Comparison of Acceleration Data Structures for High Quality Fast Reflections of Static and Deformable Models in Walkthrough Animations

Daniel Valente de Macedo Maria Andréia Formico Rodrigues
Programa de Pós-Graduação em Informática Aplicada (PPGIA)
Universidade de Fortaleza
Fortaleza-CE Brasil
{danielvalentemacedo, andreia.formico}@gmail.com

***Abstract*—In order to render realistic images, the reflectance of surfaces must be simulated accurately. Generally, the ray tracing rendering technique is used to make a material reflect its surroundings, since it represents with great fidelity the behavior of light. However, ray tracing is still a very costly algorithm, so far mostly indicated in offline rendering scenarios. This situation is even more challenging for scenes containing 3D deformable meshes, since their geometry and, thus, the acceleration structures used, need to be updated in each frame of the animation. In this paper, we present an extended version of our hybrid algorithm that combines rasterization and a pure ray tracing through the NVIDIA *OptiX* to render high quality fast reflections, including scenes with deformable models. Additionally, we analyze and compare the performances of different NVIDIA *OptiX* acceleration data structures for generating reflections of static and deformable models in walkthrough animations. The results show that NVIDIA *OptiX* acceleration structures reach high frames per second for static objects. However, there is a performance decay in terms of frames per second when dealing with deformable models, since it becomes necessary to update the acceleration structures to cope with changing geometry, but even under these restrictions, we were able to achieve interactive frame rates.**

***Index Terms*—realistic rendering; reflections; raytracing; static and deformable models; acceleration data structures**

I. Introduction

Computer graphics technology has advanced to the point where 3D models and visual effects (reflections, lighting, shadows, etc.) can now be rendered with near-perfect photorealism, particularly in offline rendering. However, a major challenge remains in real time computer graphics, where realistic effects must be computed at frame rate.

Regarding reflections, mathematical models are used to describe the appearance of surfaces, being generally part of a rendering engine [1]. This is a major computation task for static objects and even more time consuming for deformable models, since the 3D mesh vertices move in each frame and, thus, need to be re-computed at each step of the animation. Thus, fast and high quality solutions for calculating the amount of light reflected from the visible object surfaces that arrives to the synthetic camera through image pixels are fundamental to realistic rendering.

The rendering engine or "renderer" is responsible for scene rendering using rasterization, ray-tracing, etc. Traditionally, graphical applications based on displaying still-rendered images prioritize visual accuracy, while real-time and interactive applications value real-time performance. More recently, considered attention has been paid to develop new solutions that, at the same time, are fast and can offer good visual quality [2]. The more frames per second (FPS), the smoother the motion appears. With regard to real-time applications, 30 FPS is often a reasonable value to create the illusion of movement while displaying images.

In digital games, these images are normally generated by rasterization [3], a method used to convert geometries formed by vertices, edges and faces, in pixels. Rasterization is an extremely fast process which takes advantage of APIs such as OpenGL [4] and DirectX [4], benefiting directly from hardware (GPU). The downside, however, is that many visual effects generated by rasterization are only rough approximations to reflections, shadows, lighting, etc., since they are not physically based, thus, they do not mimic realistically the behavior of light in the real world.

Ray tracing [5] is a classical algorithm that simulates the paths taken by rays of light as they traverse the scene and interact with it. The use of ray tracing facilitates the simulation of reflection effects, since they are already built into this recursive algorithm. Its advantage over rasterization is the ability to accurately generate sharp reflections, refractions and shadows [6]. Although many optimizations have been made over the years and processor power has increased tremendously, ray tracing is still too performance intense to create high FPS from a scene that constantly changes, but an advantage is that it is also a highly parallelizable algorithm on the GPU. Due to such features, libraries dedicated to GPU ray tracing have been developed very recently, such as the *NVIDIA OptiX* [7], whose ray tracing runs in parallel on the video card through CUDA [8].

To simulate reflections in real-time, game engines use a combination of cubemaps, created from Sphere Reflection Capture with *Screen Space Reflection* (SSR) [9], [10]. Since the SSR works only on screen space, it fails to calculate reflections of objects that lie outside the view frustum or that are occluded by other objects. Moreover, the reflections generated by cubemaps are not realistic, since the synthesized reflections on the objects in the scene come from the same point. Another important limitation when using different cubemaps is the generation of lighting seams between adjacent objects.

In previous work [11], we present a novel hybrid algorithm for rendering high quality fast reflections of static objects in 3D walkthroughs. The algorithm combines SSR and a pure ray tracing through the NVIDIA *OptiX*. The work presented in this paper extends our previous work [11] by dealing with newly implemented features in our hybrid algorithm, which currently also includes not only static meshes but also deformable models. When objects are deformable during animation, it becomes necessary to adapt the acceleration methods to cope with changing geometry. To address this problem, we need to update the position of the vertices every frame in the *OpenGL* and replicate them in the NVIDIA *OptiX*. However, to ensure the correct functioning of the NVIDIA *OptiX* module, we also need to rebuild the acceleration data structure [12] every animation frame. Thus, further extending prior work [11], we currently also focus on using, testing and comparing new ray tracing acceleration structures using NVIDIA *OptiX* during walkthrough animations, for generating reflections of static and deformable models in different test scenarios.

II. Related Work

Realistic Real-time graphics in 3D walkthroughs have been promised for many years. Recently, many game engines (which some are open-source) have been proposed and are available to the gaming community. Some examples are *Blender* [13], *Unity* [14] and *Unreal* [9] engines. Those game engines have been offering new opportunities to anyone interested in building visually realistic graphical applications and, consequently, to advance the state-of-the-art in the field.

The simulation of reflections and refractions in the context of graphical applications in real time has been possible using environment maps and GPU, from the technique introduced by Blinn and Newell [15].

In digital games and interactive 3D applications, it is very common to use a cubemap at the center of the reflective object and in accordance with the normal vectors of the reflection point (the pixel that is being rendered at a given time). A sampling is made in the cubemap [9]. Basically, the cubemap consists of six textures generated from six cameras. Each camera points to a different direction, having the same origin. The visual results obtained using cubemaps are usually convincing. However, since it is a very costly process where it is necessary to render the scene six times from the point of reflection, the cubemap is not generated by each object, nor by each frame at runtime.

An alternative solution is to spread out reflection points in the scene to be rendered and force reflective objects (which are closer) to use the closest reflections for the calculations. This approach also generates a reasonable result, however, physically incorrect, since it is only an approximation. A physically correct way to represent reflection would be to generate it from the surface of the point (or pixel) that is being rendered at a given time.

