A Heuristic Approach to Render Ray Tracing Effects in Real Time for First-Person Games

Paulo Marcos Figueiredo de Andrade
Esteban Walter Gonzalez Clua
Instituto de Computação
Universidade Federal Fluminense (UFF)
Niterói, Brazil
paulomfa@gmail.com, esteban@ic.uff.br

Fabio Siqueira D'Alessandri Forti
Design
Escola Superior de Propaganda e Marketing (ESPM)
Rio de Janeiro, Brazil
fabio_forti@yahoo.com.br

Thales Luiz Rodrigues Sabino
Tecgraf
Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro, Brazil
tluisrs@gmail.com

Abstract—**Realistic computer graphics effects as mirror reflections, transparency, caustics and detailed shadows are hard to simulate using real time raster rendering since they require global illumination approaches. One option is to ray trace these effects using real time hybrid rendering, where ray tracing and raster rendering are used together to generate the best visual experience. Unfortunately, the ray tracing stage of a hybrid renderer is very demanding, making it hard to maintain real time frame rates in virtual environments with many elements to be ray traced. This work presents a heuristic to select the best subset of elements to ray trace in a real time hybrid renderer, in order to improve the visual experience offered by an equivalent raster render, and still maintain a real time experience. The selection process considers rendering time constraints, spatial information of the environment, past elements selected for ray tracing and current information about the candidate elements.**

Keywords—Hybrid Rendering; Real Time Ray Tracing; Deferred Shading;

I. Introduction

Real time hybrid rendering is a recent research topic [1], [2], [3], [4], with many studies presenting visual improvements upon raster renderers. These visual improvements usually deals with raster rendering limitations, such as detailed shadows, transparency, reflections, among others. Unfortunately, in many studies, real time or interactive frame rates are only possible in very restricted conditions. These conditions are not acceptable for games, where a steady frame rate and the variety and sophistication of the virtual environment are crucial for the user experience.

In this paper, we present a heuristic that chooses, in real time, the best elements, in a virtual environment, to render using ray tracing. The selection is based upon four conditions: the processing power available to maintain acceptable real time frame rates, the current position and recent movement of the camera inside de virtual environment, the characteristics of each element and the previous selection of elements to be rendered using ray tracing.

In order to validate the heuristic, we test in indoor and outdoor virtual environments. For each virtual environment, the camera moves upon fixed paths, simulating the movement of first-person games. For each path, a group of elements is a candidate to be rendered using ray tracing. In order to test the heuristic, not all candidate elements should be ray traced, in order to maintain real time frame rates. Considering each path, in order to test the heuristic, some elements are required to be chosen. It is also required from the heuristic that elements do not change constantly form raster to ray trace rendering. In the work, an element can be part of an object or the full object, like a reflexive area of a mirror or an object made of glass.

The development of an efficient heuristic is challenging due to the structural diversity of game levels (virtual environments) in first person shooters and the exploratory liberty of the player inside the virtual environment. The two main obstacles to develop a good heuristic are visual consistency (one element should not change frequently from raster to ray tracing rendering), and contribution to the visual experience when the element is in the camera's field of view. The challenge can be defined in a simple question: when only one of two elements can be ray traced, which one best contribute to the visual experience.

This work employs a real time heuristic hybrid renderer called PHRT. PHRT is an improved version of the hybrid renderer presented in [4] and was developed to test heuristics for hybrid rendering. The first heuristic implemented was the heuristic proposed [5], but not tested. With PHRTs first version, early heuristic results were presented in [6]. In [6], the test environment was too generic and with polygon count too high for good real time results. In order to test the heuristic considering first person games, we created two virtual environments with low polygon count. For each environment, three camera paths inside these environments simulate the forward movement of a player in a first person shooter game.

This paper is organized as follows: section II briefly discuss related works; section III presents PHRT, the GPU based hybrid renderer employed in the tests. Section IV introduces the heuristic and section V details its mechanics. Section VI presents tests and results, and section VII details the conclusions and future works.

II. Related Works

This work is based on recent developments in real time hybrid rendering, selective rendering and heuristic based hybrid rendering.

A. Real Time Ray Tracing

The advance of desktop computers with multi-core processors and the massive parallel capabilities of current Graphic Processing Units (GPUs) encouraged the research of interactive and even real time ray tracing renderers (RTRT). One of the first relevant studies of RTRT is the OpenRT Project [7]. In 2004, the project succeeded in porting the game Quake 3 to OpenRT, achieving an average frame rate of 20 frames per second, in a cluster of 20 state of the art desktops, at that time. Another important project is Brigade [8], a path tracer for real time games, developed by Jakko Bikker.

With the advances in general computing using GPU (GPGPU), hybrid ray tracing renderers [1], [8], [9], [10], combining the use of CPUs and GPUs were proposed. Unfortunately, the demanding task of transferring data between CPU and GPU became a bottleneck to this approach.

