SIGRAD 2016
M. Hayashi (Editors)

Towards Full-Scale Ray Tracing in Games

Afshin Ameri E. and Thomas Larsson

Mälardalen University, Sweden

Abstract

We discuss the current status when it comes to real-time ray tracing of games as a full-scale alternative to renderers based on rasterization. Modern games include massive geometry, beautiful graphics, and advanced rendering effects, and still, they run with a high and steady frame rate at high resolutions. We argue that to make ray tracing a viable option, significant further development is required in terms of improved algorithms and software libraries, as well as hardware innovations that will greatly benefit the average gamer's hardware equipment.

1. Introduction

Ever since our PCs began to house parallel features such as instructions pipelining, superscalar execution, SIMD units, and multi-core CPUs, game developers and other programmers have nurtured the idea of having real-time ray tracing at their disposal. Ray tracing is often referred to as being embarrassingly parallel, since each pixel can be generated independently and in parallel with the others by firing independent rays that travels through acceleration data structures, which are either pre-built or reconstructed on the fly. There are ample of opportunities for data-parallel computations that can be distributed over a large number of cores, both during ray traversals and shading calculations.

During the last decade, researchers have tried to realize real-time ray tracing for the masses. Outspoken claims of real-time performance have not been rare (see e.g. [FGD*06, Bik07] and the references therein). It has turned out, though, that despite impressive algorithmic innovations and a substantial improvement of hardware performance, these solutions deliver real-time rates only when applied in certain contexts, such as using low resolutions, simplified lighting models, friendly scene compositions, or high-end hardware.

When will ray tracing become a full-scale alternative to rasterization in game rendering? What kind of hardware is required? What software libraries can make a real difference? These are difficult questions. The fact that rasterization is the predominant rendering technique in commercial games preserves status quo. Games have driven the GPU market to unprecedented levels during the latest 20 years. The introduction of programmable shader stages in the graphics pipeline in 2001 is the single most important step that has kept rasterization at the forefront of game development. Established and standardized APIs that undergo timely and profound modernization further reinforce rasterization as game developer's first choice.

Indeed, the modern GPU is excellent at data-parallel processing of both triangles and pixels, which makes rasterization blazingly fast. With resolution and frame rate standards such as 1920×1080 and 60 fps, it is very challenging to keep up with rasterization for any fundamentally different strategy. Modern rasterizers also use lighting models that incorporate anti-aliasing, shadows, reflections, ambient occlusion, etc. A Whitted-style ray tracer would simply not be enough. As more advanced global illumination effects are supported by game engines, sometimes by using hybrid techniques that include ray tracing ingredients, a competing real-time ray tracer must probably support some type of distribution ray tracing, path tracing, or photon mapping. The extreme workloads this put onto the hardware are well-known in off-line renderers used to produce animated movies.

There are some lights on the horizon, though. First of all, ray tracing naturally supports visibility queries to generate realistic secondary lighting effects: shadows, reflections, and refractions. By hierarchical index methods, the expected time complexity for ray shooting n primitives is $O(\log n)$ for fixed resolution. Furthermore, it is convenient to support a mix of different types of modelling primitives, and advanced effects can be added adaptively by shooting more rays. Rasterization techniques, on the other hand, give cruder approximations of such effects and usually require more parameter tuning to avoid rendering artefacts.

Another noteworthy aspect is that major hardware companies show a lot of interest in ray tracing by developing libraries, e.g., OptiX by Nivida [PBD*10], Embree by In-

Figure 1: *Example scene rendered in full HD by a simple ray tracer that uses the Embree kernel framework.*

tel [WWB*14], and FireRays by AMD. This is probably because graphics and visualization are becoming increasingly important, and ray tracing can also be used to show off the performance of their latest hardware inventions.

2. Experiment

What is the current performance when it comes to using high-performance ray tracing libraries? To get an idea of the rendering speed, we created a simple ray tracer using Embree v.2.9 [WWB*14] and Visual Studio 2013. Embree was initialized to use packets of 8 rays for all ray types. Image tiles (16×16) were rendered in parallel by using OpenMP multithreading with dynamic scheduling. Only a simple Phong illumination model was used. We ran our test on a PC with an Intel Core i7-5960X 3.00 GHz CPU (8 cores with AVX2 support) and 16 GB RAM.

