A Heuristic to Selectively Ray Trace Light Effects in Real Time

Paulo M. F. Andrade — Media Lab – UFF
Thales L. Sabino — Media Lab – UFF
Esteban W. G. Clua — Media Lab – UFF
Paulo Pagliosa — UFMS

Abstract

Raster rendering is often associated to real time rendering, while ray tracing is usually considered when quality is more important than speed. Recent works [Hertel and Hormann 2009; Lauterbach 2009; Sabino et al. 2011] offer ray trace effects in raster based real time renderings, in order to improve image quality, where ray tracing strategies seems to provide better results. Unfortunately, for many applications like 3D games, simulation and virtual reality, not only the image generation frame rate must be high, but it also must be consistent, i.e., the frame rate must not drop too much during the experience. In order to have a solid frame rate performance, level and environment designers carefully plan the scenario, in order to have a fluid, real time experience. Differing from raster based renderers, where the number of polygons to render is the critical factor for performance, ray trace payload is more affected by the number of rays that must be generated in order to render the scene. Since the number of rays can be influenced by multiple factors like number and location of the lights and surface characteristics, we propose a heuristic that use a strategy to limit the number of rays generated in every frame, while still improving the visual quality of a real time raster rendering. The heuristic can choose the best candidate objects to receive or generate ray trace effects, without impact the overall performance of the renderer.

Keywords: ray tracing, rasterization, OptiX^{TM}, GPU, hybrid rendering, real-time, rendering, deferred shading

Authors' contact:
paulo@andrade.com; tsabino@ic.uff.br;
esteban@ic.uff.br; pagliosa@facom.ufms.br

1. Introduction

In the search for photorealism in raster based real time rendering, programmers created many clever tricks to simulate global illumination effects, like cube maps for reflections and shadow maps to simulate areas occluded by objects. Different from raster based rendering, ray tracing offer more accurate and complex global illumination effects. Effects like complex shadows, lens and mirror like effects, caustics and ambient occlusion are common effects for most of ray trace renderers. The reason is that ray trace mimics the way light works in the real world. Even today, many visual effects created by ray trace renderers cannot be done by raster based renderers in real time.

With the recent advances in parallel computing, researchers started to investigate the possibility of a real time ray trace renderer using the massive power of parallel computing. Most of these studies are motivated by the observation that the basic ray trace algorithm is easily parallelizable. One of the first experiments of real time ray trace (RTRT) using parallel processing is the OpenRT Project [Dietrich, A et al. 2003]. In 2004, the project team succeeded in port the game Quake 3 to OpenRT, and the game achieve 20 frames per second on a cluster of 20 state of the art computers. Since then, many other projects examine the possibilities of RTRT, with different strategies and results [Seiler et al. 2008][Bikker [S.d.]]. Up to now, even the best RTRT renderer can not compete in visual quality and speed with a modern real time raster based renderer in any given scenario.

Since the visual quality of a state of the art real time raster based renderer is still far from possible for a pure RTRT renderer, researchers started to investigate hybrid solutions, where some effects are ray traced in a predominantly raster based scene [Hertel and Hormann 2009; Lauterbach 2009; Sabino et al. 2011]. The main problem with hybrid solutions is guaranteeing a steady frame rate. With a raster only solution, the designer can interactively test the environment and change elements in order to improve the frame rate. The environment designer can reduce the number of polygons in the viewport, moving objects to other places, or by simplifying objects in order to reduce the number of polygons. In a hybrid raster and ray trace render, the challenge to improve performance is bigger, since the overall performance is not only affected by the number of polygons to raster, but also, by the number of rays to be traced at any given time.

In this paper, we propose a heuristic to dynamically select objects to be ray traced in a hybrid render, in order to deal with resource constraints. The heuristic also prioritizes effects and elements that most contribute to the visual experience of the user.

2. Related Work

Since a pure RTRT renderer, even using current parallel architectures, can not compete in speed with state of the art raster renderers, other approaches were used to increase frame rate. Some approaches tried to divide the workload between the CPU and the GPU, like the approach proposed in [Chen and Liu 2007], where a GPU accelerated rasterization with Z-buffer is used to determine the first ray-triangle hit of eye rays (primary rays). Secondary rays are generated using the CPU in order to provide global illumination effects.

