Ray-Traced Reflections in Real-Time Using Heuristic Based Hybrid Rendering

Paulo Andrade
UFF
IC, Medialab

Thales Sabino
Tecgraf/PUC-RJ

Esteban Clua
UFF
IC, Medialab

Paulo Pagliosa
UFMS
FACOM

Figure 1: Screenshot of the farm virtual environment, rendered with ray traced shadows and reflections.

Abstract

Game engines typically use cube maps or screen space local reflections for simulating reflections effects at the real-time renderer pipeline. While these techniques work in many situations, they cannot deal well with reflections of reflections, reflections from covered spaces, reflections from elements outside the screen space or reflections from elements that change its shape or position in real-time. In this paper, we present a method capable of creating true real-time ray-traced reflections by using a heuristic based method and a hybrid renderer. Our solution maintain real-time frame rates by selecting, for each frame, the best reflexive surfaces to ray trace, based on the user's point of view.

Keywords: ray-tracing, real-time rendering, hybrid rendering, specular reflection

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

1. Introduction

Reflections are a typical component of real environments, making it an important factor in the creation of realistic digital images. Reflections can be specular, producing a mirror like effect in the surface, and can be diffuse, generating blurred reflections. The surface can also have both characteristics, with different intensities for each reflection.

Current game engines implement one or more techniques to produce reflections. No current technique can offer true real reflections for all the possible situations. Some techniques cannot render reflection of elements outside the screen space or elements occluded by others, when considering the camera field of view. Other methods cannot deal with moving elements. Most of these limitations are due to the deferred rendering and lighting process, commonly used by these real-time rendering engines, where only fragments of the elements inside the screen space are shaded.

This work presents a strategy to offer true ray-traced specular reflections, in real-time, by using a heuristic that can select, also in real-time, the best elements to receive reflections, considering the first-person view of the virtual environment. These elements are inside the camera's field of view and are the elements that best contribute to the user's visual experience. This selective rendering approach renders only the most relevant elements, within specific time constraints. This time constraints are used to maintain real-time frame rates. Tests show that, when focusing on the most relevant elements in the virtual environment, it is possible to offer superior visual experience, while maintaining real-time frame rates.

Our heuristic based hybrid rendering strategy can overcome some rendering limitations imposed by methods used in current game engines. Some of these methods are discussed in Section 2. The heuristic approach can also accommodate other ray-traced effects, like refractions and shadows.

2. Related Work

In order to generate reflections, most game engines support an old method called cube map [Greene 1986], a variant of the environment mapping method, proposed by Blinn and Newell [1976]. This planar reflection method maps all possible reflections onto six cube faces. A texture covering these faces represents the surrounding environment. This texture must be updated to reflect changes in the environment. Cube maps have three shortcomings. First, it does not deal with motion

parallax [Yu et al. 2005]. Second, constant updating the cube map affects the overall performance. Third, the reflector element is omitted from the environment map, making reflections from reflections impossible.

First mentioned in [Sousa et al. 2011], Screen Space Reflections, also called Real-Time Local Reflections, allow curved surfaces to reflect nearby surroundings in real-time. This method also allows self-reflections but cannot reflect occluded surfaces or surfaces outside the camera's field of view.

Reflection Capture Actors, also called Environment Probes [Sinha et al. 2012], are invisible objects that are placed throughout the environment, to capture virtual environment visual information. These objects, usually a cube or sphere, capture visual information from the virtual environment and feed reflection data into the Reflection Environment, which allows image-based lighting and reflections. Multiple probes are used to reduce parallax problems related to image-based methods. Strategically placed probes can feed reflection data from near elements. Near probes have priority against distant probes, reducing the parallax effect. Environment Probes can produce reflections from static elements outside the screen space. Environment Probes main limitation is that the probe placement is static and happens during the level design process. Environment Probes are inspired by Paul Debevec work with physical light probes [Debevec 2001, 1998] to capture HDRI data.

In order to design the heuristic, observations from the selective rendering [Cater et al. 2003, 2002; Chalmers et al. 2007; Debattista et al. 2005; Hasic and Chalmers 2009; Lever et al. 2005; Mania et al. 2008], and visual attention [Cater 2004; Desimone and Duncan 1995; Green and Bavelier 2003; Parker et al. 1999; Sundstedt et al. 2005; Yan and Seif El-Nasr 2006] fields of study were considered. Both field of studies supports the idea that it is not necessary to render every detail of moving images, in order to offer a rich visual experience.

