A Modular Infrastructure for Multi-Kernel Ray Traversals

T. Schiffer and D. W. Fellner

1Institut für ComputerGraphik & Wissensvisualisierung, TU Graz, Austria
2TU Darmstadt & Fraunhofer IGD, Germany


1 Introduction

Ray tracing on high performance computing hardware is a popular and active field of research. Our interactive ray tracing framework (http://www.cgv.tugraz.at/mrt) targets at modern massively parallel architectures and features different variants of ray tracing (Whitted-style, path tracing, ...) and applications. The modular system design is based on a wavefront tracing approach and ray loads with different characteristics (varying coherence, first-hit vs. any-hit) are submitted in batches to the ray tracing engine.

Our main objective is to investigate efficient ray traversal methods which consist of multiple kernels. Designing and implementing such highly optimized algorithms and data structures requires low-level tuning and a large amount of experimentation, which often conflicts with common software design principles like abstraction and encapsulation of data. Furthermore, the implementation should allow detailed evaluations in the research context while avoiding statistically significant performance impairment.

2 System Infrastructure and Discussion

The system infrastructure is based on a high performance computing module that is an object-oriented abstraction for NVidia’s CUDA and the OpenCL standard. This module provides a kernel language (similar to CUDA C or OpenCL static C++) including a runtime compilation framework, which allows generating optimally performing code on the target platforms. For managing host and device memory we implemented a dedicated fast memory manager based on the two-level segregated fit algorithm [Masmano et al. 2008]. We have experimentally evaluated the memory manager for memory allocation in various ray traversal algorithms and other parts of the ray tracing system showing a 10-25% decrease on overall running time compared to the native memory managers.

Our modular ray tracing engine is based on the high performance computing module and implements different ray traversal algorithms for GPUs (using bounding volume hierarchies):

- Monolithic ray traversals: our implementation of [Aila et al. 2012]
- Multi-kernel ray traversals: our modified version of [Garanza and Loop 2010] and [Schiffer and Fellner 2014]

An implementation of a ray traversal method is decomposed into algorithmic units represented by functor objects as shown in 1. Each unit solves a specific task (e.g. traversal step) denoted by an interface class. These interfaces can be implemented in multiple ways (depicted by Impl1 and Impl2) using different launch configurations, work-thread mappings (e.g. persistent threads) and other low-level aspects. This design not only simplifies the target-oriented experimentation and optimization of certain parts of algorithms, but also has several positive effects on the code quality: Functors encourage code reuse and help to maintain a modular high-level code structure, but encapsulate implementation details to keep the code base clean.

For research and development purposes it is essential to gather statistical performance data of an algorithm. For each functor interface a corresponding statistical object is defined that captures the relevant measurements. These objects and the functors are part of the scriptable object system and can thus be easily configured for different evaluations and benchmarks.

References


