Adaptive Control of Visual Effects for Frame Rate Maintenance in Real Time Ray Tracing

Paulo M. F. Andrade
Media Lab – UFF

Esteban W. Clua
Media Lab – UFF

Alselmo A. Montenegro
Media Lab – UFF

Paulo A. Pagliosa
DTC - UFMS

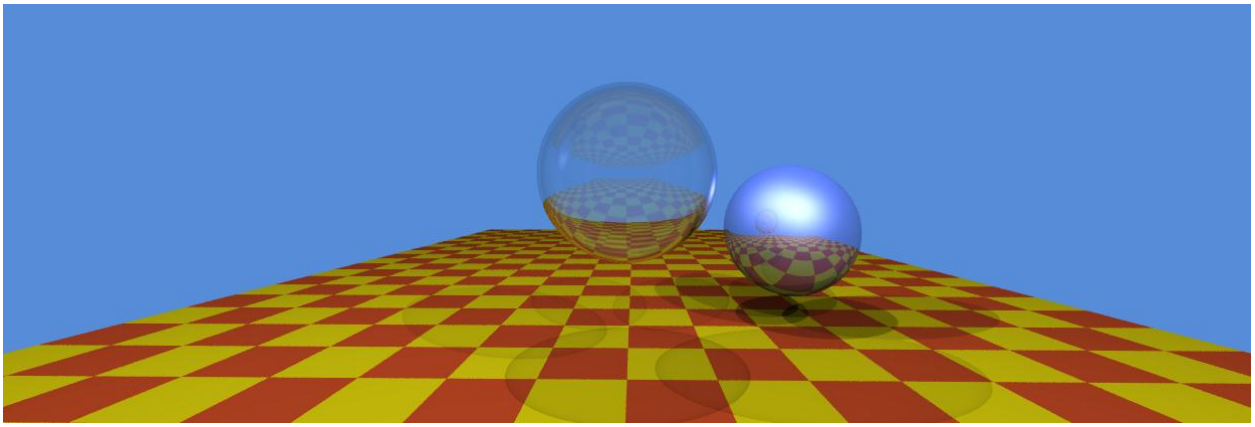

Figure 1: A Whitted-style render of a glass and a metal sphere and five lights.

Abstract

This paper presents strategies to control frame rate performance for a Real Time Ray Tracing engine based on OptiX framework by dynamically altering and adjusting visual effects based upon information directly collected from the scene. Performance studies were done to investigate the impact of the number of lights and the use of reflexive or translucent materials in order to validate the concept.

Keywords: Real Time Ray Tracing, CUDA, OptiX.

Authors' contact:
{pandrade,esteban,anselmo}@ic.uff.br
pagliosa@dct.ufms.br

1. Introduction

When compared with real-time raster applications, Real Time Ray Tracing (RTRT) usually is not feasible for complex interactive applications such as 3D games. In general, ray tracing is better suitable for rendering realistic static images. Nowadays, multicore architectures has become commonplace. Since ray tracing is considered an embarrassingly parallel task [Bigler 2006], RTRT using parallel processing has became an important a field of study.

RTRT has many advantages when compared with common raster methods. Considering only visual aesthetics, Ray Tracing is better suitable for shadow generation, reflections, and refractions and has elegant solutions for fog, glow and particles [Bikker 2007].

State of the art software for ray tracing usually prioritizes visual realism, seeking to simulate physical optics, in order to create photo-realistic appealing images and effects. For real time applications, photo-realism can be reduced in favor of high frame rates. This is observed in games, where the hardware is identified by the graphic API and, based on its configuration, the software chooses to activate or deactivate specific effects. Considering first person shooters, were frame rates directly influence the experience of the gamer, it is usually better to avoid low frame rates and sacrifice realism, even if most of the time the level design of the game allows better visuals with good frame rates.

In this paper, we discuss strategies to automatically turn off or modify visual effects in order to guarantee acceptable frame rates for RTRT applications. Considering the per-pixel nature of ray tracing calculation, where, in most cases, every pixel color can be calculated independently and the massive parallel processing power offered by GPUs [Kirk and Hwu 2010] we propose strategies to dynamically turn on/off or modify effects for specific pixels of the scene, in order to maintain decent frame rate performance. We also developed a RTRT prototype, based on nVidia OptiX framework [Parker at al. 2010], in order to test the hypothesis.

2. Related Works

One of the first relevant studies of RTRT is the OpenRT project [Dietrich et al. 2003]. In 2004, the project team succeeded in porting the game Quake3 to OpenRT, achieving an average of 20 frames per second, with the game running on a cluster of 20 state of the art PCs at that time. The OpenRT is an API similar to OpenGL, but following a traditional Ray Tracing pipeline. Unfortunately the project seems to have stopped.