Approaches like that worked but could not compete in quality with state of the art renderers.

Another strategy employed in hybrid render is use ray tracing only for visual effects that can not be easily done or are slower in a raster only renderer. Hertel and Hormann [2009] uses a kD-tree accelerated ray tracer to determine shadow-ray intersections, in order to improve the quality of highly detailed shadows. Lauterback [2009] also use ray tracing in to improve the quality or hard and soft shadows using a similar approach.

With the introduction of multiple render targets in both DirectX 9 and OpenGL 2.0, developers started to use a shading strategy denominated deferred shading [Thibieroz and Engel 2003] to enhance visual quality or real time rendering, by implementing a post-production pass. With the possibility of render in passes, researches started to utilize a pass to include ray tracing visual effects in the rendering pipeline. Wyman and Nichols [2009] use a ray tracing pass to create superior caustic effects while Cabeleira [2010] and Sabino et al. [2011] use ray tracing to include accurate reflections and refractions.

None of this approaches deal with the performance challenges that combining different render strategies available for developers and designers.

3. Hybrid Rendering using Deferred Shading

Most of today’s state of the art real time renderers use deferred rendering (also called deferred shading). The basic idea is to compute all the geometry visibility tests before any light computation (shading) happens. By separating the geometry rendering from the light processing, this render steps allows shading process to occur only in visible polygons, avoiding multiple light computations for the same pixel, a problem that must be treated in forward rendering approaches. The visible geometry determination can be compared to the primary ray phase of a ray tracer, where eye rays are projected in the direction of the scene, crossing a view plane defined by a grid of pixels. The rays collide with the object and the collision result in information of the geometry that will determine de color of the pixel.

During the first render pass, other information can also be computed like z-depth, normals and texture coordinates. This information is called G-Buffer data and is stored in memory buffers called Multiple Render Targets (MRTs), to be used in subsequent render passes.

With the information corresponding to the primary ray intersection and the corresponding geometry, rays for shadows, direct and indirect light, refractions, reflections, caustics and other effects can be generated and added to the already generated data in the MRTs, according to the way the final image must be generated.

The deferred rendering approach in a hybrid rendering allows to define which visual effects should be generated with ray tracing and which effects should be generated using other strategies.

In order to evaluate the heuristic, we use the hybrid real time renderer developed by Sabino [2012]. Sabino’s renderer use NVidia OptiX^{TM} [Ludvigsen and Elster 2010; Parker et al. 2010] to deal with the ray trace stage of the renderer. The real time render pipeline has four stages: deferred rendering and primary ray resolution, shadows, reflections and refractions, and the final composition.

3.1 Deferred Rendering and Primary Ray Resolution

In this stage, after all the data is stored in the GPU memory, a deferred shading pass is calculated in order to fill the G-Buffer. The G-Buffer has information of the visible geometry for the other phases.

3.2 Shadows

With the information stored in the G-Buffer, OptiX^{TM} create shadow rays for every light source.

3.3 Reflections and Refractions

Still with the information stored in the G-Buffer, OptiX^{TM} compute reflections and refractions.

3.4 Composition

The composition stage is the final stage of the pipeline, where all the images generated by the other steps are combined in order to produce the final image.

4. A Heuristic for Ray Tracing in Hybrid Rendering

Even for the simplest effects, ray tracing could be a challenging effort and can seriously drain the processing resources, if not used with caution. One of the reasons of Its high cost is the recursive nature of the ray tracing algorithm, in order to generate some global illumination effects. For example, light rays bouncing from surface to surface, generates indirect illumination. The number of light rays bounces and the sequential nature of the bounces can strongly influence the render pipeline. A common way to control render time in offline ray trace rendering is to establish a limit in both the number of ray bounces and the number of secondary rays generated by every bounce. Depending on the surface characteristics, a ray collision can produce more than one new ray.

