Real-Time Ray Tracing Using Nvidia OptiX

H. Ludvigsen[1] and A. C. Elster[1]

[1]Dept. of Computer and Info. Science, Norwegian University of Science and Technology, Trondheim, Norway

Abstract

Modern GPUs with their several hundred cores and more accessible programming models are becoming attractive devices for compute-intensive applications. They are particularly well suited for applications, such as image processing, where the end result is intended to be displayed via the graphics card. One of the more versatile and powerful graphics techniques is ray tracing. However, tracing each ray of light in a scene is very computational expensive and have traditionally been preprocessed on CPUs over hours, if not days. In this paper, Nvidia's new OptiX ray tracing engine is used to show how the power of modern graphics cards, such as the Nvidia Quadro FX 5800, can be harnessed to ray trace several scenes that represent real-life applications in real-time speeds ranging from 20.63 to 67.15 fps. Near-perfect speedup is demonstrated on dual GPUs for scenes with complex geometries. The impact on ray tracing of the recently announced Nvidia Fermi processor, is also discussed.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Hardware Architecture—Parallel processing I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing

1. Introduction

Ray tracing makes it possible to render realistic shadows, reflections and glass-like objects, which requires trickery when only rasterization is used. In rasterization, one computes the area on the screen where each object is to be shown, but do not analyze light's impact on the scene. Ray tracing, however, provides this features "by nature", because it approximates how light actually behaves. The main drawback of ray tracing is its computational complexity. The computations have traditionally been done on the CPU, but modern graphical processing units (GPUs) with their several hundred cores and now also more accessible programming models, are attractive devices for compute-intensive applications. By off-loading the ray tracing calculations to a modern GPU, ray tracing is becomng viable for computer games and real-time visualizations. In addition, ray tracing has applications in optical and acoustical design, radiation research, volume calculations and collision analysis.

Ray tracing is parallelizable because each ray may be traced independently. There is typically one ray per pixel, so common 1024 x 1024 pixel images would require tracing 10^6 rays. Ray tracing is hence a very attractive application for the massively parallel newer GPUs. Nvidia thus recently (Sept. 2009) released OptiX, a ray tracing engine for their Quadro and Tesla GPUs. This paper describes our initial experiences with this engine for real-time ray tracing.

Traditionally, ray traced images are computed a priori. This process might take a couple of minutes or several days per rendered image. Real-time ray tracing (> 20 fps) facilitates that realistic graphics can be manipulated interactively, like in a computer game. Similarly, feedback could be given instantly during optical design, radiation research and the other areas mentioned above. Last, but not least, the realistic visual effects possible by ray tracing could be added to games and other real-time visualization applications.

2. Previous work related to real-time ray tracing

Both CPU and customized hardware were used until recently for real-time ray tracing. However, the performance obtained in both cases have not been satisfactory compared to rasterization. Wald et al. [WSBW01] presented a highly optimized CPU implementation where the algorithms take advantage of caches, SIMD instructions and coherence in image and object space. Their implementation outperformed the earlier ray tracers, and even rasterization with graphics hardware for complex scenes. In their simplest scene with 40 thou-

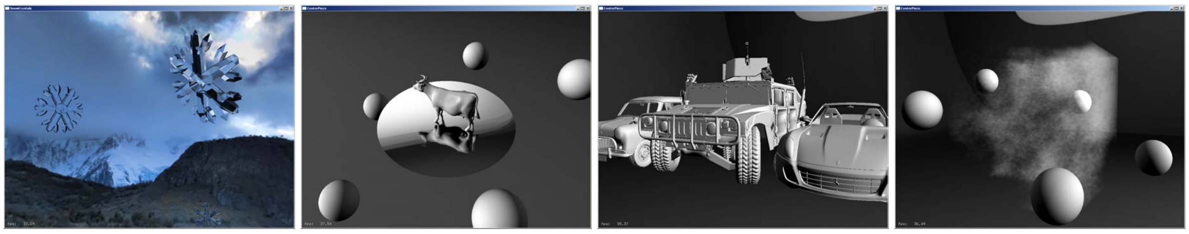

Figure 1: *Screenshots from the snow crystals scene and three centerpiece modes implemented in OptiX*

sand triangles, they obtained 1.8 fps at a resolution of 512 x 512 pixels on an 800 MHz Pentium III.

A different approach was used by Woop et al. [WSS05], who developed a programmable ray processing unit (RPU) chip specialized for real-time ray tracing. At only 66 Mhz, their prototype was capable of rendering a simple scene of 806 triangles at 21 fps, and a highly complex scene of 187 million triangles obtained 4 fps. However, this was at the modest resolution of 512 x 384.