Another important project is Aruana [Bikker 2007], a RTRT Open Source Renderer developed by Jakko Bikker. Arauna doesn't use any special GPU optimization; it only use the CPU processing power, but explore the capabilities of multicore architectures. Unlike OpenRT, Arauna is directed to personal computers. Some games were developed using Arauna as proof of concept.

3. OptiX to Ray Trace Images

In order to test the strategies, we choose to use NVIDIA's® OptiX™ [Parker at al. 2010], ray tracing engine and API. OptiX has a modern architecture, has a programmable pipeline and is build upon the use of massive parallel processors of GPUs. Currently, OptiX target architecture are state of the art NVIDIA GPUs.

Even being a generic ray tracing engine, OptiX can be used to ray trace off-line and real time images. For this project, our focus is RTRT and we used the Whitted [1980] ray trace method to render images in real time. OptiX is also a domain-specific just-in-time compiler that generates custom ray tracing kernels for CUDA [Nickolls 2008] capable GPUs. The custom GPU kernel combines user-supplied programs and monolithic kernel architecture for general computation of the ray tracing [Parker at al. 2010].

4. Implementation

To test the strategies and measure the performance, a Whitted RTRT was built with OptiX. A simple scene of a sphere and a plane was used for testing. The tests were run in a Core 2 Duo notebook with 3 GB of Ram and a GeForce 8600M GT with 256 MB of memory.

In order to implement a Whitted-style RTRT, OptiX programs were created to affect specific parts of the pipeline. The first step was to define the Context and the elements of the Context. The Context is defined in the Host code (CPU) and is an instance of a running OptiX engine. The Context binds the Ray Generation programs for color and shadows calculation; binds the Exception program to treat unexpected errors; binds the Ray Generation program; binds the Miss program that will define the background color (light blue in the examples); binds the Bounding Box and Intersection programs for every geometry of the scene and binds the Closest Hit and Any Hit programs for every material used in the scene. After these bindings, one or more instances of the defined geometries are created and bounded to the corresponding materials. One geometry instance can have one or more materials bounded to itself. The final step is the geometry grouping. Geometry grouping is used to organize the scene and define its acceleration structure. Every group has its own acceleration structure, which allows the usage of specific structures for static and movable objects.

5. Strategies for Real Time Frame Rate Maintenance

Considering the optimal scenario where there are an unlimited number of cores available for a parallel ray tracer program, every ray can be computed independently and in parallel. In this situation, the frame rate would be determined by the slowest ray. But, for scenes with effects like reflections and refractions, secondary rays must be created after the primary ray collision with elements of the scene. So, the frame rate performance is determined by the slowest primary ray computation plus the time consumed by the sequence of secondary rays created after the primary ray. Considering this simplistic scenario, in order to control performance, is necessary to act upon the secondary rays.

A ray that hits a translucent material will generate more than one secondary ray in sequence, in order to correctly treat the transparency. These secondary rays can be generated recursively until a limit is reached. Figure 2 shows the rendering of a hollow glass sphere using zero (ray casting) to five secondary rays generated in sequence, after the primary ray. A shadow ray is also generated for the scene.

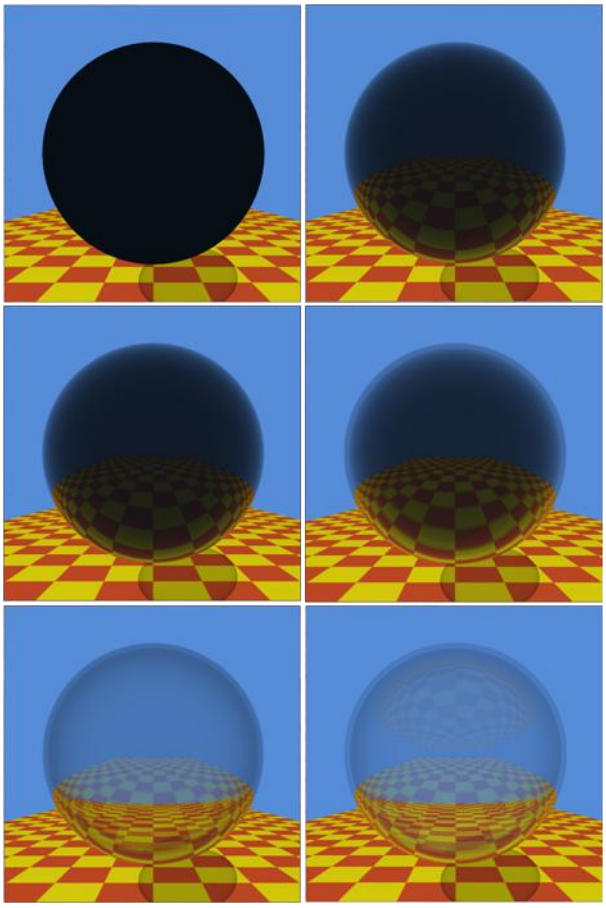

Figure 2: Six images of the same glass sphere rendered with an increase number of secondary rays.