As regards refractions, Wyman [16] makes real-time refraction look more realistic by introducing a simple, image-space approach that easily runs on modern graphics cards. The method requires two passes on a GPU and allows refraction of a distant environment through two surfaces, compared to current interactive techniques that are restricted to a single surface. Sun *et al.* [17] also presents a technique for simulating light through refractive surfaces using the GPU at interative rates.

In a more recent work, Mara *et al.* [18] use a two-layer deep g-buffer to simulate lighting effects such as mirror reflections. Although their two-layer deep g-buffer captures occluded objects, the reflected object still has to reside within the camera's view *frustum*.

In order to accelerate the process of simulating the reflection, new techniques have been proposed, such as Real-Time Local Reflections or SSR [19], which simulates the light behavior in a manner equivalent to the ray tracing technique by ray marching in the zbuffer. However, it only works on screen space. The algorithm performs similarly to a post-processing image effect, that is, it is independent of the scene complexity since it uses the g-buffer to do the calculations.

Recently, with the advance of GPUs and parallel processing, researchers again turned their attention to ray tracing [20] for interactive simulation of visual effects, at this time, in the context of digital games and graphics applications in real time [21] [22]. Carr *et al.* proposed the first ray tracing algorithm on GPU. However, it actually runs only on GPU the calculation of the intersection between rays and triangles [23]. Subsequently, Purcell *et al.* presented a solution for the generation of rays, traversal, intersection between rays and triangles, shading and creation of secondary rays, all running on separate GPU kernels [24].

Some techniques also use pre-computed maps to assist and accelerate the processing of rendering scenes with ray tracing, or cache data between frames [15], [25]. A graphical application in real time generated with the ray tracing and path tracing algorithms, with rates around 50-60 FPS, has also been reported [26].

Though ray tracing can correctly simulate light reflections, doing it in real time still remains a challenge and, even greater, for deformable models. The most expensive task in a ray tracer is the intersection test. It is a costly operation which is potentially repeated a very large number of times (for each pixel of the frame, we need to test all the triangles from the scene). Most techniques to speed up renders are based on acceleration data structures [12], which need to be updated every frame of the animation. Basically, they are data structures used to organize the objects in a scene, allowing a reduction in the number of tests to identify a set of objects on specific algorithms.The important characteristics of the acceleration structures are construction time, memory use, and ray traversal time.

Wald *et al.* claim in [12] that there is no ideal acceleration structure, able to show the same high performance for all types of 3D scenes, being the developer's responsibility, the choice of the best structure for each scenario. However, the *Bounding Volume Hierarchy (BVH)* structure is reported as the most suitable for scenes with deformable objects or even for dynamic scenes in general [27]. Lauterbach [28] presents how the hierarchy maintenance problem can be addressed using a deformable BVH algorithm that provides a very fast hierarchy refitting to update an existing hierarchy between frames. The described solution can be implemented as a parallel algorithm capable to rebuild BVHs much quicker, providing a fast solution to general dynamic models for ray tracing. An approach to ray trace for animated scenes, with frame rates ranging from 5 to 15 FPS is also presented in [29]. It uses motion decomposition and fuzzy KD-trees for 3D scenes with deformable models. Wald *et al.* have built a two-level BVH with a separate BVH per mesh [30]. Each sub-BVH is constructed using a binned surface area heuristic (SAH) kernel in the case of static meshes, and a refitting kernel in the case of deformable models.

The NVIDIA *OptiX* library [7] has emerged to assist programmers in developing more visually realistic graphical applications with ray tracing in real-time in GPU. This library does not necessarily focus on 3D scenes rendering. Actually, it corresponds to a graphics pipeline for a *shader-centric* ray tracing in which shaders or programs can be written and inserted in different points of the pipeline and can be run almost entirely on GPU. OpenRL library [31] has also emerged to with focus on ray tracing programming using a OpenGL inspired API, making it easier to learn for experienced graphics programmers. Besides that, it is already integrated into *Unity 5* [14]. Another library has recently been launched for Intel CPUs, called *Embree* [32].

Very recently, the use of hybrid solutions combining ray tracing and rasterization have also been explored. For example, in [33], the reflection simulation is done by approximating the SSR technique by using a bounding cube with, which contains associated buffers to their faces representing the scene, *i.e.*, such information is used as a backup way to correct the flaws generated by the SSR algorithm. Hakura and Snyder [34] introduce a hybrid rendering. Actually, a scheme that dynamically raytraces near geometry of reflective and refractive objects. However, it approximates more distant geometries by using hardware supported environment maps in a preprocessing stage. Finally, the authors of [35] propose a hybrid solution that uses a simple heuristic to ignore irrelevant objects in the effects calculation phase using ray tracing. Also in a hybrid form, Cabeleira [6] presents a solution in which all global illumination effects are generated by ray tracing. Part of the process is done on the CPU and another on GPU level. In the end, all this information is combined to generate the resulting image. Ganestam and Doggett [36] describe a solution in which the visual effects of objects (that are located close to the camera), stored in a customized Bounding Volume Hierarchy (BHV), are processed by ray tracing and other remaining objects that are distant from the camera's field of view are calculated by rasterization.

III. The NVIDIA OptiX Library

The *NVIDIA OptiX* library can implement many types of renderers. It is a programmable ray tracing framework for software developers. It is used to rapidly build ray tracing applications that yield extremely fast results across *NVIDIA* GPUs, with CUDA C/C++ programming. A call graph showing the control flow through the *NVIDIA OptiX* ray tracing pipeline is shown in Figure 1.

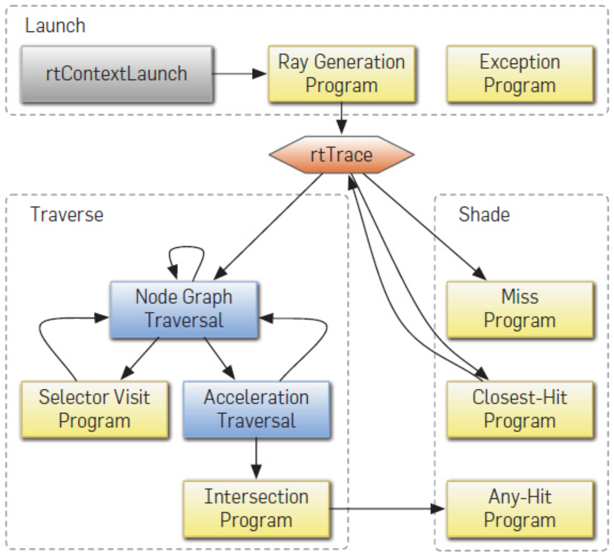


Fig. 1. The *NVIDIA OptiX* ray tracing pipeline.