In the recent years, attempts have been made to do ray tracing on the GPU, and the results have been promising. Purcell et al. [PBMH05] explained how ray tracing can be mapped to programmable graphics hardware and used a simulator to analyze the performance one might obtain on future graphics hardware. Their conclusion was that graphics hardware indeed look promising. Gunther et al. [GPSS07] followed up on this work by presenting a GPU ray tracer using optimized BVH-strctures to obtain 13.6 fps in a 2 million triangle scene at 1024 x 1024 pixels using an Nvidia Geforce 8800 GTX.

Some of the most recent work have obtained *true* real-time performance using commodity GPUs. Shih et al. [SCCC09] implemented a high performance CUDA-based ray tracer and obtained 30 to 43 fps in scenes ranging from 66 to 871 thousand triangles at 1024 x 1024 pixels on the older Nvidia Geforce 8800 GTS. Aila and Laine (Nvidia Research) [AL09] have developed a CUDA-based ray tracer which is hand-optimized at the assembly level and pushed performance towards (their) theoretical limit. Using the newer Nvidia Geforce GTX285, they obtained from 75 to 142 millions of primary rays per second, which at a resolution of 1024 x 768 correspond to 95 to 180 fps. Modern GPUs may also be used to do real-time implicit surfaces rendering [SN10]. Singh and Narayanan [SN10] also include several other recent related references.

3. Nvidia OptiX

OptiX [Nvi09c] is a recent programmable ray tracing engine that runs on top of Nvidia CUDA [Nvi09a]. It is a set of library functions for both graphics rendering or other applications that trace rays. The OptiX engine currently only runs on newer (GT200 core) Nvidia Quadro and Tesla cards. To use OptiX, the programmer writes *programs* that handle the various events of the ray tracing. These programs are really CUDA kernels, but are called programs in OptiX terminology. The events they handle include ray generation, ray hit, ray miss, etc. In the host code, the API is set up through a *context* structure that holds the configuration and components of the ray tracing. Such components include the previously mentioned programs, geometry that the rays hit and materials that define the surface properties of the geometry.

4. Our implementations using OptiX

Two ray traced scenes have been implemented, where one of them has several modes that each demonstrate how OptiX handles different applications of ray tracing. Some screenshots are given in Figure 1. Our snow crystals scene was implemented from scratch with transparent snow crystals which fall slowly across the screen. Our centerpiece scene depicts a centered object that changes with different modes of the scene. The object used include a cow model that comes with the OptiX SDK. The camera rotates around the centerpieces. Possible modes of the centerpiece scene are:

1. Phong shaded cow model and spheres
2. Reflective cow model and Phong shaded spheres
3. Reflective cow model and glass spheres
4. Cow model with diffuse reflection on floor
5. Cow model with diffuse shadow on floor
6. High definition car models (1 million triangle polygons)
7. Voxel map of cloud fractal

To compare our results to previous work on real-time GPU ray tracing, the polygon meshes from [SCCC09] and [AL09] were obtained, and OptiX used to ray trace them. The camera angle was adjusted to be approximately equal to the camera angles used in screenshots given in [SCCC09] and [AL09]. Note, however, that the results in [SCCC09] are on older hardware, so in this case, our results are as much about what scenes one can implement efficiently rather than fair comparisons. Screenshots of some of the scenes as rendered in the test bench, are given in Figure 4.

Currently most PCs support up to two GPU cards. We hence also tested OptiX with two identical Nvidia Quadro FX 5800 GPUs. The performance in fps was measured for

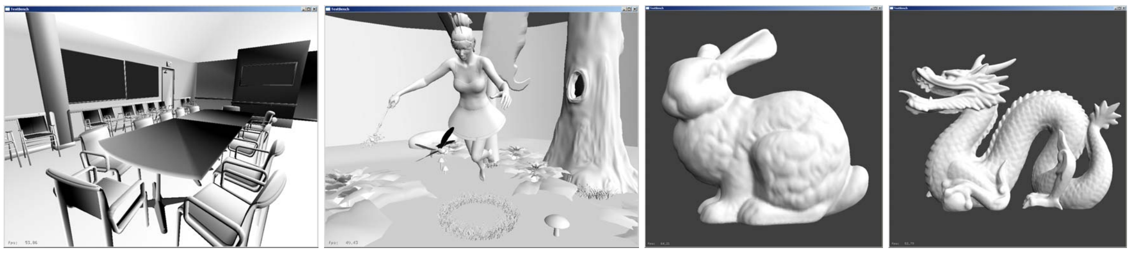

Figure 2: *Screenshots of conference, fairy, bunny and dragon scene from OptiX test bench*

some of the scenes that come with the OptiX SDK in addition to the scenes implemented in this project. The speedup was calculated as the ratio between performance with 2 and 1 GPUs.