The current development of RTRT fully at the GPU level [11], [12], is promising, however, experiments demonstrate that even using current parallel architectures, RTRT renderers cannot compete in speed and overall visual quality with state of the art real time raster renderers. Real time raster renderers employs strategies like cascaded light propagation volumes for real time indirect lighting [13] and rectilinear texture warping for fast adaptive shadow maps [14], to name a few, in order to avoid global illumination approaches, and still obtain visual quality.

B. Hybrid Rendering

Hybrid rendering proposes the use of different rendering strategies on order to offers an improved visual experience. Hybrid rendering is a common approach in offline renderers like Renderman [15] but just became popular as a strategy to improve the visual experience of real time raster renderers [2], [16].

C. Selective Rendering

Another research topic related to this work is selective rendering [17], [18], [19], [20]. Selective rendering considers psychophysical investigations on how the human visual perception works, in order to determine whether a detailed feature in the image is visible to the eye. Based on these observations, it is possible to avoid unnecessary computations involved in the generation of some characteristics of the final image.

Visual attention in real time applications [16], [21], [22], [23] is also an influence in this work. According to the experience inside the virtual environment, some details will not be perceived. Factors like speed of the movement inside the environment, familiarity with the environment (alien city, as opposed to a modern city), psychological experiences (encounter with powerful enemies) and motivations (search of an object), can affect the perception of details.

D. Heuristic Hybrid Rendering

Heuristic hybrid rendering proposes the use of heuristics to improve the overall visual experience by choosing the best render method for each task. One example is the approach proposed by Ammann [20] to render dynamic heightfields using a threshold based on terrain screen coverage to choose between mesh rendering and ray tracing.

III. PHRT - The GPU Based Hybrid Rendering

PHRT can render images using both raster and ray tracing methods. PHRT employs OpenGL for raster rendering and NVidia OptiX^{TM} [24], [25] for ray tracing. PHRT can render scenes using OpenGL, OptiX^{TM}, or a combination of both, when using the heuristic decisions.

PHRT reads a 3D mesh file containing information about the objects, materials, lights, textures, cameras and animation of the virtual environment and an XML file containing parameters for specific rendering tests. We use two files to allow different tests without changing the virtual environment file. The heuristic, detailed in section IV, uses information from both files and real time information generated during the virtual exploration of the environment. PHRT is capable of following predefined virtual paths inside the environment. PHRT collects information related to the rendering process during for every frame rendered. Information as average frame rate and elements selected for ray tracing is stored for later analysis and heuristic fine-tuning.

As most of the recent real time renderers, PHRT employs a technique called deferred shading [26], [27]. The basic idea of deferred shading is to compute all the geometry visibility tests before any light computation (shading) happens, using a raster process. By separating the geometry rendering from the light processing and by using visibility tests, the shading process is done only for specific polygons, avoiding multiple light computations in unnecessary fragments, typically coming from elements outside the visible space, a problem that must be treated in forward rendering approaches. The visible geometry determination process in the deferred shading pipeline is equivalent to the primary ray hit phase of a ray tracer, where eye rays projected from a virtual point of view are launched in the direction of the scene, crossing a view plane defined by a grid of pixels that represent the final image. In ray trace rendering, when a ray hit the surface of an object, information about the object, like its surface and geometry, is acquired. This information is used to define the final color of every pixel in the grid. Since the visibility test made by the GPU, in raster-based rendering, is very fast, and the result is the same information, the visibility test substitutes the first phase of the ray trace rendering.

During the first raster render pass, where the visibility tests are computed, scene Z-depth, surface normals and texture coordinates are also calculated. This set of related information about the image generation is called G-Buffer data and are stored in memory buffers denominated Multiple Render Targets (MRTs). The G-Buffer data are used in subsequent render passes of a raster renderer. The same happens in PHRT.

With the information corresponding to the primary ray intersection and the corresponding geometry of the scene, particular rays for shadows, direct and indirect light, refractions, and reflections are calculated and added to the already created raster data in the MRTs.

This deferred rendering method, in a hybrid renderer, allows the selection of which visual effects the ray tracing process should generate and which effects are the responsibility of the rasterization process.

The real time hybrid render pipeline has four main rendering stages: deferred rendering and primary ray resolution, shadows, reflections and refractions, and the final image composition. Figure 1 illustrates the rendering pipeline.