Also related to this work are the general principles of good level design for first-person games [Brown and Chen 2001; Dormans 2010; Feil and Scattergood 2005; Hullett 2010]. Good level design is important not only for entertainment, but for overall game performance. A poorly designed level can cause game slowdowns that degrade user's experience.

3. PHRT Renderer

PHRT is the name of the heuristic based, real-time hybrid rendering engine used in this work. PHRT is an improved version of the engine presented in [Sabino et al. 2012; Sabino 2012] and is capable of ray-trace specific elements in the virtual environment, according to a heuristic input. The decision if which elements should be rendered using ray tracing happens for every frame. An early version of PHRT was used in [Andrade et al. 2014a, 2014b]. PHRT's current version can better deal with reflections and have many performance improvements. In addition, both the proposed heuristic and PHRT's current version were specifically designed to deal with recursive reflections, where one reflexive surface reflects another reflexive surface.

PHRT use deferred shading [Deering et al. 1988; Pritchard et al. 2004; Saito and Takahashi 1990; Whitted and Weimer 1982] for rasterization and OptiX [Ludvigsen and Elster 2010; Parker 2009; Parker et al. 2010] for ray tracing effects. PHRT has the following features:

- Render in real-time reflections, refractions, ray-traced and shadow mapping shadows;
- Render the virtual environment using only OpenGL, only OptiX ray tracing or a heuristic based hybrid mode;
- Render pre-defined camera animation, in order to test the heuristic and collect data for statistics.

In order to have a steady real-time frame rate, the ray-traced reflections phase have a time limit. The heuristic takes into account this time constraint to decide which elements can be ray-traced.

Figure 2 is an overview of PHRT rendering pipeline. The pipeline is described in Sections 3.2 to 3.6.

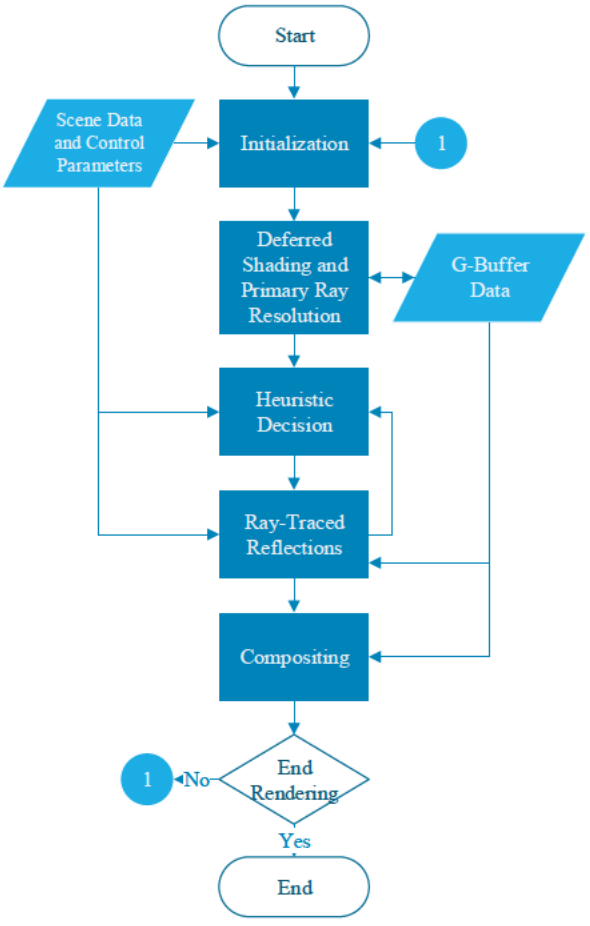

Figure 2: PHRT rendering pipeline.

3.1 Pre-Production

The Pre-Production phase is an off-line phase and happens before PHRT's execution. In this phase, the environment or level designer assign a priority index *K* for each reflexive surface. This information is stored in an XML file that is read in the initialization phase. *K* value identifies how important one element is when compared to the other elements in the virtual environment. An element with a high *K* value is considered more relevant than an element with a lower *K* value. With *K*, the environment designer helps the heuristic to do good choices, by informing which elements are more important to render during the ray-traced reflections phase.

3.2 Initialization

The Initialization phase starts before the proper rendering stage. PHRT loads the 3d meshes, textures, rendering parameters and heuristic parameters. The virtual environment is built in the computer's memory, and both the deferred and OptiX rendering processes are initialized.

