Centro de Informática U·F·P·E

Pós-Graduação em Ciência da Computação

"An Authoring Tool of Photo-Realistic Virtual Objects for Augmented Reality"

Por

Guilherme de Sousa Moura

Dissertação de Mestrado

VIRTUS IMPAVIDA

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, FEVEREIRO/2010

FEDERAL UNIVERSITY OF PERNAMBUCO

COMPUTER SCIENCE CENTER

POST-GRADUATION IN COMPUTER SCIENCE

GUILHERME DE SOUSA MOURA

"AN AUTHORING TOOL OF PHOTO-REALISTIC VIRTUAL OBJECTS FOR AUGMENTED REALITY"

THIS DISSERTATION HAS BEEN SUBMITTED TO THE COMPUTER SCIENCE CENTER OF THE FEDERAL UNIVERSITY OF PERNAMBUCO AS A PARTIAL REQUIREMENT TO OBTAIN THE DEGREE OF MASTER IN COMPUTER SCIENCE

SUPERVISOR: Prof. Dr. Judith Kelner

RECIFE, FEBRUARY/2010

Moura, Guilherme de Sousa
An authoring tool of photo-realistic virtual objects for augmented reality / Guilherme de Sousa Moura. - Recife: O autor, 2010.
xi, 69 folhas: il., fig., tab.

Dissertação (mestrado) - Universidade Federal de Pernambuco. CIN. Ciência da Computação, 2010.

Inclui bibliografia.

1.Computação. 2. Computação gráfica. 3. Programação (computadores). I.Título.

005 CDD (22.ed.) MEI-2010-026

ACKNOWLEDGMENTS

I would like to thank, first of all, Professor Judith for all the ideas and support during this dissertation. The advices and corrections were extremely important for me and the development of this work.

Thanks to professors Judith and Djamel for also allowing me to develop my work and building the premium infrastructure in Recife.

Many thanks to my co-worker Saulo Pessoa for helping me in the development of this whole project and for the knowledge sharing about many related aspects.

Thanks to João Paulo for the support in this project management, dedicating his time to constantly remind me of my activities.

I would like to thank also my fellow co-workers (Joma, Pedro, Mozart, Daliton, Artur, Ronaldo, Gabriel, Chico, Andrea, and many others) in GRVM (Virtual Reality and Multimedia Research Group) that offered support for this work and made my time cheerful.

Many thanks to Sarah, for all the love and support, that helped me in the most harsh and stressful moments. Sometimes it is necessary to forget your duties and dedicate some time to the people you love.

Finally many thanks to my entire family. First to my mother Victória, who helped me even in the moments where I didn't want any help, and even if I couldn't explain my difficulties. Thanks to my brother Paulo Victor, my sisters, Daniela and Mariana, to my step-father Renato and my step-brothers, Renato, Renê and Renan. Finally thanks to my father, Paulo, who taught me the most important things in life, without them I could not complete this achievement.

CONTENTS

Figures Index

Tables Index

ABSTRACT

This master dissertation presents an authoring tool to aid the creation of Augmented Reality applications that use photo-realistic rendering. The tool, named RPR-SORS Editor, uses a photo-realistic API (RPR-SORS) and focuses the creation and edition of virtual objects. The developed application is capable of handling the material parameters from the RPR-SORS API and the render engine used (OGRE) interactively, with an additional option to visualize the photo-realistic objects in Augmented Reality. The final result can be exported to another application that uses the photo-realistic API and render engine mentioned. A case study about an Augmented Reality application for architectural design was developed in order to evaluate the proposed tool.

Keywords: Augmented Reality, Photo-realism, Computer Graphics, Rendering, Materials, Authoring Tool.

RESUMO

Esta dissertação de mestrado apresenta uma ferramenta de autoria para auxiliar a criação de aplicações de Realidade Aumentada que fazem uso de renderização fotorrealística. A ferramenta, denominada *RPR-SORS Editor*, utiliza uma API de fotorrealismo (RPR-SORS) e tem como foco a criação e edição de objetos virtuais. A ferramenta desenvolvida é capaz de manipular de forma interativa os parâmetros de materiais do RPR-SORS e do motor de renderização utilizado (OGRE), com a opção de visualizar os objetos fotorrealistícos em Realidade Aumentada. O resultado final da edição pode ser exportado para uma outra aplicação que utilize a API de fotorrealismo e o motor de renderização citados. Um estudo de caso referente a uma aplicação de Realidade Aumentada para arquitetura foi realizado de forma a avaliar a ferramenta desenvolvida.

Palavras-chave: Realidade Aumentada, Fotorrealismo, Computação Gráfica, Renderização, Materiais, Ferramenta de Autoria.

1. Introduction

This chapter introduces briefly the problems approached by this dissertation, justifying the importance of a photo-realistic scene editor for Virtual and Augmented Reality applications.

1.1. Motivation

Virtual Reality (VR) and Augmented Reality (AR) applications use virtual rendered objects to create the sense of immersion for the user. Although there are some applications that require the objects to be highlighted (completely different from the real ones, e.g. training applications for AR), some situations require the objects to be most realistic as possible, to delude the user and create a better immersion rate.