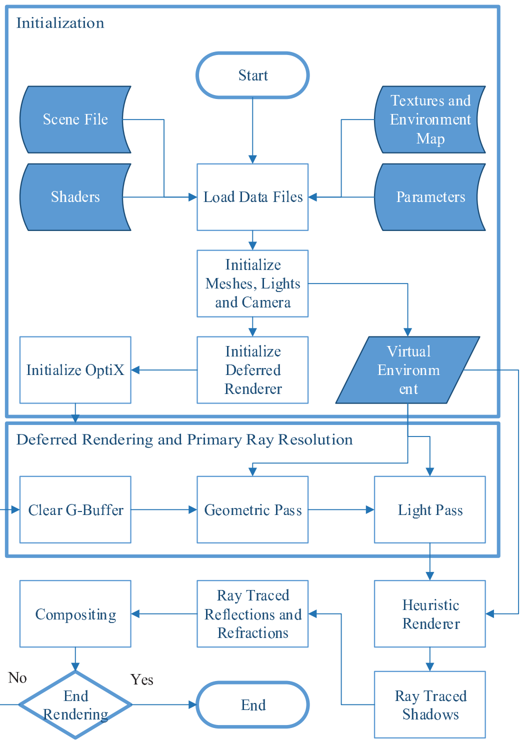

Fig. 1. Real Time Heuristic Hybrid Rendering Pipeline.

A. Deferred Rendering and Primary Ray Resolution

During 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 now has the information necessary for the visible geometry determination and the other render stages.

B. Ray Traced Shadows

With the information stored in the G-Buffer, OptiX^{TM} is used to calculate shadow rays for every light source of the scene. The ray tracing shadow stage can be specifically ignored for some elements, according to the heuristic, environment information and external parameters.

C. Ray Traced Reflections and Refractions

The reflections and refractions also use the information stored in the G-Buffer. Similar to the shadow stage, the heuristic decisions, environment information and external parameters are used to define which elements should have reflections and/or refractions in the final image.

D. Compositing

The compositing stage is the last stage of the pipeline, where all the information produced by the other steps are combined in order to produce the final image for the frame. Figure 2 shows the four stages of the hybrid renderer. Figure 2 (a) is the deferred shading result. Figure 2 (b) shows the ray traced shadows. Figure 2 (c) is the addition of reflections and refractions, and Figure 2 (d) is the final compositing.

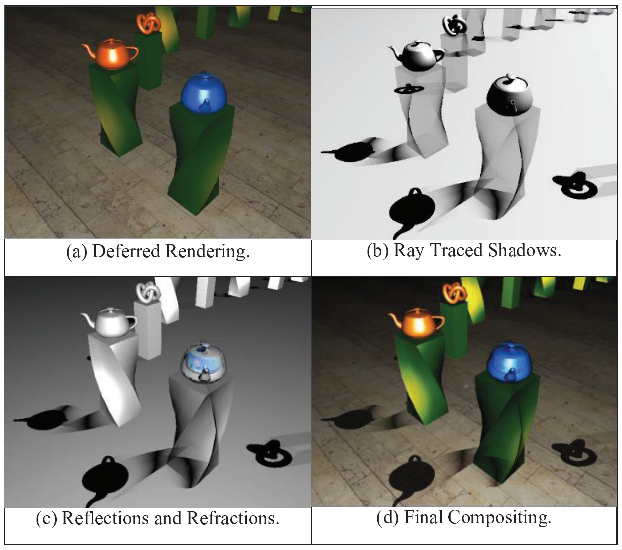

Fig. 2. (a) Deferred Rendering Result. (b) Ray Traced Shadows. (c) Addition of Reflections and Refractions. (d) Final Compositing.

IV. Heuristic Definition

Considering only the elements that should be involved in the ray tracing phase, the heuristic can be defined as “for a given set of X elements, select the Y most relevant elements that can be ray traced given a time limit T”. X is the set of elements that are relevant for the image generation at a given time and T is the time available for the ray tracing phase. Both parameters X and T can change for every frame. Depending on the camera position in the virtual environment, some elements will not be considered for rendering since these elements are out of the cameras field of view. In addition, some elements cannot be rendered due to its cost.

The heuristic definition presents the following challenges:

A. Time Constraints

Since one of the main requisites of the hybrid renderer is a steady frame rate, one of the main constraints for the ray tracing phase is the time available after the raster phase. Even to produce the simplest light effect, ray tracing is a demanding task and can take a long time to finish. One of the reasons of its high cost is the recursive nature of the ray tracing algorithm. In order to produce global illumination effects, light rays must bounce from surface to surface, in order to produce indirect illumination and other light effects. The number of light rays bounces and the sequential nature of the bounces can strongly affect the rendering pipeline and the global performance. 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 created in every bounce. Depending on the surface characteristics, a ray collision can produce more than one new ray, greatly affecting the performance.

The heuristic must choose wisely which elements must be ray traced in a given time, in order to maintain the time constraints of the ray tracing stage. The time spent in the ray tracing stage is affected by the type of light effects, characteristics of the surfaces, number of lights in the virtual environment and relative size of the visible portion of the elements to be ray traced, among others. The render of all effects can surpass the time available for the stage.