This library presents mechanisms for expressing ray-geometry interactions and does not have built-in concepts of lights, shadows, reflectance, ambient occlusion or any other feature for scene rendering.

Its main interests are ray generation, acceleration data structures and how to represent the scene and the rays path in the scene. *NVIDIA OptiX* follows the same idea of *OpenGL*, which means that it is possible to insert, at any time, customized programs or shaders to inform what information should be processed through the graphics pipeline.

Currently, there are 8 different types of programs that can be used with *NVIDIA OptiX* [7], briefly described as follows: (1) *ray generation*: the whole pipeline starts in this program, where is implemented the creation of rays in the scene; (2) *intersection*: responsible for the implementation and collision detections between rays and scene geometries (spheres, cylinders, planes, etc.); (3) *closest hit*: called as soon as the scene traversal algorithm detects the nearest collision point from the origin of the ray; (4) *any hit*: called when a new ray is generated from the closest hit program, which collides with any geometry (the creation of rays is finalized when a secondary ray is generated for calculating shadows in the scene); (5) *miss*: called when a ray does not collide with any scene geometry; (6) *exception*: it is used to fix errors during the generation of rays (e.g., memory leaks or incorrect access of index in the buffer); (7) *selector visit*: controls the traversal of rays in the scene structure; (8) *bounding box*: responsible for calculating the area associated with each primitive, enabling the use of acceleration data structures on any geometry. Figure 1 illustrates the execution flow of an application that uses *NVIDIA OptiX*.

The *NVIDIA OptiX* engine also offers some methods that can be used to construct the ray tracing acceleration structure, which is only valid with a correct pair of builder and traverser (more details can be found in [7]). A builder can take one of the following values: *NoAccel* (specifies that no acceleration structure is explicitly built), *Bvh* (a standard bounding volume hierarchy, useful for most types of graph levels and geometry), *Sbvh* (a high quality *Bvh* variant for maximum ray tracing performance, with slower build speed and slightly higher memory footprint than *Bvh*), *MedianBvh* (a medium quality *Bvh* with quick build performance, useful for dynamic and semi-dynamic content), *Lbvh* (a simple *Bvh* with very fast build performance, useful for dynamic content), *Trbvh* (a high quality *Bvh*, very much alike *Sbvh* but with fast build performance similar to *Lbvh*), and *TriangleKdTree* (a high quality kd-tree builder, for triangle geometry only, that may provide better ray tracing performance than the *Bvh* builders for some scenarios).

In the work of [37], different test cases are presented and compared to verify the robustness of the implementations of existing structures and libraries for the process of generating shadows at run time with the ray tracing.

IV. The Generation of Real-Time Reflections

In this section, we describe in general terms the simple although effective hybrid solution we have proposed and implemented to generate reflections (Figure 2) for static and deformable objects in real-time.

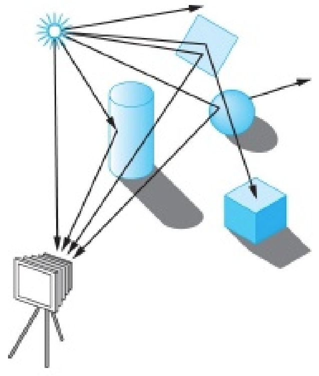


Fig. 2. Rays of light from a point source.

Initially, the algorithm loads an *obj* file with the 3D scene information to be rendered. Then, we create the g-buffer with the diffuse color, worldspace, normal, z-buffer and reflection information. More specifically, the reflection calculation consists of two steps: (1) the classic SSR algorithm [19]; and (2) a pure ray tracer. The complete execution flow diagram of the rendering engine is shown in Figure 3.

During the first step, we generate two images, one that contains the reflection result based on the reflection information sampled from the g-buffer, as shown in Figure 3.b, and a mask that represents in which parts of the image the screen-space method for adding reflections failed, highlighted in Figure 3.c. After that, we start the second step, in which the following information is passed: the 3D scene geometry and the g-buffer.

The ray tracer verifies each pixel of the mask generated during the first step, searching for pixels that have been marked but were not calculated correctly. If any pixel satisfying this situation is found, a reflected ray from the viewer's eye is calculated and ray traced, resulting in a new complementary reflection image, as shown in Figure 3.d. As a result, the reflection of objects that lie outside the screen-space or that are occluded by other objects can be processed.

With the complementary image completion, our engine combines this image with its screen-space version and applies a *Gaussian blur filter* [38] to blur the reflection image and minimize small artifacts that may appear due to the merging process of the SSR with the raytraced reflections. We also apply, in both steps, an attenuation to the reflection based on the length of the reflection ray from the reflected surface (making it fade out along the reflection) for generating a polished visual result, shown in Figure 3.e.

The process to generate shadows is done just before the simulation of reflections. Our algorithm uses a zbuffer cube-map to calculate the shadows emitted from a point light. A mask of pixels that are suffering occlusion of light is created, *i.e.*, an image with value 0 for pixels that are in shadow and 1, for otherwise. This mask can be seen in Figure 3.a. Finally, all the lighting is calculated and merged with the reflection and shadow masks, resulting in the final composition of the scene.

For the deformable models, whenever the vertices of the objects are modified, we replicate this change in both the vertex buffers of the *OpenGL* and *NVIDIA OptiX*. Following that, we rebuilt/upgrade the acceleration structure currently being used. This process is performed at the beginning of each frame, whenever necessary.

V. Tests and Performance Results

All the rendering tests were performed on an Intel Core i7 with 3.40GHz and 16GB of RAM, using an *Nvidia GeForce GTX 780 TI* graphics card. More specifically, the performance testings were divided into two phases and were conducted with a walkthrough in the *Sponza* 3D scene from Crytek [39] with 6 additional mesh objects positioned in the middle of the central corridor (4 armadillos and 2 T-rex dinosaurs, with 35k and 20k vertices each, respectively), which totalizes 330k vertices in the whole scene.