The Selection Graph is built during the Initialization phase. The Heuristic Decision phase use the Selection Graph to select which elements in the virtual environment must be rendered using ray tracing. Section 4 present the Selection Graph construction and update procedures

3.3 Deferred Shading and Primary Ray Resolution

The Deferred Shading and Primary Ray Resolution phase use the virtual environment data to fill the G-buffer. Visibility tests are done using the G-buffer data, and basic light computation is done using OpenGL. OptiX uses the Z-Buffer data stored in the G-buffer as the equivalent of the primary ray hit phase of the ray tracing algorithm. In PHRT, OptiX is used to trace secondary rays for shadows, reflections and refractions.

3.4 Heuristic Decision

In general terms, the heuristic must select the best candidates for the Ray-Traced Reflections phase. The heuristic goal 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 constraint *T*." *Y* are the more relevant elements at that moment, according to the heuristic. *T* is the time available for the ray tracing phase. *X*, *Y* and *T* can change for every frame, according to the camera position in the virtual environment. *X* does not represent all the reflexive elements in the virtual environment, just the elements involved in the in the current image generation.

3.5 Ray-Traced Reflections

With the virtual environment information stored in the G-buffer, secondary rays are traced from each selected surface into the virtual environment. If a secondary ray hit one element in the virtual environment, the element color and light information is retrieved for the compositing phase. The environment map information is used if there is no element in the ray's path. If the secondary ray hit a reflexive element, this element is included in the Selection Graph if it is not already in. This procedure allows reflections from reflexive surfaces outside the camera's field of view.

After OptiX computes all the reflections for the selected elements, PHRT verifies if there is still time available in the Ray-Traced Reflections phase. If there are, the Selection Graph is updated, and the Heuristic Decision phase is executed again, while there is time for rendering.

3.6 Compositing

Compositing is the final phase, when both the raster and ray traced information is combined to generate the final image. Figure 3 shows PHRT's full rendering pipeline, also showing ray traced shadows and refractions.

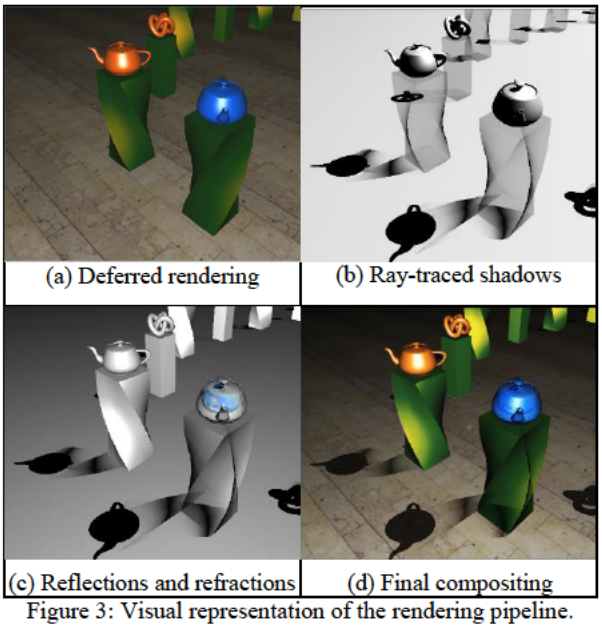

Figure 3: Visual representation of the rendering pipeline.

4. Heuristic Mechanics

The heuristic core is the selection of the best candidate elements to ray trace, giving a specific time constraint *T*. The heuristic was designed considering first-person point of view.

The three basic observations considered, in order to design the heuristic are:

1. During the virtual environment exploration, first-person game players and virtual reality users tend to center the focus of the camera at the most relevant elements in the virtual environment;

2. First-person game players and virtual reality users pay more attention to elements near and in front of the virtual camera;
3. Elements that frequently change its appearance attract more attention than elements that maintain its overall appearance.

These considerations were used to design the visibility metric *V*, presented in Section 4.1. Visibility is computed for each candidate element to ray trace and is used to calculate the processing cost *C* and relevance *R* of each candidate element. *C* and *R* are stored in the Selection Graph.

With *C* and *R* values for each candidate element, PHRT constructs and maintain a directed graph as the Selection Graph for the Heuristic Decision phase. The Selection Graph is constructed during the Initialization phase and is updated during the Heuristic Decision phase. Later, when there is no more time available for rendering, the Selection Graph is rebuild.

This heuristic have some similarities with the heuristics presented in [Andrade et al. 2014b], but was adapted to deal specifically with reflections. One of the key differences is that this heuristic can deal dynamically with reflections from reflections.