B. *Element Selection*

The reason behind the idea of choosing a subset of elements *Y* that most contribute to the visual experience in a first person game came from the observation on how the vision works when we are in movement. 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 details in the virtual environment.

Considering the way the human vision works, it is reasonable to assume that objects near the center of the field of view call more attention than objects far from it. The same can be said for objects near the observer when compared to objects afar. Another observation is that according to environment features and conditions (weather, indoor, underwater, for example), some objects cannot be visually improved by ray tracing effects.

Psychological and motivation aspects can also affect visual perception. For example, if the user is looking for a gold coin in the environment, the user will be more susceptible to pay attention to golden objects.

V. Heuristic Mechanics

The heuristic employs a directed graph as the support data structure to choose the elements for the ray tracing phase. The heuristic has four phases, two phases that happen before the rendering process, called off-line phases, one phase that happens during the rendering process and another phase that happens when all the elements have already been selected for rendering.

A. *Pre-production Phase*

This is an off-line phase, where an environment or level designer identifies the elements in the virtual environment that are the candidates for ray tracing. The same way an environment designer chooses shaders for a given element, in the pre-production phase, the environment designer chooses which elements must be ray traced and what ray tracing effects should be used. For the tests, the ray tracing effects are shadows, reflections, refractions and transparency. The designer also defines a priority index *K* for each candidate. Priority index *K* is used to identify how important, when compared to the other elements, this element is, according to the designer point of view. Priority index *K* is a designer tool to guarantee that an element will always be chosen for ray tracing, or is more relevant than others.

B. *Selection Graph Construction Phase*

This second phase, also off-line, is used to construct the weighted selection graph employed by the heuristic to select the best elements for ray tracing. Every node in the selection graph correspond to an element. In addition, every node has an estimated processing cost *C* and a relevance *R*. Cost *C* is estimated by taking into account the element visibility *V* and its ray tracing cost *Q*. Equation (1) presents visibility *V*. *A* is the area of the element in the view plane of the camera. *P* is the normalized distance between the center of the element and the center of the view plane, where the value of 1 means that the element is in the view plane centre, and a value of 0 means that the element is outside the view plane. *D* is the distance between the element and the view plane in the virtual environment. The higher is the distance from the view plane the smaller is visibility *V*, making big and distant elements less "visible" than near elements with the same area *A* in the view plane. Equation (2) presents the element cost *C*.

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

$$C = V * Q \qquad (2)$$

While cost *C* is a way to measure how much work is necessary to render an element, relevance *R* represents the element contribution to the visual experience. In Equation (3), selection *S* is a binary value where the value 1 means that the object was previously selected for ray tracing and the value 0 means it was not. Selection weight *I* is a constant that defines how important is choosing a previously selected element for ray tracing.

$$R = (S * I + V) * K \qquad (3)$$

The selection graph has two kinds of edges. The first kind is a linked list where one node connects to the other node which cost *C* is smaller than its own cost, but bigger than the cost *C* of all the other nodes. the time spent in the ray tracing stage the time spent in the ray tracing stage The second kind connects each node to all the other nodes where relevance *R* is bigger than its own relevance.

Table I presents all the parameters mentioned before and inform if the parameter can change its values for every frame or has a fixed value.

TABLE I. Equation Parameters

Param.	Definition	Constant	Variable
K	Element relevance among the others	X	
V	Element visibility		X
Q	Ray tracing cost	X	
C	Processing cost		X
A	Element area in the view plane		X
P	Distance from the center of the view plane		X
D	Distance from the view plane (camera)		X
R	Relevancy		X
S	Previously selected or not		X
I	Weight of past selection	X	

At the end of this phase, the heuristic has a directed graph ready for use during the rendering phase.

C. Node Selection Phase

The third phase happens during the rendering phase, when the GPU receives the selection graph. The heuristic chooses the N most relevant *R* nodes whose cost *C* allows the nodes to be rendered and still maintain the expected frame rate. N is the number of the stream processors of the GPU. If there is still time to render other elements, the heuristic chooses the most relevant nodes that have cost *C* small enough to be rendered in the available time.

D. Graph Rebuild Phase

When there is no more time available to choose another node, the selection graph is reconstructed and updated for the next rendering phase. New cost *C* and relevance *R* are calculated, and the node edges are rebuilt. When the new graph is created, the renderer clears the G-Buffer and renders the new frame.

VI. Tests and Results

For the indoor tests, we use the sponza virtual environment with some primitives created in Autodesk 3ds Max 2013. Just the primitives are candidates for ray tracing. Reflection and transparency are the focus of the sponza tests. For the outdoor tests, we created the atrium virtual environment with trees and curved surfaces of different sizes. In atrium, the focus is detailed shadows. The candidate elements to ray trace in sponza are the primitives. In atrium, the candidate elements are the trees, the statues and the water in the pool.