In the first phase of the testings, we analyzed and compared the performance of the acceleration structures (*Lbvh*, *Bvh*, *MediamBvh*, and *KD-tree*) available in the *NVIDIA OptiX* for generating reflections for static and deformable objects,

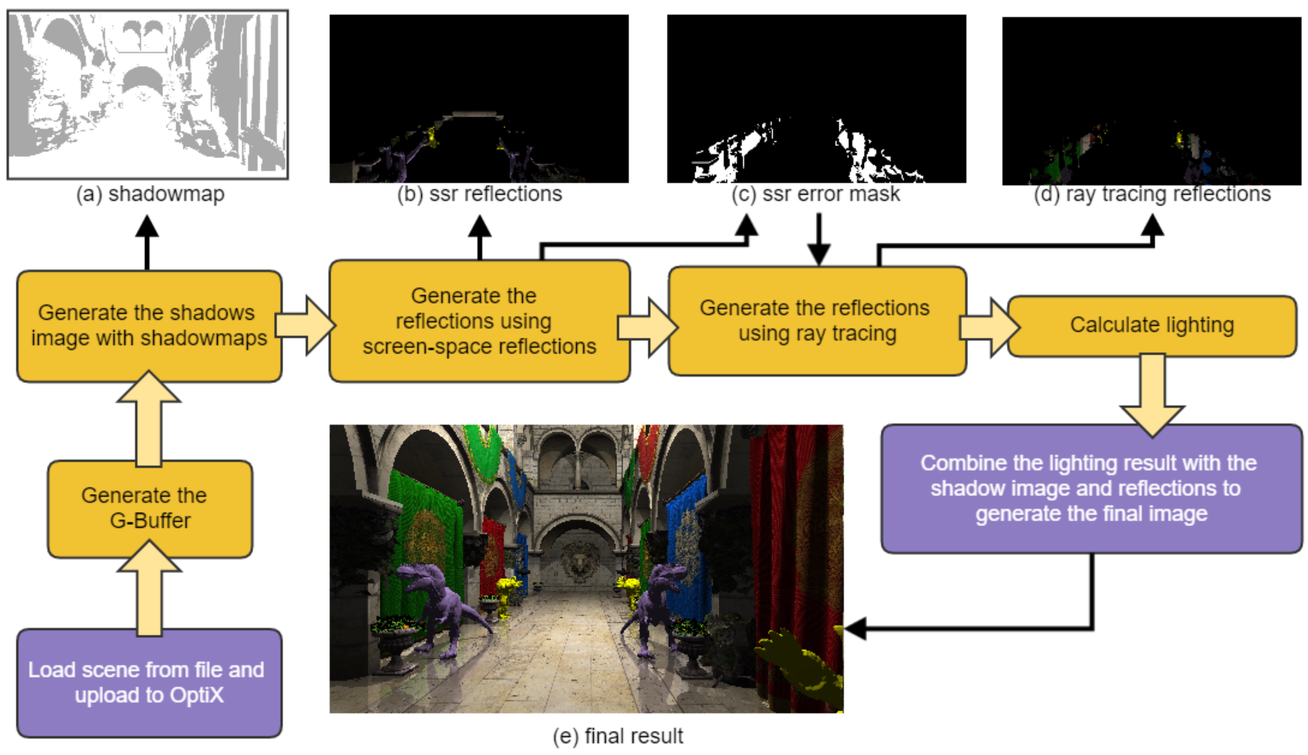

(a) shadowmap
(b) ssr reflections
(c) ssr error mask
(d) ray tracing reflections
Generate the shadows image with shadowmaps
Generate the reflections using screen-space reflections
Generate the reflections using ray tracing
Calculate lighting
Generate the G-Buffer
Load scene from file and upload to OptiX
Combine the lighting result with the shadow image and reflections to generate the final image
(e) final result

Fig. 3. Execution flow diagram of our rendering engine.

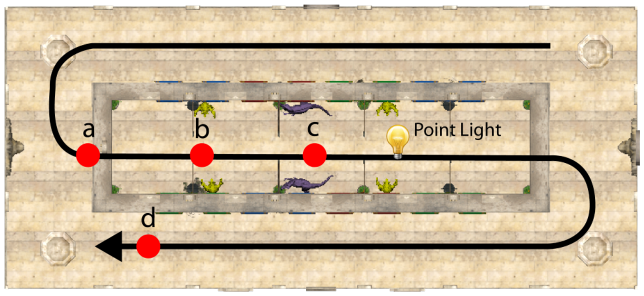


Fig. 4. Floor plan of the scene used in the tests, showing the 3D meshes of the armadillo and T-Rex dinosaurs, the light point position and the camera path: (a) Frame 1800, (b) Frame 2100, (c) Frame 2300 e (d) Frame 4400.

by using two different versions of the 3D scene. The first one, with 330k vertices (original mesh); and, the second, a simplified version of the first scene, with 182k vertices (the same scene with 4 armadillos and 2 T-rex dinosaurs, but now with 3.5k and 7.5k vertices each, respectively). In the second phase, we focused on the quality and performance of our hybrid solution for generating scenes with reflections, shadows and lighting in real time.

For our benchmark, we have used images with a resolution of 1280 × 720 pixels and a camera in motion simulating a FPS gameplay. The 3D scene contains 618,000 triangles and 21 different 512 × 512 textures. The floor reflection properties were enabled and there is a point light source in constant motion along the central corridor. Figure 4 features a floor plan representation of *Sponza*, which illustrates the position of the 3D scene elements and the camera path.

The walkthrough animation was defined in such a way that the three halls of the *Sponza* are regions that belong to the camera path. The path's control points form a S-shaped curve, modeled by a *Catmull-Rom* spline. The animation along this path consists of 4,900 frames. To plot the results, we consolidate the numerical values obtained by considering intervals of 10 frames to calculate the average performance.

In Figure 5 we can analyze the result of the first comparative test of the acceleration structures for the original mesh, where all objects are static. We can see that *NVIDIA OptiX* can achieve competitive results for all the acceleration structures considered in our tests, since all of them, the *Lbvh*, *Bvh*, *MediamBvh*, and *KD-Tree* structures, respectively, reach a very similar average performance of 16ms, 15.6ms, 15.8ms and 15.3ms. In particular, for the static scene, the KD-tree had a slightly better result, showing that it is a good acceleration structure choice for static scenes.