4.1 Selection Graph Construction

The Selection Graph is first constructed during the Initialization phase and considers all the candidate (reflexive) elements in the virtual environment. Every node in the Selection Graph corresponds to an element. In addition, every node has an estimated processing cost *C*, and a relevance *R*. *C* is estimated by taking into account the element visibility *V* and its ray tracing cost *Q*. The ray tracing cost *Q* initial value is 1 for every element. *Q* value is increased by 1 when a secondary ray from its surface hit another reflexive surface in the virtual environment, and so on. *Q* is the equivalent of the number of ray bounces between reflexive surfaces, and it is limited to 5, representing 5 bounces after the primary ray hit in the reflexive surface. This limitation is only due to performance constraints of the target hardware used in the tests. When two reflexive surfaces are involved, *C* is the sum of the costs of each surface.

Equation (1) present visibility *V*. In Equation (1), *A* is the visible area of the element projection in the view plane of the camera. *P* is the normalized distance between the geometric center of the element bounding box and the center of the view plane. *P* equals 1 means that the element is in the view plane center, and *P* equals 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 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) present the element cost *C*. *O* is only used when the selected element also reflect a reflexive surface. *O* equals 1 when the distance between the two reflexive surfaces is equal or less than *D*. *O* increases in 1 every time the distance between the two reflexive surfaces doubles, when compared to *D*.

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

$$C = \sum (V * Q/O) \quad (2)$$

While cost *C* is a strategy to measure how much work is necessary to render a particular element, relevance *R* represents the element contribution to the visual experience. In (3), selection *S* is a binary value where 1 means that the element 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 select a previously selected element.

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

The Selection Graph has two types of edges. The first type is a linked list, where one node connects to the other node where the cost *C* is smaller than its own cost but bigger than the cost *C* of all the other nodes. The second type, also a linked list, connects each node to all the other nodes where relevance *R* is bigger than its own relevance but with cost *C* equal or less its own cost. The edges related to cost *C* identify the elements that can be rendered, given the time constraint *T*, and the edges related to relevance *R* are used to identify the element that best contribute to the visual experience.

Figure 4 represents a Selection Graph with the edges related to relevance *R*. The cost *C* for each node is in the green (left) part of the circles. The relevance *R* is in the blue (right) part of the circles. The arrows point to the nodes with cost smaller or equal than the actual node and relevance equal or bigger than the actual node.

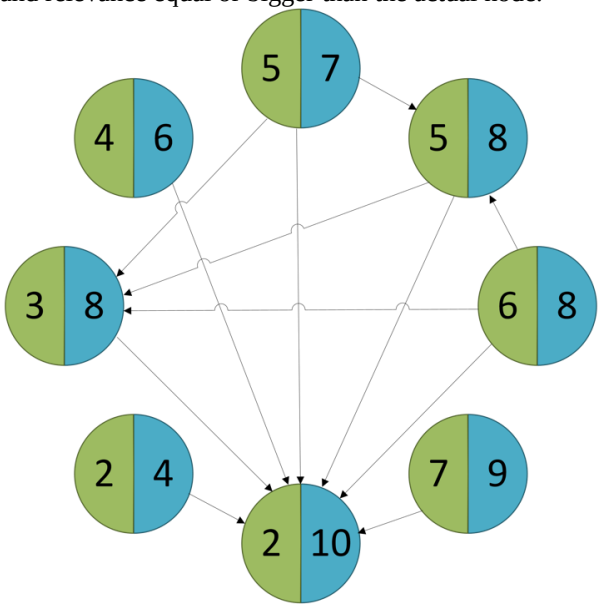

Figure 4: Visual representation of a Selection Graph.

Table 1 summarizes all the parameters for Equation (1), (2) and (3). For each parameter, Const. and Var. indicate if the parameter value is constant or variable during PHRT execution.

Table 1: Heuristic parameters.

Param.	Definition	Const.	Var.
K	Element relevance among the others	X	
V	Element visibility		X
Q	Ray tracing cost		X
O	Factor for the relative distances between 2 reflexive surfaces		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	Element was previously selected or not		X
I	Weight of element being previously selected	X	

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

4.2 Node Selection

Node Selection happens during the Heuristic Decision phase. The heuristic selects the *N* most relevant *R* nodes whose cost *C* allows the nodes to be rendered and still maintains the expected frame rate. *N* is the number of the stream processors of the target GPU. If there is still time to render other elements, the heuristic choose the most relevant nodes that have cost *C* small enough to be rendered in the available time *T*.

4.3 Selection Graph Reconstruction

When there is no more time available to select 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.