Three camera paths were created for each environment. These camera paths simulate the movement of a first person game. The camera animation resulted from following the paths helped verify if the heuristic chooses the right elements to render during the movement inside the virtual environment.

Figure 3 and Figure 4 are, respectively, perspective views of sponza and atrium virtual environments. In Figure 3, only the primitives are ray traced. In Figure 4, the entire environment is rendered using ray tracing.

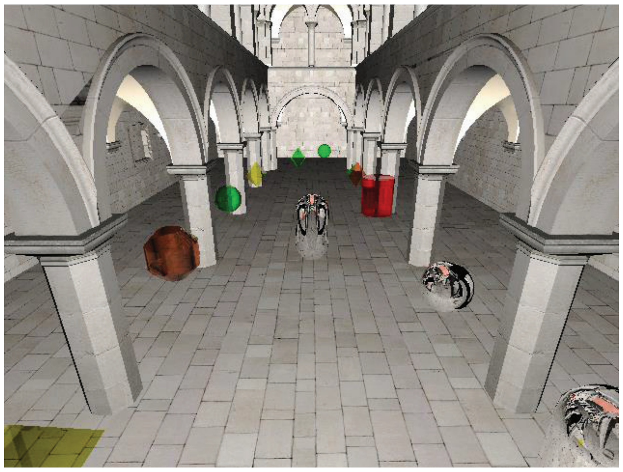
Fig. 3. Perspective View of the Sponza Virtual Environment.

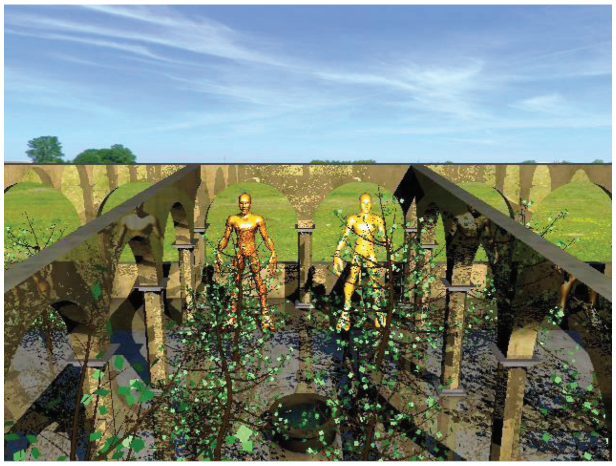
Fig. 4. Perspective View of the Atrium Virtual Environment.

For the tests, three camera paths for each environment were rendered in OpenGL (raster), ray tracing and using the heuristic hybrid renderer. We measure the average time to render each frame in milliseconds. Each camera animation runs for 1000 frames.

All the tests were performed in an Asus GJ73JW notebook with 16 GBytes of RAM, an Intel Core I7 Q 740 1.73GHz CPU and an NVIDIA GeForce GTX460M GPU running Windows 8. The screen resolution for the tests is 640x480 pixels.

To validate the heuristic, the tests have three pre-requisites:

- The average frame rendered by the hybrid renderer must be around 150 milliseconds.
- The most relevant elements must be rendered in ray tracing.
- Elements rendered in ray tracing can only be rendered again in raster if it does not affect the visual continuity.

Figure 5, Figure 6 and Figure 7 are from the sponza tests and represent the raster rendering, the ray trace rendering and the hybrid rendering, respectively.

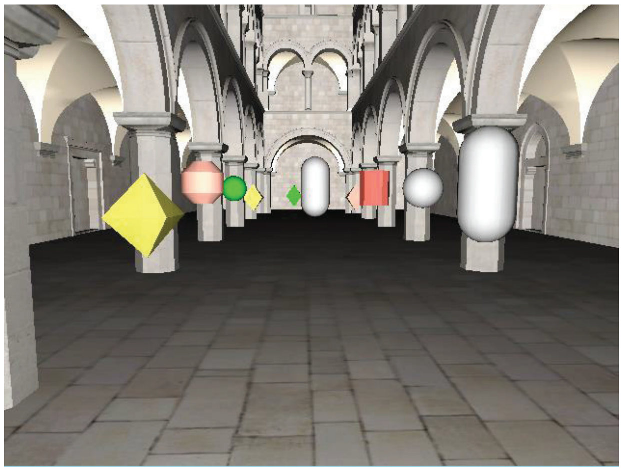
Fig. 5. Raster Rendering. Sponza.