Table 1 shows the frame rate for the six images and the slowdown percentage relative to the previous image. The frame rate decreases with the size of the sphere relative to the size of the image.

Table 1: Frame Rates from 0 to 5 secondary rays.

N. sec rays	0	1	2	3	4	5
Frame Rate	27.5	15.0	10.1	5.8	3.2	1.8
Slowdown	-	54%	67%	57%	55%	56%

The results in Table 1 indicates that the impact of secondary rays for the translucent material reduce the frame rate performance in more than 50%. Since the sphere occupies around 50% to 60% of the scene, this results point to a linear decrement in performance with the number of secondary rays generated from a single primary ray.

5.1 Effects of the Size and Distance of Objects

For every ray traced, is easy to determine the distance from the origin to the object intercepted by the ray. Using the distance information and the size of the object it is possible to control the recursive level of secondary ray generation for the specific object, since details in distant objects are less perceptible. Shadows and reflections can also be ignored for distant objects in order to improve performance.

The size and distance of objects to the camera defines the percentage of the rendered image that the object will occupy. Considering ray tracing, this directly impacts the performance, since computational expensive objects will need more resources if more pixels of these objects are to be generated.

In order to measure the distance/size impact, we measure frame rate for four distances of an object to the camera. All the distances are a multiple of the same value, what means that, if all the spheres are in the same scene, for every sphere, the nearest sphere has always the same distance, considering the Z axis. The other axis has their values fixed.

When the distance increases horizontally, the size of the object reduces proportionally for its weight and height. This size reduction directly impacts the number of secondary rays traced. But the measured results do not point to a proportional increase in frame rates. It seems that the recursive secondary ray generation affects more the performance than the number of primary rays traced.

For the tests we use a six level recursive secondary ray and one light. Table 2 presents the frame rates and speedup obtained when the sphere moves away from the camera and Figure 3 shows the sphere rendered in four different distances from the camera.

Table 2: Frame rates for the four distances of the sphere from the camera.

Distance	1	2	3	4
Frame Rate	1.12	2.27	3.29	4.13
Speedup	-	102%	45%	25%

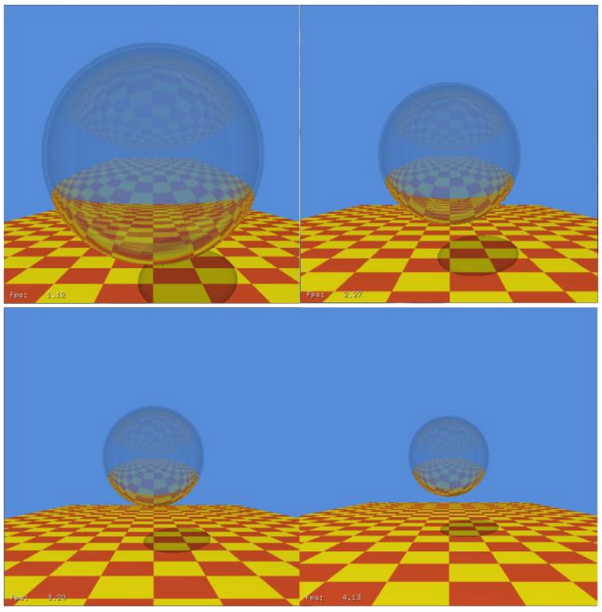

Figure 3: Four images of the same glass sphere rendered at different distances from the camera.

5.2 Material substitution and Number of Lights

Another strategy employed in order to improve performance is to substitute materials that are more demanding for simpler ones. Since is possible to associate more than one material to the same surface, using the intersection distance or the relative size of the object to the viewport, is possible to select the material to be used. Since the tests are made at pixel level, parts of the surface can have different materials and change the materials in real time.

In the Whitted example, we only consider direct lights. For direct lights, a loop is done for every light in order to compute the corresponding shadows.

Figure 4 shows a glass sphere and a metal sphere with five colored lights. The shadow was colored to help identify the corresponding lights. Table 3 shows the frame rates and slowdown relative to the number of lights for both the glass and the metal sphere.

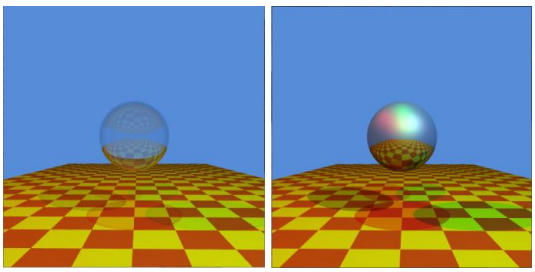

Figure 4: A glass and a metal sphere with five lights.