The scene had a total face count of 517 600 and it was composed of a house (created by Herminio Nieves) and a Mercedes-Benz SLS AMG (downloaded from tf3dm.com). There was one directional light source, and some texture maps were used. Figure 1 shows an image of this scene which was rendered at a resolution of 1920×1080. In this particular case, our rendering performance was 7.8 fps, which corresponds to 34.2×10^6 rays/s. When we let the camera orbit around the house, the fps varied from 6.0 to 20.4 with an average of 10.8. We expect that the performance could be improved further by using more aggressively optimizing compilers, e.g., the Intel SPMD Program Compiler (ispc), and by additional code optimizations and parameter tuning, but we would probably not be able to reach 60 fps, assuming the same hardware setup is used.

3. Challenges

There is an interesting old joke that says: "Ray tracing is the technology of the future and it always will be!" To get high-quality images, we need accurate shadows and reflections from multiple light sources, as well as anti-aliasing and some ambient occlusion. Let's assume that a rough estimate of 100 rays/pixel would be enough. Now consider full HD resolution and a target frame rate of 60 fps. Furthermore, assume the computing resources are occupied by other computations concerning game physics, collision detection, artificial intelligence, etc., half of the time. This leaves about 8 ms for rendering each image. Under these circumstances, we need a ray tracing budget of around 25 billion rays/s. If we compare this with the performance reached here, or with the rendering speeds reported elsewhere [WWB*14, PBD*10], there is a substantial difference.

The road ahead requires improved algorithms and more efficient parallel hardware. Algorithmic improvements need to focus on more efficient data structures for handling general scenes with dynamic features and faster ways to reproduce global illumination effects. On the hardware side, strong cooperating heterogeneous chips are needed as well as highly optimized kernel libraries that fully utilizes the architectures, e.g., by online learning and autotuning for increased efficiency. In addition, the solutions have to be applicable on affordable hardware for the average gamer.

For 50 years, we have been able to rely on Moore's law for doubled transistor density leading to an exponential rate of performance improvement. It has been described as a "miraculous development", but the rate of progress has started to decelerate as the transistor size is reaching closer to its physical limit. This may seem like a threat against the high performance we are eagerly seeking, but it may in fact be the opposite; it may trigger innovation and lead to even better hardware designs. Interestingly, Imagination Technologies recently demonstrated that dedicated low-power ray tracing hardware can deliver 300 million rays/s.

Even so, there is still a big performance gap to overcome before ray traced games can take over. In fact, the performance requirement specified above are relatively modest in the sense that it is easy to imagine even higher image resolutions and a wish for more accurate global illumination techniques. Without doubt, full-scale real-time ray tracing will be a main challenge for many years to come.

References

[Bik07] BIKKER J.: Real-time ray tracing through the eyes of a game developer. In *IEEE Symposium on Interactive Ray Tracing* (2007), pp. 1–10. 1

[FGD*06] FRIEDRICH H., GÜNTHER J., DIETRICH A., SCHERBAUM M., SEIDEL H.-P., SLUSALLEK P.: Exploring the use of ray tracing for future games. In *ACM SIGGRAPH Symposium on Videogames* (2006), pp. 41–50. 1

[PBD*10] PARKER, BIGLER, DIETRICH, FRIEDRICH, HOBEROCK, LUEBKE, MCALLISTER, MCGUIRE, MORLEY, ROBISON, STICH: OptiX: A general purpose ray tracing engine. *ACM Trans. Graph. 29*, 4 (2010), 66:1–66:13. 1, 2

[WWB*14] WALD I., WOOP S., BENTHIN C., JOHNSON G. S., ERNST M.: Embree: A kernel framework for efficient CPU ray tracing. *ACM Trans. Graph. 33*, 4 (2014), 143:1–143:8. 2