5 Tests and Results

PHRT, the real time hybrid renderer developed for this work is not a true game engine so, many common optimizations present in the game engines are not implemented, which limit our tests to small environments. In addition, at the moment of the tests, OptiX was not optimized for games, making the ray tracing part of the pipeline very demanding. However, these limitations do not impair the validation of heuristic use, on the contrary, with an increased rendering performance, the heuristic can offer even better results.

To test our heuristic, we created two virtual environments, one outdoor and one indoor. The outdoor environment is a cartoon like farm, where ponds, troughs and barrels have reflexive surfaces. The indoor environment is an underground sewer, where some elements, the water surface and some walls are reflexive. The water surface of the indoor environment is divided in segments, since the water surface covers huge areas.

For both environments, a fixed forward only camera path was created. These paths are used test how the heuristic performs considering element placement and surface sizes. In both paths, the camera "see" elements with different sizes and areas with big and small number of reflexive surfaces. Both camera paths runs for 10.000 frames in the animation and modeling software. For the tests, both paths were traversed 10 times. During the traversal, we measure the time to render each frame and which elements were selected for ray tracing, during the frame rendering.

The indoor environment was designed to offer many opportunities for recursive reflections, as shown in Figure 5.

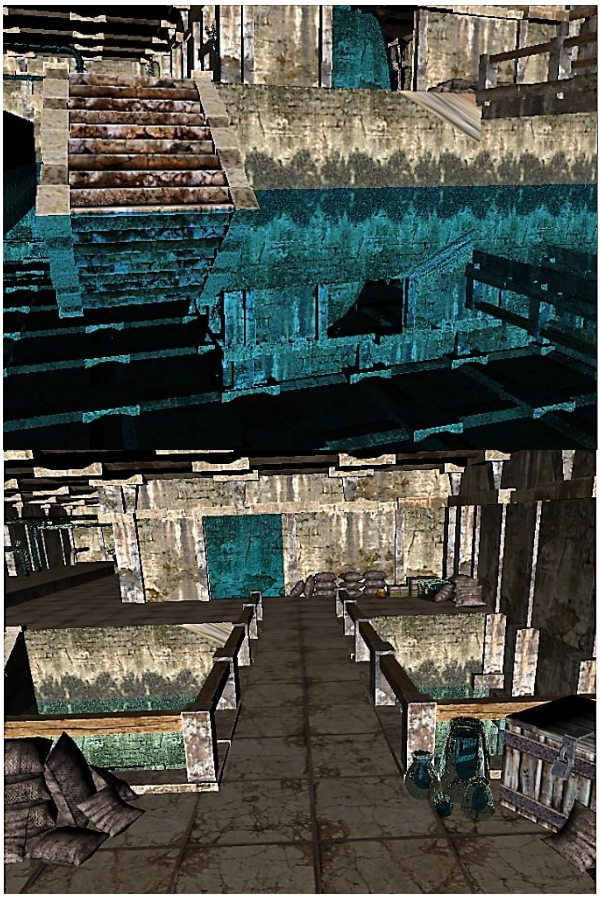

Figure 5: Examples of recursive reflections.

The virtual environments were designed to be traversed in real time (20 frames per second or more). This environment design for real time follows the same principles applied in level design for games: 3d meshes with low polygonal count (low poly meshes), adequate distribution of meshes in the virtual environment and optimized textures. The low poly meshes were bought from the Unity Asset Store [Unity 2014].

Figures 6 and 7 are top and perspective views of the farm and sewer environments, respectively. The farm and sewer use 3d meshes from Cartoon Town and Farm

[K4 2014] and TaD – Sewer Kit [Forge 2014], respectively. The reflexive surfaces in both environments are colored in blue in Figures 5 and 6, but in a true virtual environment, a texture should be applied.

Figure 6: Views of the farm outdoor environment.