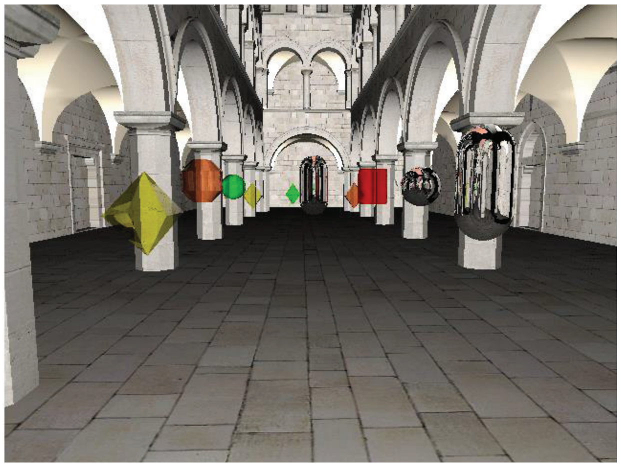

Fig. 6. Ray Trace Rendering. Sponza.

In the sponza tests, no ray traced shadows are visible because the surface to receive the shadows must participate in the ray tracing phase of the hybrid renderer, demanding that all the scene floor became a ray tracing surface. This was a limitation of PHRT early versions. PHRT Current version does not have this limitation, but we decided to maintain the sponza virtual environment in the tests to verify PHRTs performance improvements.

In Figure 7, the heuristic decided not to ray trace the green sphere and the last three octahedrons.

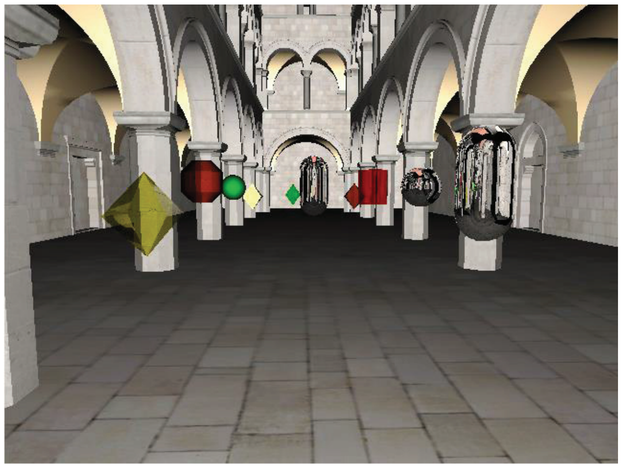

Fig. 7. Heuristic Hybrid Rendering. Sponza.

Figure 8 and Figure 9 are, respectively, raster and hybrid renderings of the atrium tests.

Table II summarizes the results for the three animation tests. The first column identifies the path used in the animation test. The second column is the average time to render a frame, for that path. The third column is the average time to render using the heuristic, and the fourth column is the average time to render all the candidate elements using in ray tracing. All the values are in milliseconds.

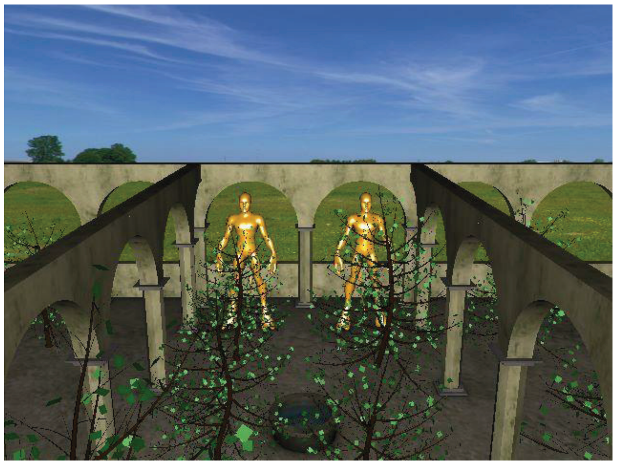

Fig. 8. Ray Trace Rendering. Atrium.

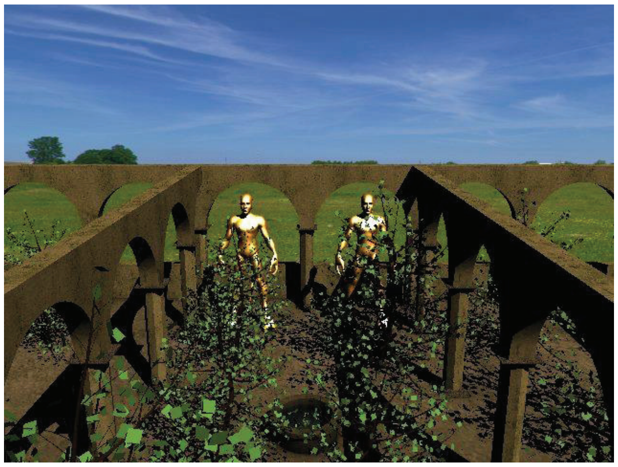

Fig. 9. Heuristic Hybrid Rendering. Atrium.

