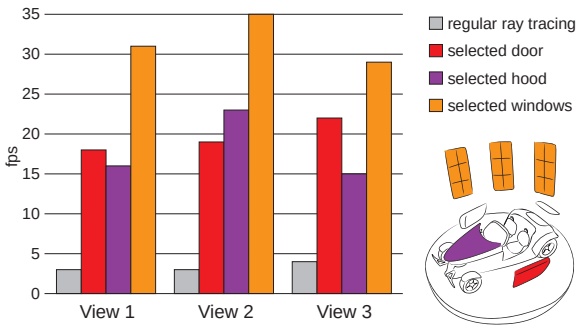


Figure 4: Performance for object manipulation under different view points. Note, that the frame rates while selecting the window (orange bars) are particularly high due to the fact that the window is not reflective.

cost of ray tracing the active objects during the update stage. It is important to note that due to the initial pre-computation of the static scene parts a short lag is introduced before the manipulation can start. This initial lag amounts to at most one frame of the regular ray tracing algorithm.

A large memory overhead is introduced by using the proposed G-buffer approach. The pre-computation stage generates at least six buffers holding information about primary and reflection ray intersections, i. e. positions, normal vectors, and material information in each case. At the same time the update stage introduces at least another three G-buffer components to hold the reflection information. For temporarily storing the intermediate frame buffer images $I(R)$ and $I(O)$ two additional frame buffers are required. Extending the described technique to refractions and shadows will further multiply the memory requirements.

The proposed method handles only first order reflections, refractions and shadows. For an arbitrary number of ray generations, the pre-computation as wells as the update stage have to generate deeper G-buffers containing information about these ray intersections. When merging the results pre-computed intersections can be used until an intersection with an active object is found. Further ray intersections must be updated using regular ray tracing against the whole scene.

6. Conclusions and Future Work

We presented an approach that improves interactive object manipulation in ray tracing systems. By dividing the scene into objects currently under user manipulation and a static remainder we are able to pre-compute reflections for the non-interactive scene part. In a composition step only the manipulated objects need to be considered, which allows for displaying the final image at interactive frame rates without reducing visual quality or introducing visual artifacts.

The extension of the proposed method to refractions and shadows introduces a large memory overhead. With further generalization to higher ray-recursion depths the advantage of the pre-computed effects diminishes due to additional computational overhead. We are exploring ways to reduce both memory and computational costs to allow for a more general application of the algorithm.

We are currently investigating the adoption of a multi-GPU solution, as described by Springer et al. [SLRF08], where one GPU is dedicated to the pre-computation pass while the other GPU produces the final image from previously pre-computed G-buffers and the current user interaction. Although the overhead for buffer transfer between GPUs is considerable it would allow for view point manipulation at the rate of pre-computing the static scene parts. User navigation can be further improved by predicting view point changes using depth-image warping, an image-based rendering technique to generate new views from reference images considering per-pixel color and depth information (e. g. [MMB97]).

References

[DWS*88] DEERING M., WINNER S., SCHEDIWY B., DUFFY C., HUNT N.: The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics. In *Computer Graphics (Proceedings of ACM SIGGRAPH 88)* (1988), vol. 22(4), ACM, pp. 21–30.

[DWS04] DIETRICH A., WALD I., SLUSALLEK P.: Interactive Visualization of Exceptionally Complex Industrial CAD Datasets. Technical Sketch, ACM SIGGRAPH, 2004.

[MMB97] MARK W. R., MCMILLAN L., BISHOP G.: Post-Rendering 3D Warping. In *SI3D '97: Proceedings of the 1997 Symposium on Interactive 3D Graphics* (1997), ACM, pp. 7–16.

[PBMH02] PURCELL T. J., BUCK I., MARK W. R., HANRAHAN P.: Ray Tracing on Programmable Graphics Hardware. In *Proceedings of ACM SIGGRAPH 2002* (2002), Computer Graphics Proceedings, Annual Conference Series, ACM, pp. 703–712.

[SBW*07] SPRINGER J. P., BECK S., WEISZIG F., REINERS D., FROEHLICH B.: Multi-Frame Rate Rendering and Display. In *Proceedings IEEE Virtual Reality 2007 Conference* (2007), IEEE, pp. 195–202.

[SLRF08] SPRINGER J. P., LUX C., REINERS D., FROEHLICH B.: Advanced Multi-Frame Rate Rendering Techniques. In *Proceedings IEEE Virtual Reality 2008 Conference* (2008), IEEE. accepted.

[ST90] SAITO T., TAKAHASHI T.: Comprehensible Rendering of 3-D Shapes. In *Computer Graphics (Proceedings of ACM SIGGRAPH 90)* (1990), vol. 24(4), ACM, pp. 197–205.

[Wal07] WALD I.: On fast Construction of SAH based Bounding Volume Hierarchies. In *Proceedings of the 2007 Eurographics/IEEE Symposium on Interactive Ray Tracing* (2007), IEEE, pp. 33–40.

[WMG*07] WALD I., MARK W. R., GÜNTHER J., BOULOS S., IZE T., HUNT W., PARKER S. G., SHIRLEY P.: State of the Art in Ray Tracing Animated Scenes. In *STAR Proceedings of Eurographics 2007* (2007), Eurographics, pp. 89–116.

[WSB01] WALD I., SLUSALLEK P., BENTHIN C.: Interactive Distributed Ray Tracing of Highly Complex Models. In *Proceedings of the 12th Eurographics Workshop on Rendering* (2001), Eurographics, pp. 277–288.

[YCM07] YOON S.-E., CURTIS S., MANOCHA D.: Ray Tracing Dynamic Scenes using Selective Restructuring. In *Proceedings of the 2007 Eurographics Symposium on Rendering* (2007), Eurographics, pp. 73–84.