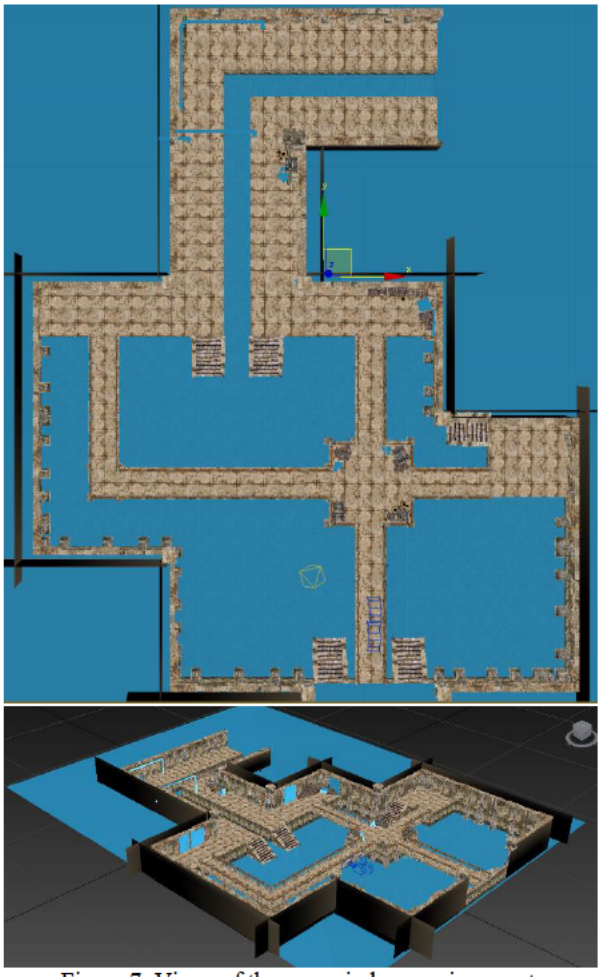

Figure 7: Views of the sewer indoor environment.

In order to test our heuristic (HEU1) qualitatively, we developed a distance heuristic (HEU2). HEU2 select elements that can be rendered in real time and are near the camera. The center of the bounding box around each element is used to measure the distance between the element and de camera.

All the tests were performed in a desktop computer with a 3.50 GHz Intel Core I7-477K CPU and a NVIDIA GeForce GTX 760 GPU. This computer runs Windows 8.1.

The three metrics used to evaluate the proposed heuristic are described as follows. Since the heuristic maintain the minimum of 20 frames per second, the most important tests are qualitative.

5.1 Transitions Between Rendering Methods

When moving inside a virtual environment, the camera can change frequently between ray tracing and raster rendering. Frequent changes can cause poor visual experience. This metric considers the number of changes during each traversal, for each heuristic. Because of its naïve strategy, HEU2 produce many more rendering method changes then HEU1. These changes can easily be perceived when the camera moves forward and backward frequently. For HEU1, previously selected elements have priority to be rendered in ray tracing. Parameters S and I control this priority, reducing the number of changes.

Figure 8 shows a transition between raster, ray tracing and raster again, in order to illustrate the effect of frequent changes in rendering methods.

Figure 8: Changes between raster and ray-trace rendering.

Table 2 present the percentage of transitions produced by the proposed heuristic (HEU1), when compared to the distance heuristic (HEU2), for both environments.

Table 2: Percentage of transitions between heuristics.

Environment	HEU1/HEU2
Outdoor	3,8 %
Indoor	4,6 %

As shown in Table 2, the number of transitions produced by our heuristic is a fraction of the transitions produced by the distance heuristic. The huge difference shows that a good heuristic must have a method to deal with constant changes between rendering methods.

The experiments also shows that HEU1 performs better, when the value of I is, most of the time, bigger

than V. However, if I is always much bigger than V, elements near the camera are never selected.

5.2 Average Number of Elements Selected

The average number of elements selected for ray tracing is an important metric, since it shows how efficient the heuristic is in rendering elements. This metric is affected by the average size of the reflexive surfaces but, in general, the more the number of elements selected, the better the heuristic is.

Table 3 present the average number of elements selected for each environment and heuristic, and the total number of reflexive surfaces.

Table 3: Number of elements selected.

Environment	HEU1	HEU2	Surfaces
Outdoor	5	4	17
Indoor	15	12	33

The relative small number of selections for the outdoor environment is due to the camera path. In a good part of the path, just a small number of reflexive elements are inside the camera view. Also, since both ponds have high priority (high K value), the ponds are rendered most of the time, giving little processing resources for other reflexive surfaces. For the indoor environment, the opposite happens. The water surfaces have low priority, allowing more reflexive elements to be selected. As expected, this observations show how important is a good level design, in order to offer improved visual experience.

5.3 Average Frame Rate

Table 4 present the average frame rate for each environment and heuristic.

Table 4: Average frame rate.

Environment	HEU1	HEU2
Outdoor	25,4	26,4
Indoor	21,8	23,3

HEU2 perform better than HEU1, but this is expected since, as presented in 5.2, HEU2, on average, select fewer elements to render than HEU1. Also, the path has a key influence, as observed in Section 5.2. The indoor path, on average, has more reflexive elements visible to the camera than the Indoor environment.

6 Conclusions

This paper proposes a heuristic to render reflections in real time, using a hybrid renderer. The main advantage of this method is to offer reflections without screen space limitations and true recursive reflections.