TABLE II. RENDER TIMES FOR THE ANIMATION TESTS (IN MILLISECONDS)

Sponza Virtual Environment			
Path	**Raster Rendering**	**Hybrid Rendering**	**Ray Tracing Rend.**
1	15,2	102,2	361,7
2	22,1	117,3	398,5
3	25,2	139,7	594,1
Atrium Virtual Environment			
Path	**Raster Rendering**	**Hybrid Rendering**	**Ray Tracing Rend.**
1	26,7	165,4	340,4
2	21,8	125,6	543,2
3	30,6	207,9	845,3

In all the tests, the heuristic did good choices in the selection of the elements to ray trace. During the tests, no element repeatedly changed from raster to ray tracing and vice-versa.

We also did some fine-tuning in the heuristic in order to deal with huge ray tracing elements. When the area to ray trace corresponds to more than 40% of the frame size, the heuristic ignores the ray tracing phase.

Figure 10 represents a scene where more than 60% of the image was rendered using ray tracing. This image, with huge transparent and reflexive areas, took more than 10 times the average time to render than the heuristic hybrid render tests. This image was created using the hybrid renderer, with the heuristic selection turned off. Without the heuristic, the floor and some walls were selected for ray tracing, making the scene impossible to render in real time.

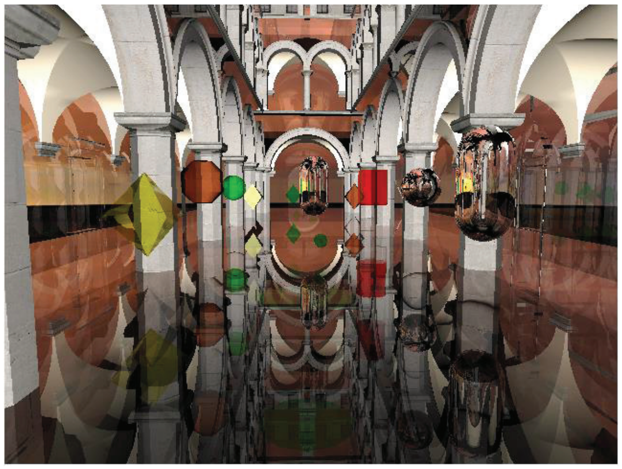

Fig. 10. Hybrid Rendering. 60% of the Scene is Ray Traced.

VII. Conclusions and Future Works

We have described a heuristic to select elements to be ray traced in a hybrid rendering pipeline. The selected elements are the ones that better contribute to the visual experience, considering constraints and the assumption that objects near the observed, and near the center of the field of view are more relevant, when playing a first person shooter. We also present a strategy to create and maintain, in real time, a graph with the best candidates to be ray traced. All the tests and scenarios were built to run at a minimum of 20 frames per second in the target computer.

The tests indicate that the ray tracing phase is still too demanding to deal with very complex environments with many elements to ray trace, but we believe that the ray tracing phase will become less and less demanding as the GPU architecture evolves. We informally tested the renderer in newest machines to guarantee that the hybrid renderer could run for more than 20 frames per second in the selected virtual environments. It is worth mention that we noticed impressive speed gains when using recent NVidia's GPUs.

We are still testing the heuristic and developing more virtual environments that resemble indoor and outdoor game environments. We plan to run stress tests with these new virtual environments. In order to measure both the performance and degradation level of the selection graph reconstruction, the heuristic hybrid renderer will run for very long and random paths, in the new environments. New variations of the basic heuristic are also in development and will be compared to the original heuristic.

Considering the hybrid heuristic, we plan to implement an option to generate low quality versions of reflections as another option to the hybrid renderer, instead of discarding the render of reflections that are too demanding to render given the time constraints.

Finally, we plan to test PHRTs new version in a state of the art hardware in order to verify if the gap between the raster and ray tracing renderer is decreases or not, when compared to order GPUs.

References

[1] S. Beck, A. Bernstein, D. Danch, and B. Fröhlich, "CPU-GPU Hybrid Real Time Ray Tracing Framework," *Eurographics 2005*, vol. 0, no. 0, pp. 1–8, 2005.

[2] S. Hertel, K. Hormann, and R. Westermann, "A hybrid GPU rendering pipeline for alias-free hard shadows," *Eurographics 2009 Areas Pap.*, no. April, pp. 59–66, 2009.

[3] J. Cabeleira, "Combining Rasterization and Ray Tracing Techniques to Approximate Global Illumination in Real-Time," *Direct*. 2010.

[4] T. Sabino, P. Andrade, E. Gonzales Clua, A. Montenegro, and P. Pagliosa, "A Hybrid GPU Rasterized and Ray Traced Rendering Pipeline for Real Time Rendering of Per Pixel Effects," in in *Entertainment Computing - ICEC 2012 SE - 25*, vol. 7522, M. Herrlich, R. Malaka, and M. Masuch, Eds. Springer Berlin Heidelberg, 2012, pp. 292–305.