For the second comparative performance testings of the acceleration structures, we have applied a sine wave algorithm to modify all the vertices of the two dinosaurs each frame during the animation, that is, we had to update 40,000 vertices each frame, requiring a rebuild of the acceleration structure each frame, as shown in Figure 6. In Figure 7, we can note that there is a considerable performance drop, with average rendering time values of 85.4ms, 163.0ms, 94.7ms and 159.1ms for the

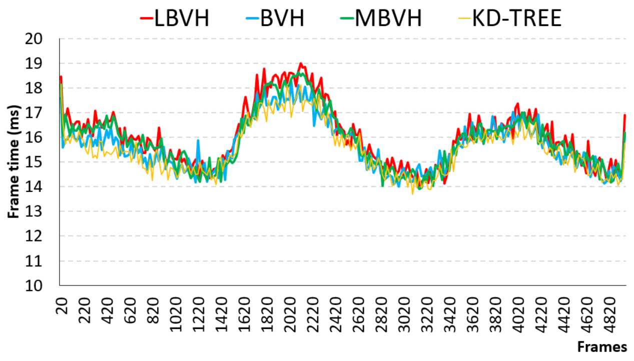


Fig. 5. Comparison of different acceleration data structures for reflections of static models in the walkthrough animation.

Lbvh, *Bvh*, *MediamBvh* and *KD-Tree* structures, respectively.

Considering the performance drop when using deformable meshes, we reduced the number of polygons of each dinosaur from 20k to 7.5k and of each armadillo from 34k to 3.5k, and rerun the static scene testings with this more simplified geometric version of meshes. In Figure 8, it is possible to verify that the *NVIDIA OptiX* still reaches excellent results for all acceleration structures, obtaining average values of 13.2ms, 12.4ms, 12.7ms, and 12.3ms, respectively, for the *Lbvh*, *Bvh*, *MediamBvh* and *KD-Tree* structures.

Following that, we run the performance testings with deformable models and the whole *Sponza* scene, but this time using the simplified geometric version of the armadillo and dinosaurs objects. Figure 9 shows an improvement in performance, with average values of 66.1ms, 96ms, 67.7ms, and 96.4ms for the *Lbvh*, *Bvh*, *MediamBvh* and *KD-Tree* structures, respectively. The performance decay remains accentuated, however, running the tests with a reduced vertex count on the deformable meshes, the results become much more acceptable.

To verify the reconstruction processing time of the acceleration structures in a more precise way, we ran strictly the ray tracing test to process only the reflections of the deformable dinosaurs meshes (*i.e*, the *NVIDIA OptiX* only processed the dinosaurs for reflections). Figure 10 shows performance results of 29.8ms, 49.8ms, 28.5ms and 49.9ms for the *Lbvh*, *Bvh*, *MediamBvh* and *KD-Tree* structures, respectively. Performance gains of approximately 55%, 48%, 57% and 48%, respectively, for the *Lbvh*, *Bvh*, *MediamBvh* and *KD-Tree* structures were obtained when compared with the performance testings done with the simplified geometric version of the scene with two deformable dinosaur meshes. Thus, we can conclude that though this result may be acceptable, the reconstruction of the *NVIDIA OptiX* acceleration structures are not recommended to be rebuilt every frame in real-time applications.

Tables I and II show a comparative study we have conducted to identify how similar the walkthrough frames generated using the SSR technique and our hybrid solution are, when compared to the rendering results obtained from the purely ray tracing algorithm (which is considered in this work as a benchmark, since it generates the most accurate and realistic visual results). Taking into account the visual similarity between the images, to calculate the visual quality differences among them in a quantitative manner, we used a simple technique of color buckets, as follows. For each image, we test each pixel and for each channel of this pixel, the obtained value is added to the corresponding bucket (R, G or B). After processing the entire image, the total value for each component is compared to the total image value generated by the ray tracing. We obtain the percentage value that represents how many pixels from one image differ from the other. Using these results, we calculate the level of similarity between the generated SSR and Hybrid images, and the reference image (synthesized with ray tracing).

In Table I and II we can see that our hybrid solution generates visual quality results always very close to 100%, whereas the SSR in some cases reaches around 97%. This difference of 3% can not be considered, in anyway, as unrepresentative, because the visual quality tests took into account the whole picture and not just the image parts that are suffering reflection.

Figure 11 shows the acceleration structures behavior for the majority of the performance testings we have performed. We conclude that we can not affirm with 100% of certainty that they always you have this behavior for any test scenario, but we have obtained similar results with those presented by the *NVIDIA OptiX*, showing that the *KD-Tree* structure is the slowest to rebuild its structure and the *Lbvh* is the fastest, being the reverse, respectively, also true for the rendering time.

For generating a comparative analysis of the performance of our hybrid solution for static and deformable meshes against other two solutions (one of them purely using SSR and another using ray tracing), we created a chart that represents the frame time need to be produced in each frame during the animation, which is shown in Figure 12, using the simplified version of the static scene and the *KD-tree* structure.

As we can see, our hybrid solution to render reflections shows a better performance than the solution using ray tracing and that, at several consecutive frames of the walkthrough, it competes fairly with the SSR technique in both scenarios, even though having a lower relative performance for deformable meshes, it still reaches acceptable interactive frame rates.

In the FPS chart, there are three ranges of key frames that show important characteristics and, thus deserve to be discussed: (1) [207, 1443]; (2) [1649, 2576]; and (3) [3503, 4224].

In the first and third intervals, during the animation, the camera is pointing to the side aisles of the scene. The reflection generated on the corridor floor is made up of simple elements that do not suffer occlusion, which helps the SSR to simulate a large part of the reflections quickly, leaving little scene portions to be rendered by the ray tracing algorithm. That is, that makes our solution more competitive, since it reaches FPS results close to those obtained using a pure SSR technique.

Most importantly, in the second interval, the camera is pointed at the central corridor while it moves along the animation path where the armadillos and dinosaurs are positioned. This is a very interesting take of the animation since those characters attract great visual attention of the beholder.