All OptiX testing was done on a system with:

- GPU: Nvidia Quadro FX 5800
- CPU: Intel Core 2 Quad Q9550 2.83 GHz
- Memory: 4 x Corsair 2 GB DDR3 1333 MHz
- OS: Microsoft Windows XP 64 bit
- Compiler: Microsoft Visual Studio 2008

For all scenes, the performance was measured over 100 frames after a 3 second warm-up. The inverse of the average frame render time gives the fps. Table 1 summarizes the performance of the snow crystals and centerpiece scenes with its modes.

Table 1: *Performance in fps at 1024 x 768 pixels*

Scene	fps
Snow crystals	22.10
Centerpiece Phong	67.51
Centerpiece reflective	38.73
Centerpiece reflective and refractive	24.41
Centerpiece diffuse reflection	23.10
Centerpiece diffuse shadows	25.32
Centerpiece 1 million polygons	24.99
Centerpiece voxels	22.08

The main issue faced in our snow crystals scene is refraction and reflection of rays when they hit the snow crystals. At the initial intersection the ray is branched into two rays. And when the refracted ray hits exits the crystal, there is another branching into two rays. This branching imposes a performance penalty, especially when the crystals cover much of the screen area. The performance obtained is real-time with an average of 22 fps. A major problem is how the fps varies depending on what happens in the scene. When a large crystal is close to the screen, the fps is low at around 20 fps. When this crystal exits the screen, the fps spikes up to around 40. This behaviour results in unstable performance, and is one of the major drawbacks of the ray tracing algorithm.

In our centerpiece scene, the performance varies for each of the modes. All of the modes offer real-time performance at 22-25 fps, but the cheaper Phong shading and reflection only mode results in 67.51 and 38.73 fps, respectively. Diffuse reflection and diffuse shadows were a disappointment performance-wise. In both cases the ray branching is set to only into 4 new rays, but the fps is nevertheless barely real-time. This shows how ray branching is the major challenge of ray tracing performance. However, the scene consisting of three car models and a scooter which has a total of 1 million polygons, shows that OptiX is indeed capable of rendering real-time scenes with high definition and complex models. All representable for real-life objects. The voxel mode has "only" 250,047 voxels, but does not benefit from primitives being covered by other primitives such as in the mode with 1 million polygons. At an fps of 22.08, this shows that OptiX is capable of rendering voxels scenes in real-time.

4.1. Performance vs. optimized GPU ray-tracers

Table 2 shows the number of triangle polygons and measured OptiX performance of scenes from [AL09], and Table 3 shows the same for scenes from [SCCC09]. Also shown in these tables is the performance the authors of [AL09] and [SCCC09] obtained with their ray tracer.

Table 2: *Triangles and performance in Mray/s of scenes in [AL09] at 1024 x 768 pixels*

Scene	Conference	Fairy	Sibenik
Triangle polygons	282,759	174,117	80,133
Mray/s OptiX	28.22	20.69	38.12
Mray/s [AL09]	142.2	74.6	117.5

Table 3: *Triangles and performance in fps of scenes in [SCCC09] at 1024 x 1024 pixels*

Scene	Bunny	Sponza	Dragon
Triangle polygons	69,451	66,454	871,414
Fps OptiX	49.89	28.16	36.43
Fps [SCCC09]	45.30	42.47	31.88

As seen in Table 2, our results using OptiX are 3-4 times slower than the implementation in [AL09]. The hardware in both cases is the GT200 generation GPU. This shows that

OptiX has potential for much higher performance. An explanation of the discrepancy can be the flexibility of OptiX, and the fact that the implementations in [AL09] was hand optimized at the assembly level for performance only.

Table 3 shows that our OptiX implementations outperform [SCCC09] slightly in the bunny and dragon scene, but lags behind by about 30 % in the sponza scene. However, the GPU used in [SCCC09] is a Nvidia Geforce 8800 GTS that is several generations older and substantially slower than the Quadro FX 5800 used in our test bench. Again, one would assume that the implementation in [SCCC09] is heavily optimized and not as flexible.