[5] P. M. F. Andrade, T. L. R. Sabino, E. W. G. Clua, and P. A. Pagliosa, "A Heuristic to Selectively Ray Trace Light Effects in Real Time," in *XI Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGames 2012)*, 2012, pp. 1–4.

[6] P. Andrade, T. Sabino, and E. Clua, "Towards a Heuristic based Real Time Hybrid Rendering - A Strategy to Improve Real Time Rendering Quality using Heuristics and Ray Tracing," in *Proceedings of the 9th International Conference on Computer Vision Theory and Applications*, 2014, pp. 12–21.

[7] A. Dietrich, I. Wald, and P. Slusallek, "The OpenRT Application Programming Interface - Towards A Common API for Interactive Ray Tracing," *Proceeding 2003 OpenSG Symp.*, pp. 23–31, 2003.

[8] J. Bikker and J. van Schijndel, "The Brigade Renderer: A Path Tracer for Real-Time Games," *Int. J. Comput. Games Technol.*, vol. 2013, pp. 1–14, 2013.

[9] C.-C. Chen and D. S.-M. Liu, "Use of hardware Z-buffered rasterization to accelerate ray tracing," in *Proceedings of the 2007 ACM symposium on Applied computing SAC 07*, 2007, pp. 1046–1050.

[10] A. Pajot, L. Barthe, M. Paulin, and P. Poulin, "Combinatorial Bidirectional Path-Tracing for Efficient Hybrid CPU/GPU Rendering," *Comput. Graph. Forum*, vol. 30, no. 2, pp. 315–324, 2011.

[11] A. García, F. Ávila, S. Murguía, and L. Reyes, "Interactive Ray Tracing Using the Compute Shader in DirectX11," in in *GPU Pro 3*, W. Engel, Ed. A K Peters/CRC Press, 2012, pp. 353–376.

[12] S. Parker, "Interactive ray tracing with the NVIDIA®OptiX^{TM} engine." SIGGRAPH, 2009.

[13] A. Kaplanyan and C. Dachsbacher, "Cascaded light propagation volumes for real-time indirect illumination," in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D 10*, 2010, vol. 1, no. 212, p. 99.

[14] P. Rosen, "Rectilinear texture warping for fast adaptive shadow mapping," in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D '12*, 2012, vol. 512, no. row 1, p. 151.

[15] P. Christensen, "Ray Tracing for the Movie 'Car'," in *2006 IEEE Symposium on Interactive Ray Tracing*, 2006, vol. 138, no. 1, pp. ix–ix.

[16] C. Lauterbach, "Fast Hard and Soft Shadow Generation on Complex Models using Selective Ray Tracing," *Lloydia Cincinnati*, no. January, 2009.

[17] K. Cater, A. Chalmers, and G. Ward, "Detail to attention: exploiting visual tasks for selective rendering," in *EGRW 03 Proceedings of the 14th Eurographics Workshop on Rendering Techniques*, 2003, pp. 270–280.

[18] A. Chalmers, K. Debattista, G. Mastoropoulou, and L. Paulo, "There-Reality : Selective Rendering in High Fidelity Virtual Environments," *Int. J.*, vol. 6, no. 1, pp. 1–10, 2007.

[19] K. Cater, A. Chalmers, and P. Ledda, "Selective quality rendering by exploiting human inattentional blindness: looking but not seeing," in *Human Factors*, 2002, pp. 17–24.

[20] C. S. Green and D. Bavelier, "Action video game modifies visual selective attention.," Nature Publishing Group, 2003.

[21] M. S. El-Nasr and S. Yan, "Visual attention in 3D video games," *Proc. 2006 Symp. Eye Track. Res. Appl. ETRA 06*, vol. 31, no. 1980, p. 42, 2006.

[22] V. Sundstedt, K. Debattista, P. Longhurst, A. Chalmers, and T. Troscianko, "Visual attention for efficient high-fidelity graphics," *Proc. 21st spring Conf. Comput. Graph. SCCG 05*, p. 169, 2005.

[23] K. F. Cater, "Detail to Attention : Exploiting Limits of the Human Visual System for Selective Rendering," University of Bristol, 2004.

[24] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. Mcallister, and M. Stich, "OptiX : A General Purpose Ray Tracing Engine," *ACM Trans. Graph. TOG*, vol. 29, no. 4, pp. 1–13, 2010.

[25] H. Ludvigsen and A. C. Elster, "Real-Time Ray Tracing Using Nvidia OptiX," *Science (80-.).*, pp. 1–4, 2010.

[26] M. Deering, S. Winner, B. Schediwy, C. Duffy, and N. Hunt, "The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics," *ACM SIGGRAPH Proc.*, vol. 22, no. 4, pp. 21–30, 1988.

[27] T. Saito and T. Takahashi, "Comprehensible rendering of 3-D shapes," *ACM SIGGRAPH Comput. Graph.*, vol. 24, no. 4, pp. 197–206, 1990.