In order to control de performance of the hybrid renderer, a time constraint must be defined for the ray trace stage of the rendering pipeline. With this time constraint, the heuristic must choose which objects should be traced in order to achieve the best visual appearance.

4.1 Object Selection

Considering only which objects should be involved in the ray tracing phase, we can define the heuristic as: “given X objects, select the Y most relevant objects that can be traced given a time limit T.

The reason behind the idea of choosing a subset of objects that best contribute to de visual experience came from the real world perception that when images change constantly, as when we drive a car or walk in the street, the mind ignores many visual elements. This is the reason we have orientation signs in the streets. Signs call attention to inform about something relevant. In a first person shooter game or a driving simulator, the faster the experience is, the less is the perception of detail in a scenario.

Considering the way the human vision works, it is reasonable to assume that objects near the center of the field of view are more important the objects far from it. The same can be said for objects near the observer. Another observation is that according to environment conditions (weather, indoor, under water, for example), some objects cannot be visually improved by ray tracing effects. With this observation, the heuristic can be expanded to “given X objects, select the Y objects nearest from the center of the field of view and from the observer, that better contribute to the visual experience considering the ray trace effects to be applied, and that can be used in the ray tracing pass considering the time constraint T”.

4.2 The Heuristic

The proposed heuristic has four phases, two fixed phases that happen before the real time rendering, and two phases that happen for every frame.

The first phase, called pre-production phase, consist in identify and select the objects and their relative effects that must be used in the ray tracing pass, for a given scenario. This information could be defined by the designer or generated by an algorithm that analyzes every object for their characteristics and relationship to the other elements in the scene.

The second phase is the graph build phase, where a selection graph is build in order to help determine the best candidates to ray trace.

During the pre-production phase, every object receives an importance factor (K). This factor defines how important is this object in respect with the others given some scenario conditions and characteristics. Also, every object has an initial visibility cost (V), and the corresponding estimated number of rays to be used to generate each visual effect related to the object (Q). (Q) could be a list of values for each object, with each value corresponding to an estimated cost for every effect, or can be the sum of costs involved to produce all the visual effects.

Visibility (V) is defined by the average area (A) of the 2D projection of the object in the view plane multiplied by the distance of the center of the 2D projection to the distance of the view plane (P), divided by the distance (D) of the object from the view plane in the 3D space. The higher the object distance, less visible the object is. The visibility equation is presented in equation 1. The total cost (C) for a given object is presented in equation 2.

$$V = \frac{(A * P)}{D} \tag{1}$$

$$C = V * Q \tag{2}$$

The object center to view plane center (P) is a value between [0,1], where 1 corresponds to the center of the projection be in the center of the field of view and 0 means that the object is out of the field of view.

To select each object is also necessary to calculate the relevance factor (R), where R is based on the factor that the object was selected to be traced in the previous frame (S). Is a binary variable, where 1 means that the object was previously selected, and 0 means the it was not. The importance of select previously selected objects is also a fixed factor, according to the scenario and is defined by the variable (I). Equation 3 is the object relevance equation.

$$R = (S * I + V) * K \tag{3}$$

All the equations are calculated for each object, in order to update the selection graph, in every frame.

Table 1 present all parameters discussed before and inform if the parameter has their value constant or variable during the render phase.

Table 1: Equation parameters

Par.	Definition	Cons	Var.
K	Object relevance among the others	X	
V	Object visibility		X
Q	Estimated number of secondary rays	X	
C	Processing cost		X
A	Projected area		X
P	Distance from the view plane center		X
D	Distance from the observer		X
R	Relevancy		X
S	Previously selected		X
I	Previously selection relevance	X	

The third phase happens during the render phase, when the GPU receives the selection graph. Every node in the graph is represent an object to be traced and every node point to the second node with cost (C) smaller than the cost of the previous node, but larger than the costs of the other nodes already in the graph. Every node also points to N other nodes not selected to render, where relevancy (R) is bigger than the relevancy of the actual node. The pointers for other nodes with higher relevancy are ordered by the relevancy.