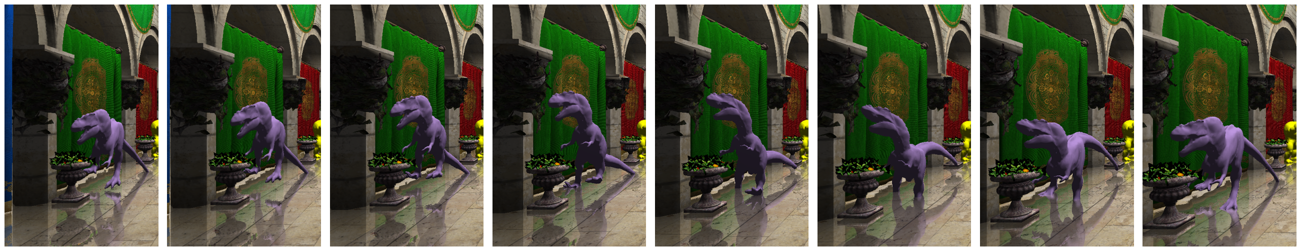


Fig. 6. Animation sequence showing some keyframes of the walkthrough with a camera zoom-in on the reflections produced around one of the deformable T-rex dinosaur.

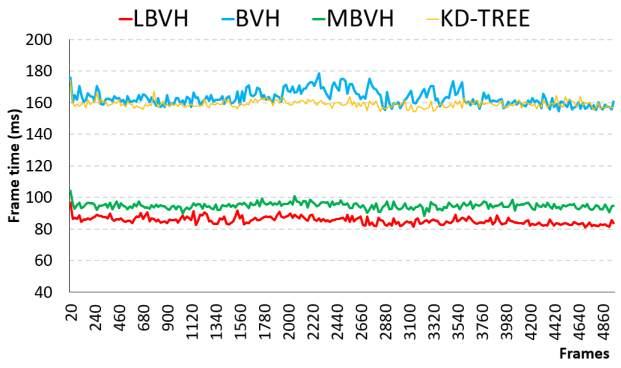


Fig. 7. Comparison of different acceleration data structures for reflections of static models and two deformable T-Rex dinosaurs, in the walkthrough animation.

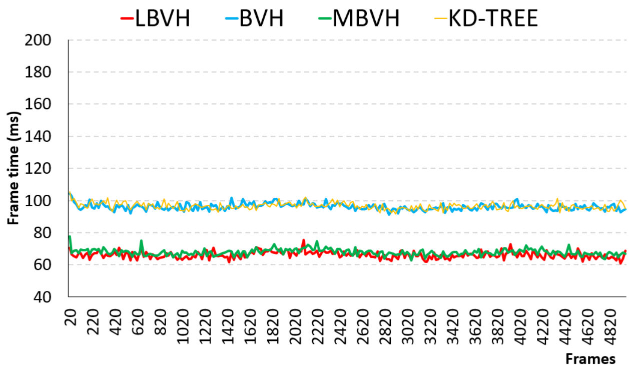


Fig. 9. Comparison of different acceleration data structures for reflections of a simplified geometric version of the static scene and two deformable T-Rex dinosaurs, in the walkthrough animation.

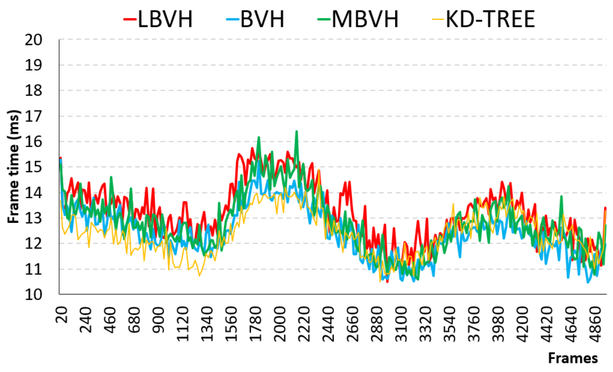


Fig. 8. Comparison of different acceleration data structures for reflections, with a simplified geometric version of the static scene, in the walkthrough animation.

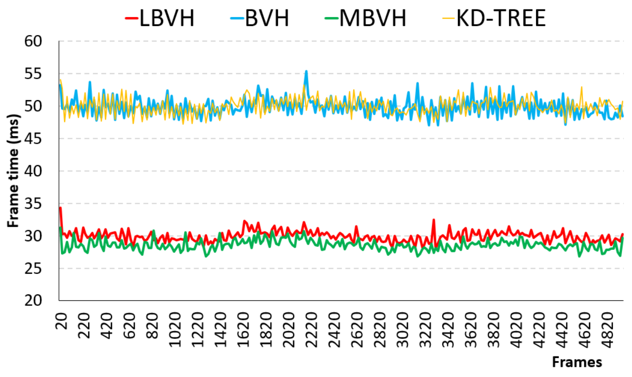


Fig. 10. Comparison of different acceleration data structures for reflections including only the two deformable T-Rex dinosaurs and ignoring all the rest of scene, in the walkthrough animation.

The floor reflections represent several objects, including those that have complex geometries. This situation is very common in animations and it is inevitable the occurrence of occlusions of parts of objects. In such cases, the SSR algorithm fails to represent the reflections efficiently, generating far too many failures.

Our solution detects these SSR failures at runtime and activates the ray tracing algorithm to solve them. Because of this, our solution decays a bit in terms of FPS. However, even then it still reaches a FPS far superior to the solution purely implemented in ray tracing in both tests.

Based on these findings, we conclude that our solution achieves frame rates per second higher than the rates obtained purely with ray tracing. Moreover, we can also clearly note in Figure 13 that the visual quality of the animation frames we generate with our solution is very close to the quality of those generated with ray tracing.

VI. Conclusions and Future Work

In this paper we analyze and compare the performances of different *NVIDIA OptiX* acceleration data structures for generating reflections of static and deformable models using

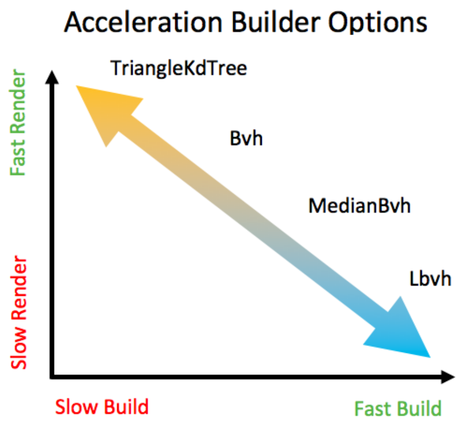


Fig. 11. Default behavior of the acceleration structures, verified in our performance testings using the *NVIDIA OptiX*.

TABLE I
IMAGE QUALITY RESULTS OF FOUR DIFFERENT FRAMES FROM THE WALKTHROUGH SEQUENCE, IN TERMS OF REFLECTIONS, COMPARED TO THE IDEAL CORRESPONDING IMAGES GENERATED BY THE RAY TRACING WITH STATIC OBJECTS .