The first major conclusion is that true real-time ray-traced reflections are achievable using a well-designed heuristic. The selective rendering approach has the drawback of creating frequent transitions between rendering methods. However, as observed in Section 5.1, the impact of this problem is minimized with a well-designed heuristic, that have a strategy to reduce transitions.

Another conclusion resulted from the visual observation of the real-time rendering of many tests is that the heuristic does good decisions considering elements to render. With the level designer input, the heuristic prioritizes the right elements to render. For the element without high K, most of the time, the heuristic select elements near the camera and the center of the view plane. When moving in real time, it was observed the elements in the corners of the view plane are barely noticeable. Unfortunately, visual quality tests are much more complex to do than numerical tests, and the distance heuristic is too simple to be for visual comparisons. However, when compared to raster only rendering, reflections clearly improve the visual experience.

As future works, there is the possibility that both Unreal Engine 4 [Rege 2014a] and CryENGINE [Rege 2014b] will support OptiX by means of supporting NVIDIA GameWorks [Coombes 2014]. With commercial engines supporting OptiX, it will be possible to migrate the heuristic to these engines and test the heuristic in a real environment designed for games. With this possibility, new heuristics can be developed. In addition, it will be easier to evaluate the visual quality and overall performance impact of heuristic based rendering in the rendering pipeline.

References

Andrade, P., Clua, E., Sabino, T., Forti, F., 2014a. A Heuristic Approach to Render Ray Tracing Effects in Real Time for First-Person Games. SBC J. Interact. Syst.

Andrade, P., Sabino, T., Clua, E., 2014b. 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. SCITEPRESS - Science and and Technology Publications, pp. 12–21. doi:10.5220/0004691300120021

Blinn, J.F., Newell, M.E., 1976. Texture and reflection in computer generated images. ACM SIGGRAPH Comput. Graph. doi:10.1145/965143.563322

Brown, D., Chen, S., 2001. The Architecture of Level Design [WWW Document]. GDC 2001. URL http://www.gamasutra.com/view/feature/3051/gdc_2001_the_architecture_of_.php

Cater, K., Chalmers, A., Ledda, P., 2002. Selective quality rendering by exploiting human inattentional blindness: looking but not seeing, in: Human Factors. ACM, pp. 17–24. doi:10.1145/585740.585744

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

Cater, K.F., 2004. Detail to Attention : Exploiting Limits of the Human Visual System for Selective Rendering. Computer (Long. Beach. Calif). University of Bristol.

Chalmers, A., Debattista, K., Mastoropoulou, G., Paulo, L., 2007. There-Reality : Selective Rendering in High Fidelity Virtual Environments. Int. J. 6, 1–10.

Coombes, D., 2014. Introducing NVIDIA GameWorks [WWW Document]. NVIDIA Website. URL https://developer nvidia.com/content/introducing-nvidia-gameworks (accessed 7.26.14).

Debattista, K., Sundstedt, V., Santos, L.P., Chalmers, A., 2005. Selective component-based rendering. Proc. 3rd Int. Conf. Comput. Graph. Interact. Tech. Australas. South East Asia Graph. 05 13–22.

Debevec, P., 1998. Rendering Synthetic Elements into Real Virtual environments: Bridging Traditional and Image-based Graphics with Global Illumination and High Dynamic Range Photography, in: Siggraph 98. p. 10. doi:10.1145/280814.280864

Debevec, P., 2001. A real time high dynamic range light probe. SIGGRAPH 2001 Tech. Sketch 4.

Deering, M., Winner, S., Schediwy, B., Duffy, C., Hunt, N., 1988. The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics. ACM SIGGRAPH Proc. 22, 21–30. doi:10.1145/54852.378468

Desimone, R., Duncan, J., 1995. Neural mechanisms of selective visual attention. Annu. Rev. Neurosci. 18, 193–222.

Dormans, J., 2010. Adventures in level design: generating missions and spaces for action adventure games. … Work. Proced. Content Gener. Games 1–8. doi:10.1145/1814256.1814257

Feil, J., Scattergood, M., 2005. Beginning Game Level Design, Technology.

Forge, 3d, 2014. TaD - Sewer Kit [WWW Document]. Unity Asset Store. URL https://www.assetstore.unity3d.com/en/#!/content/12867 (accessed 7.26.14).

Green, C.S., Bavelier, D., 2003. Action video game modifies visual selective attention., Nature. Nature Publishing Group.

Greene, N., 1986. Environment Mapping and other Applications of World Projection. IEEE Comput. Graph. Appl. 6, 21–29.