When an object finishes rendering, the graph is used to find other node where the cost (C) is smaller than the time available for the ray tracing phase. When the node with a suitable cost is found, all the other nodes with cost higher are removed from the graph and inserted in the other graph that is being built for the next frame.

When the first object finishes rendering and there are still time to trace other objects, the next object must be selected. In order to determine the best node, the selection graph is traveled until a node with cost (C) smaller than the cost still available for the ray trace phase is found. All the nodes with bigger costs are moved to the new selection node, in construction to be used for the next rendering frame. If a node is found, the graph is still traversed in case this node points to another one with higher relevance (R).

If there is no time to render a new object, all the nodes left are moved for the new graph and all the variables for calculated in order to create a new graph.

When the time left for the ray trace stage is not sufficient to render a new object, the new graph node is constructed for the next render phase.

5. Conclusion and Future Works

We have described a heuristic to select objects to be ray traced in a hybrid rendering pipeline, where the selected objects are the objects that most contribute to the visual experience of the user, based on the assumption that objects near the observed and near the center of the field of view are more relevant than others in different situations. We also offer a strategy to dynamically maintain a graph with the best candidates to be traced, based on some criteria.

As future works, new variations of the basic heuristic will be created and compared to the original in controlled tests that try to mimic common scenarios, like game levels, in order to verify the quality of the results towards the purpose of offering better graphics without negatively affect the experience.

References

BIKKER, J., [S.D.]. *Arauna Realtime Ray Tracing & Brigade Real-Time Path Tracing* [online]. http://igad.nhtv.nl/~bikker/, [accessed 23 Apr 2006].

CABELEIRA, J., 2010. Combining Rasterization and Ray Tracing Techniques to Approximate Global Illumination in Real-Time. [online]. *Direct*. http://www.voltaico.net.

CHEN, C.-C. AND LIU, D. S.-M., 2007. Use of hardware Z-buffered rasterization to accelerate ray tracing. In *Proceedings of the 2007 ACM symposium on Applied computing SAC 07.* . ACM.

DIETRICH, A, WALD, I., BENTHIN, C. AND AND P SLUSALLEK, 2003. The OpenRT Application PRogramming Interface - Towards A Common API for Interactive Ray Tracing. *Proceeding of the 2003 OpenSG Symposium*,

HERTEL, S. AND HORMANN, K., 2009. A hybrid GPU rendering pipeline for alias-free hard shadows. *Eurographics 2009 Areas Papers*, n. April, p. 59–66.

LAUTERBACH, C., 2009. Fast Hard and Soft Shadow Generation on Complex Models using Selective Ray Tracing. *Lloydia Cincinnati*, n. January.

LUDVIGSEN, H. AND ELSTER, A. C., 2010. Real-Time Ray Tracing Using Nvidia OptiX. *Science*, p. 1–4.

PARKER, S. G., BIGLER, J., DIETRICH, ANDREAS, ET AL., 2010. OptiX : A General Purpose Ray Tracing Engine. *ACM Transactions on Graphics TOG*, v. 29, n. 4, p. 1–13.

SABINO, T. L. R., 2012. *Uma Arquitetura de Pipeline Híbrida para Rasterização e Traçado de Raios em Tempo Real*. Master Thesis. Universidade Federal Fluminense.

SABINO, T. L. R., ANDRADE, P. M. F., CLUA, E. W. G. AND PAGLIOSA, P. A., 2011. Efficient Use of In-Game Ray-Tracing Techniques. *SBC - Proceedings of SBGAMES*,

SEILER, L., CARMEAN, D., SPRANGLE, E., ET AL., 2008. Larrabee: a many-core x86 architecture for visual computing. *ACM Trans Graph*, v. 27, n. 3, p. 1–15.

THIBIEROZ, N. AND ENGEL, W., 2003. Deferred Shading with Multiple Render Targets. *ShaderX2: Shader Programming Tips and Tricks with DirectX 9.0.*

WYMAN, C. AND NICHOLS, G., 2009. Adaptive caustic maps using deferred shading. *Eurographics 2009*, v. 28, n. 2.