#Frame	**SSR Algorithm (%)**	**Our Solution (Hybrid) (%)**
1800	99,18	99,76
2100	97,68	99,85
2300	98,58	99,96
4400	99,88	99,97

our new hybrid solution for simulating realistic reflections in 3D walkthrough environments.

Our comparative study shows that *NVIDIA OptiX* acceleration structures can reach high frames rate per second for static objects. However, there is a performance decay in terms of frames per second when dealing with deformable models. Unlike for static models, the largest problem in ray tracing of deformable models is not memory, but maintenance of the acceleration structure. As hierarchies typically become invalid when geometric objects move or deform, the bottleneck in ray tracing shifts towards computing a new hierarchy after each deformation. But even under these restrictions, we were able to achieve interactive frame rates. The tests also show that our hybrid solution presents quality results very close to those obtained with a purely ray tracing algorithm, with a competitive FPS, regardless of the animation moment. We believe that it is still possible to make optimizations in the rendering engine, for example, to support baking lighting and shadow for lights and static objects, besides of calculating in real time only what is really necessary to guarantee the visual quality and performance of the animation.

TABLE II
IMAGE QUALITY RESULTS OF FOUR DIFFERENT FRAMES FROM THE WALKTHROUGH SEQUENCE, IN TERMS OF REFLECTIONS, COMPARED TO THE IDEAL CORRESPONDING IMAGES GENERATED BY THE RAY TRACING, WITH THE SIMPLIFIED GEOMETRIC VERSION OF THE *Sponza* SCENE AND THE TWO DEFORMABLE MESHES OF THE T-REX DINOSAURS.

#Frame	**SSR Algorithm (%)**	**Our Solution (Hybrid) (%)**
1800	98.75	99.66
2100	97.60	99.75
2300	97.55	99.93
4400	99.88	99,97

As future work, we intend to expand our testings for highly dynamic scenes with reflective objects and also to check the possibility of applying some heuristics to consider (or not) the use of the ray tracing algorithm. It may be possible with a heuristic, for example, to make the ray tracing raytraces the pixel only when needed. For example, if the camera is moving very fast in a game, it is probably not fundamental to produce extremely realistic reflections, since the player probably will not be able to identify failures and gaps on the fly easily. That is, it may be interesting to develop also an adaptive solution for our hybrid algorithm directed by the player's behavior. We also plan to conduct more precise tests to ensure the quality of the images generated by our solution, for example, using some specific metrics such as SSIM [40] and RMSE [41].

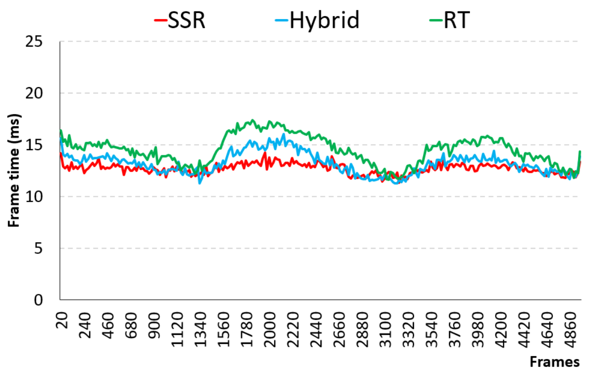
In the near future, another improvement we plan is to develop a customized ray tracer on GPU using CUDA. Accordingly, we also aim at including specific data structures into our hybrid solution. Unfortunately, as a closed-source library, the *NVIDIA OptiX* does not allow to customize the testing codes, forcing the developers to use only the data structures currently existing in the library.

ACKNOWLEDGMENT

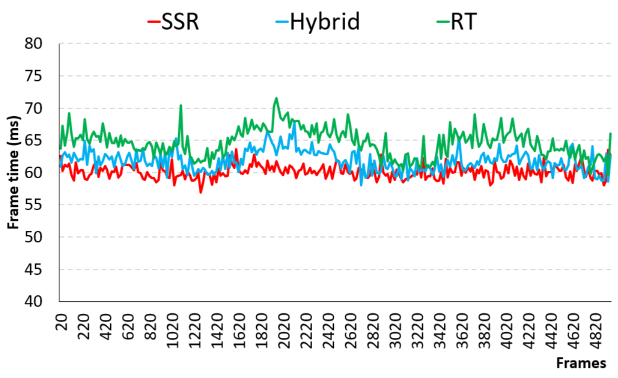
Daniel Valente de Macedo and Maria Andréia Formico Rodrigues are supported by CAPES and CNPq, under grants No. 157.257/2012-6 and 481326/2013-8, respectively, and would like to thank for their financial support.

REFERENCES

[1] J. Gregory, *Game Engine Architecture*. A.K. Peters, 2009.
[2] T. Akenine-Moller, E. Haines, and N. Hoffman, *Real-time Rendering*. A.K. Peters Ltda, 2008.
[3] V. Popescu and P. Rosen, “Forward Rasterization,” *ACM Transactions on Graphics*, vol. 25, no. 2, pp. 375–411, 2006.
[4] OpenGL, “OpenGL,” URL: https://www.opengl.org, 2015.
[5] S. G. Parker, H. Friedrich, D. Luebke, K. Morley, J. Bigler, J. Hoberock, D. McAllister, A. Robison, A. Dietrich, G. Humphreys, M. McGuire, and M. Stich, “GPU Ray Tracing,” *Communications of the ACM*, no. July, pp. 93–101, 2013.
[6] J. Cabeleira, “Combining rasterization and ray tracing techniques to approximate global illumination in real-time,” Master's thesis, MSc Thesis in Engenharia Informática e de Computadores, Universidade Técnica de Lisboa, Portugal, 2010.
[7] NVidia, “OptiX,” URL: http://www.nvidia.com/object/optix.html, 2015.a.
[8] ——, “CUDA Toolkit,” URL: https://developer.nvidia.com/cuda-toolkit, 2015.b.
[9] Epic Games, “Unreal Engine 4,” URL: https://www.unrealengine.com, 2016.
[10] D. V. Macedo, Y. R. Serpa, and M. A. F. Rodrigues, “Desenvolvimento de Aplicações Gráficas Interativas com o Unreal Engine 4,” *XXVIII Conference on Graphics, Patterns and Images*, 2015.