Hasic, J., Chalmers, A., 2009. Saliency in motion, in: Proceedings of the 2009 Spring Conference on Computer Graphics - SCCG '09. ACM Press, New York, New York, USA, p. 173. doi:10.1145/1980462.1980496

Hullett, K., 2010. The science of level design, in: Proceedings of the Fifth International Conference on the Foundations of Digital Games - FDG '10. pp. 262–264. doi:10.1145/1822348.1822387

K4, M., 2014. Cartoon Town and Farm [WWW Document]. Unity Asset Store. URL https://www.assetstore.unity3d.com/en/#!/content/17254 (accessed 7.26.14).

Lever, L., Mcderby, M., Et Al., 2005. Selective Parallel Rendering for High-Fidelity Graphics. Theory Pract.

Ludvigsen, H., Elster, A.C., 2010. Real-Time Ray Tracing Using Nvidia OptiX. Science (80-.). 1–4.

Mania, K., Mourkoussis, N., Zotos, A., 2008. Selective rendering based on perceptual importance of virtual environment regions, 2008 IEEE International Conference on Systems Man and Cybernetics. doi:10.1109/ICSMC.2008.4811515

Parker, S., 2009. Interactive ray tracing with the NVIDIA®OptiX™ engine. SIGGRAPH.

Parker, S., Parker, M., Livnat, Y., Sloan, P.P., Hansen, C., Shirley, P., 1999. Interactive ray tracing for volume visualization. IEEE Trans. Vis. Comput. Graph. 5, 238–250.

Parker, S.G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., Mcallister, D., Stich, M., 2010. OptiX: A General Purpose Ray Tracing Engine. ACM Trans. Graph. TOG, SIGGRAPH '10 29, 1–13. doi:10.1145/1833349.1778803

Pritchard, M., Brooks, J., Geldreich, R., 2004. Deferred Lighting and Shading.

Rege, A., 2014a. How Epic Games Is Putting Power of Unreal Engine 4 Into More Hands Than Ever [WWW Document]. NVIDIA Blog. URL http://blogs.nvidia.com/blog/2014/03/19/epic-games/ (accessed 7.26.14).

Rege, A., 2014b. Crytek Adds NVIDIA GameWorks to "Warface" [WWW Document]. NIVIDA Blog. URL http://blogs.nvidia.com/blog/2014/03/19/crytek/ (accessed 7.26.14).

Sabino, T., Andrade, P., Gonzales Clua, E., Montenegro, A., Pagliosa, P., 2012. A Hybrid GPU Rasterized and Ray Traced Rendering Pipeline for Real Time Rendering of Per Pixel Effects, in: Herrlich, M., Malaka, R., Masuch,

M. (Eds.), Entertainment Computing - ICEC 2012 SE - 25, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 292–305. doi:10.1007/978-3-642-33542-6_25

Sabino, T.L.R., 2012. Uma Arquitetura de Pipeline Híbrida para Rasterização e Traçado de Raios em Tempo Real.

Saito, T., Takahashi, T., 1990. Comprehensible Rendering of 3-D Shapes. ACM SIGGRAPH Comput. Graph., {C}omputer {G}raphics {P}roceedings, {A}nnual {C}onference {S}eries 24, 197–206. doi:10.1145/97880.97901

Sinha, S.N., Kopf, J., Goesele, M., Scharstein, D., Szeliski, R., 2012. Image-based rendering for virtual environments with reflections. ACM Trans. Graph. doi:10.1145/2185520.2335451

Sousa, T., Kasyan, N., Schulz, N., 2011. Secrets of Cryengine 3 Graphics Technology. ACM SIGGRAPH.

Sundstedt, V., Debattista, K., Longhurst, P., Chalmers, A., Troscianko, T., 2005. Visual attention for efficient high-fidelity graphics. Proc. 21st spring Conf. Comput. Graph. SCCG 05 169. doi:10.1145/1090122.1090150

Unity, 2014. Unity Asset Store [WWW Document]. URL https://www.assetstore.unity3d.com (accessed 7.26.14).

Whitted, T., Weimer, D., 1982. A Software Testbed for the Development of 3D Raster Graphics Systems. ACM Trans. Graph. 1, 43–58.

Yan, S., Seif El-Nasr, M., 2006. Visual Attention patterns in Games, in: Symposium of Eye Tracking Applications. pp. 21–24.

Yu, J., Yang, J., McMillan, L., 2005. Real-time reflection mapping with parallax. Proc. 2005 Symp. ….