4.2. Multiple GPUs

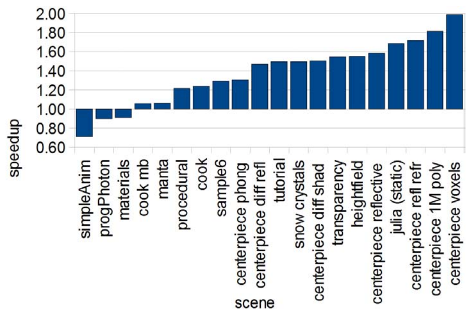

Figure 3: *Multiple GPU speedup in various scenes*

Figure 3 compares our results on 2 GPUs with 1 GPU in various scenes. At the lower extreme, there is actually a *slowdown* compared to using 1 GPU. However, at the other end of the chart, we achieve an almost perfect 200 % speedup. Common for the scenes that speed up well is that they spend a lot of time on GPU computations compared to other tasks such as image display, data transfer and CPU computations. Hence in order to take advantage of the computational power of the GPU, scenes need to have enough computational complexity that can be done on the GPU, as can be seen in our centerpiece scene.

5. Conclusions and future work

This paper studied implementations of animated real-time scenes that represent actual real-life application areas for ray tracing. Nvidia's recently released OptiX ray tracing engine allows users to harness the power of modern GPUs. Our results demonstrate that several ray tracing applications may be performed in real-time on the GPU using OptiX. All of our test cases gave real-time speeds ranging from 20.63 to 67.51 fps on 1 GPU. Our dual GPU results indicated that OptiX can give near-perfect speedup on multiple GPUs for scenes with enough computational complexity. Even though OptiX has showed to be capable of real-time ray tracing, our initial implementations were slower (3 to 5 times) than some hand optimized ray tracers such as in [AL09], indicating room for improvement. Our results do, however, demonstrate that OptiX is a flexible engine capable of real-time ray tracing on both single and multiple Nvidia GPUs.

A major difference between CPUs and GPUs is that GPUs cannot do branching efficiently. Efficient branching is important in ray tracing since the directions the rays are reflected and refracted is not known in advance. Fortunately, newer GPUs such as the Nvidia CUDA architecture handle branching better than previous generations. NVIDIA recently announced their new Fermi [Nvi09b] GPU which includes L1 and L2 cache, better double precision number support and concurrent kernel execution. The on-chip GPU cache should be beneficial for ray tracing since previously read or spatially coherent data can then be quickly accessed when traversing acceleration structures. When it is publicly available, its impact on OptiX and real-time ray tracing performance should be investigated.

Future ray tracer designs should also incorporate the ideas from recent work such as [AL09] and [SCCC09]. Incorporating ray tracing into full-scale applications such as medical and seismic visualizations, should also be investigated. Finally, we would like to thank Nvidia and other sponsors of our HPC-lab.

References

[AL09] Aila T., Laine S.: Understanding the efficiency of ray traversal on gpus. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009* (New York, NY, USA, 2009), ACM, pp. 145–149. 2, 3, 4

[GPSS07] Gunther J., Popov S., Seidel H.-P., Slusallek P.: Realtime ray tracing on gpu with bvh-based packet traversal. *Symposium on Interactive Ray Tracing 0* (2007), 113–118. 2

[Nvi09a] Nvidia Corporation: *CUDA Programming Guide version 2.3.1*, August 2009. 2

[Nvi09b] Nvidia Corporation: *Fermi Compute Architecture Whitepaper*, 2009. 4

[Nvi09c] Nvidia Corporation: *OptiX Ray Tracing Engine Programming Guide version 1.0*, September 2009. 2

[PBMH05] Purcell T. J., Buck I., Mark W. R., Hanrahan P.: Ray tracing on programmable graphics hardware. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses* (New York, NY, USA, 2005), ACM, p. 268. 2

[SCCC09] Shih M., Chiu Y.-F., Chen Y.-C., Chang C.-F.: Real-time ray tracing with cuda. In *ICA3PP '09: Proceedings of the 9th International Conference on Algorithms and Architectures for Parallel Processing* (Berlin, Heidelberg, 2009), Springer-Verlag, pp. 327–337. 2, 3, 4

[SN10] Singh J., Narayanan P.: Real-time ray tracing of implicit surfaces on the gpu. *Visualization and Computer Graphics, IEEE Transactions on 16*, 2 (march-april 2010), 261 –272. 2

[WSBW01] Wald I., Slusallek P., Benthin C., Wagner M.: Interactive rendering with coherent ray tracing. In *Computer Graphics Forum* (2001), pp. 153–164. 1

[WSS05] Woop S., Schmittler J., Slusallek P.: Rpu: a programmable ray processing unit for realtime ray tracing. *ACM Trans. Graph. 24*, 3 (2005), 434–444. 2