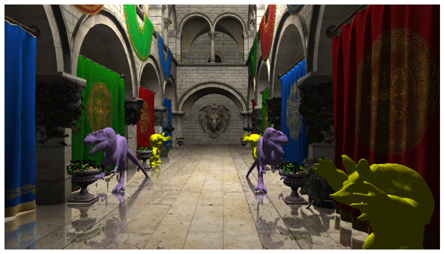


(a)

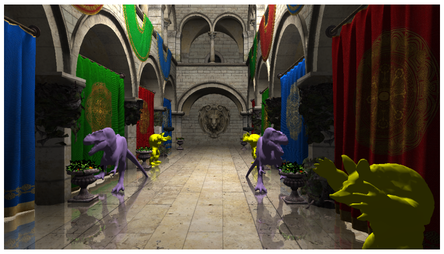


(b)

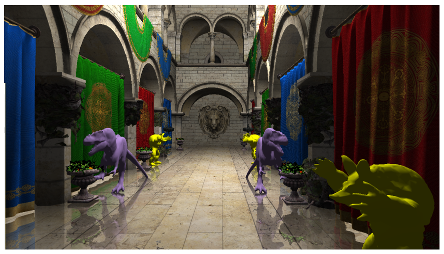
Fig. 12. Comparisons of the animation rendered using the SSR technique, our hybrid solution and ray tracing using KD-Tree for (a) static and (b) deformable meshes.



(a)



(b)



(c)

Fig. 13. Visual quality comparisons of the animation rendered using the (a) SSR technique, (b) our hybrid solution and (c) ray tracing from the deformable test.

[11] D. V. Macedo and M. A. F. Rodrigues, "Realistic Rendering in 3D Walk-throughs with High Quality Fast Reflections," *XIV Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGames), Trilha de Computação*, pp. 9–15, 2015.

[12] I. Wald, W. Mark, J. Gunther, S. Boulos, T. Ize, W. Hunt, S. Parker, and P. Shirley, "State of the Art in Ray Tracing Animated Scenes," *Computer Graphics Forum (CGF)*, vol. 28, no. 6, pp. 1691–1722, 2009.

[13] Blender, "Blender Game Engine," URL: https://www.blender.org, 2015.

[14] Unity Technologies, "Unity 5," URL: https://www.unity3d.com, 2015.

[15] J. F. Blinn and M. E. Newell, "Texture and reflection in computer generated images," *Communications of the ACM*, vol. 19, no. 10, pp. 542–547, 1976.

[16] C. Wyman, "An approximate image-space approach for interactive refraction," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 1050–1053, 2005.

[17] X. Sun, K. Zhou, E. Stollnitz, J. Shi, and B. Guo, "Interactive relighting of dynamic refractive objects," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 35:1–35:9, 2008.

[18] M. Mara, M. McGuire, and D. Luebke, "Lighting deep g-buffers: Single-pass, layered depth images with minimum separation applied to indirect illumination," Tech. Rep., 2013.

[19] M. Mcguire, "Efficient GPU Screen-Space Ray Tracing," *Journal of Computer Graphics Techniques*, vol. 3, no. 4, pp. 73–85, 2014.

[20] T. Whitted, "An improved illumination model for shaded display," *Magazine Communications of the ACM*, vol. 23, no. 6, pp. 343–349, 1980.

[21] J. Bikker, "Real-time ray tracing through the eyes of a game developer," in *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, 2007, pp. 1–10.

[22] D. Pohl, "Wolfenstein: Ray traced," URL: http://www.wolfrt.de, 2014.

[23] N. A. Carr, J. D. Hall, and J. C. Hart, "The Ray Engine," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, 2002, pp. 37–46.

[24] T. J. Purcell, P. Hanrahan, I. Buck, and W. R. Mark, "Ray Tracing on Programmable Graphics Hardware," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 703–712, 2002.

[25] G. S. Miller and C. R. Hoffman, "Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments," in *Proceedings of the 1984 SIGGRAPH - Course Notes on Advanced Computer Graphics Animation*, 1984.

[26] E. Lafortune and Y. Willems, "Bi-directional path tracing," in *Proceedings of the 3rd International Conference on Computational Graphics and Visualization Techniques*, 1993, pp. 145–153.

[27] I. Wald, S. Boulos, and P. Shirley, "Ray tracing deformable scenes using dynamic bounding volume hierarchies," *ACM Transactions on Graphics*, vol. 26, 2007.

[28] C. Lauterbach, "Interactive ray tracing of massive and deformable models," Master's thesis, Dissertation for the degree of Doctor of Philosophy in the Department of Computer Science, University of North Carolina, 2010.

[29] J. Günther, H. Friedrich, I. Wald, H. Seidel, and P. Slusallek, "Ray Tracing Animated Scenes using Motion Decomposition," in *Proceedings of Eurographics 2006*, 2006.

[30] I. Wald, C. Benthin, and P. Slusallek, "Distributed interactive ray tracing of dynamic scenes," in *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, 2003, pp. 11–20.

[31] Imagination, "OpenRL," URL: http://community.imgtec.com/developers/powervr/openrl-sdk, 2015.

[32] I. Wald, S. Woop, C. Benthin, G. Johnson, and M. Ernst, "Embree: A Kernel Framework for Efficient CPU Ray Tracing," *ACM Transactions on Graphics*, vol. 33, no. 4, 2014.

[33] M. Johnsson, "Approximating ray traced reflections using screenspace data," *MSc Thesis in Computer Science: Algorithms, Languages and Logic, Chalmers University of Technology*, 2012.

[34] Z. S. Hakura and J. M. Snyder, "Realistic reflections and refractions on graphics hardware with hybrid rendering and layered environment maps," in *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, 2001, pp. 289–300.

[35] 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.

[36] P. Ganestam and M. Doggett, "Real-time multiply recursive reflections and refractions using hybrid rendering," *The Visual Computer*, pp. 1–9, 2014.

[37] D. V. Macedo and M. A. F. Rodrigues, "A Hybrid Rendering Engine for Generating Real-Time Dynamic Shadows in Computer Games," *XIII Simpósio Brasileiro de Jogos e Entretenimento Digital, Trilha de Computação*, vol. 1, pp. 938–941, 2014.

[38] L. G. Shapiro and G. C. Stockman, *Computer Vision*. Prentice Hall, 2001.

[39] Crytek, "Sponza model," URL: http://www.crytek.com/cryengine/cryengine3/downloads, 2010.

[40] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[41] R. Hyndman and A. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, pp. 679–688, 2006.