Table 3: Frame Rate for 1 to 5 lights.

N. Lights	**1**	**2**	**3**	**4**	**5**
Metal Sphere	17.6	13.7	11.3	9.7	8.4
Slowdown	-	77.8%	82.5%	85.8%	86.6%
Glass Sphere	3.5	3.1	2.8	2.6	2.4
Slowdown	-	88.6%	90.3%	92.9%	92.3%

Direct lights don't significantly impact scene performance considering the test results presented in Table 3. Since the effect of lights is not calculated recursively, as opposed to reflection and refraction, this indicates that a loop inside the thread is much more desirable than the use of recursion. This can partially be explained by the fact that the light generation loop is inside the thread execution while recursive rays probably generate new threads.

5. Conclusions and Future Work

Ray trace performance is influenced by many factors, such as image resolution, scene complexity and visual effects. For static images, is important that the visual appearance is sharp and impressive in every detail. The same can't be said for real time applications like video games, since the action can easily hide visual details. These factors have different impact in the overall frame rate performance, according to the tests. Our tests shows that the recursive use of rays to implement translucent materials have the biggest impact in frame rate performance. In the opposite side, lights have the smallest impact. Five lights in a scene reduced the performance to a little more than 50% for the metal sphere and less than 25% for the glass sphere. Since lights can be easily controlled to affect only some objects, according to pre-defined conditions, the use of direct lights RTRT can easily create interesting effects with low impact in performance.

Another interesting characteristic is the possibility to change the object material in specific pixels of the object. This feature can help produce interesting animated effects with little processing cost and implementation effort. Dynamic change of materials can also be employed to chance costly materials in distant or unimportant objects to less demanding materials. Since this change can be bind to the distance of the ray to the camera, this could be used, for example, to chance transparent windows in a building for a texture, when the window is far from the camera. This texture could be the previous render of the transparent material, allowing distant windows to look different from different angles, when the camera moves away from then.

Interactive tests done with different materials, number of lights, shadows and with the camera moving in many directions, with the material and shadow dynamically changing according to the distance between the camera and the objects proved that is possible use the adaptive control of visual effects in order to maintain the frame rate performance. Since the impact of using recursive rays is strong in the overall performance, a solution to smoothly maintain frame rates will need to dynamically control the recursion level and implement other mechanisms to avoid subtle increase and decrease of frame rates, making the fluctuation of frame rates annoying the user.

The results point to a good number of future works. Considering the recursive use of secondary rays, implementing transparent materials without the use of recursion could dramatically improve the performance. Another future work consists in build a hybrid real time render engine, where some effects are generated by the GPU using ray tracing, and others by the CPU, using raster techniques.

Finally, ray tracing allow many other effects like soft shadows, ambient occlusion, depth of field and motion blur. The impact of these effects must also be studied in order to offer more possibilities towards high quality RTRT images using many visual effects without compromising performance.

References

BIGLER, J., STEPHENS, A., PARKER, S. G., 2006. Design for parallel interactive ray tracing systems. *IEEE Symposium on Interactive Ray Tracing.*

BIKKER, J., 2007. Ray Tracing through the eyes of a game developer. *IEEE Symposium on Interactive Ray Tracing.*

PARKER, S. G., BIGLER, J., DIETRICH, A., FRIEDRICH, H., HOMEROCK, J., LUEBKE, D., MCALLISTER, D., MCGUIRE, M., MORLEY, K., ROBISON, A., STICH, M., 2007. OptiX : A general purpose ray tracing engine. *In: ACM Transactions on Graphics (SIGGRAPH 2010 Proceedings).*

KIRK, D. B., HWU, W. W. 2010. Programming massively parallel processors – A hands-on approach. NVidia : Morgan Kaufmann.

WHITTED, T. 1980. An Improved Illumination Model for Shaded Display. *Commun. ACM 23*, 6, 343–349.

DIETRICH, A., WALD, I., SLUSALLEK, P., 2003. The OpenRT Application Programming Interface - Towards A Common API for Interactive Ray Tracing. *Proceedings of the 2003 OpenSG Symposium.* Darmstadt, Germany, April 1-2, pages 23-31, Published by the Eurographics Association.

NICKOLLS, J., BUCK, I., GARLAND, M., and SKADRON, K. 2008. Scalable Parallel Programming with CUDA. Queue 6, 2 (Mar. 2008), 40-53. DOI= http://doi.acm.org/10.1145/1365490.1365500.