The rendering of photo-realistic objects involves many calculations and techniques, such as illumination, occlusion and others. Executing all these calculations in real-time (requirement for AR) adds an extra challenge involving optimization as well as preserving the visual result.

In addition, the integration of these techniques generates a large amount of parameters to be set for every photo-realistic object in the scene. The manipulation of these parameters can become uneasy and, consequently, requires a tool to aid in this task. Such kind of tools is called Scene Editor, and unfortunately most of the applications that can handle Photo-Realism (PR) are proprietary solutions or undergoing projects.

1.2. Objectives

As a result of the observations of real world systems, it was concluded that there was a lack of tools that fulfill the goal of editing PR in real-time. This work proposes an application tool for the development of photo-realistic VR and AR solutions. The main goal of this application is to facilitate the creation of photo-realistic objects and scenes. In order to achieve this goal, it should be capable of integrating a wide range of techniques that compose a photo-realistic result, allow the manipulations of the objects and its parameters and execute in real-time. The advantage of this approach is the ability to show the final result combined (all the effects together with the surrounding environment) already in the editor screen, following the proposal of What You See Is What You Get (WYSIWYG). Another advantage is that the result of this editor could be explored by any application that needs to deal with photo-realistic objects and scenes.

1.3. ORGANIZATION OF THE DISSERTATION

This dissertation has been structured in order to introduce the concepts involved in the context of PR and present an implemented solution. The remainder of this document is organized as follows:

The second chapter defines some of the major topics and acronyms that are used throughout the dissertation. First it describes the characteristics of scene editors in general, and then it explains the concept of Photo-realistic rendering in real-time.

The third chapter describes the state of the art in PR. It begins with a basic definition of AR, and proceeds with many aspects that compose a photo-realistic result. It then describes the scene editors that are currently available and their purposes.

The fourth chapter brings a detailed solution of a scene editor integrated with photo-realistic techniques and explains how every parameter can be set up to achieve the best results.

The fifth chapter explains a case study performed using the proposed solutions as a tool to generate the photo-realistic objects.

The sixth chapter draws a conclusion for this dissertation, shows the contributions of the work, and outlines some interesting future works to it.

All references used to develop this work are shown in the seventh chapter.

2. Understanding some Concepts

This chapter will introduce some important concepts for better understanding of the final solution. The goal is to contextualize the reader to some definitions that will be used throughout the dissertation.

2.1. Scene Editors

During the implementation of an application prototype, it is common among the developers the attempt to create a configuration file that contains the data that is going to change frequently. The problems with this approach are the need to restart the application every time a parameter changes and the textual editing for all parameters. These troubles are even more evident when the user that is going to adjust these parameters is not computer savvy, and does not fully understand the meaning of the data that is being modified.

In order to overcome these problems, developers started to implement simple GUI elements into their solutions' viewports, allowing some or all of their algorithm parameters to be changed while the application was still running. This, however, is still impracticable when there is a large amount of parameters involved. Therefore, tools often called Material Editors or Scene Editors, depending on their scope, were conceived, indicating that they could manipulate as many parameters as a complete scene would require.

Material editors are applications that serve solely to edit the material properties of an object. The visualization of the material can be linked to an object; however the manipulation of the object (shape, size and position) is not mandatory for this kind of application. For instance, a material editor can use a default sphere to show all the materials that can be edited, or it can support the material applied on different meshes. In other words, mesh manipulation and object transformations are not the focus of a material editor.

On the other hand, scene editors have a wider purpose, which is to manipulate not only the materials, but also objects and scene parameters. These factors can generate a large amount of variations between scene editors, because some of them can be more specific to a context, while others can be more generic and implement global features, some of them will be never used by various users [1]. Depending on the tool implementation, objects can be fully editable or not. They can be moved, scaled and rotated, and their mesh can be remodeled changing the object shape. The illumination of the scene can also be manipulated, changing the types of lights that reach the objects, from point light sources to global environment illumination. Finally, a scene editor can also configure camera parameters. Camera properties, effects, and movement are examples of features usually present in such applications.

There are some features however, that do not necessarily apply to the main purpose of these applications but are relevant to facilitate the overall user experience, in both material and scene editors. For instance, the user must be able to save the current state of the application, and load previously saved states to resume the work. In addition, it is important that the user can import and export the scene elements from as many file formats as possible. This way, the editor becomes versatile and can be integrated with other applications.

The solution proposed in this work fits the scene editor category, implementing some features present on the majority of scene editors. At the same time, it adds some new features (i.e., photo-realistic rendering) in order to fulfill the purpose presented in this work.

2.2. RPR-SORS

In Computer Graphics (CG), many methods have been developed to increase the realism in synthetic images. In this context, the problem of PR is among the most researched topics [2]. This problem involves the synthesis, from the scene description input data, of a whole image that seems like a real photo. Lately, new and more specific problems arose in the context. One of them is the Photo-realistic Rendering of Synthetic Objects into Real Scenes (PR-SORS), which was firstly demanded by the cinematographic industry. This industry, since the last decades and until now, has the necessity of inserting seamless digital characters and objects into movies, e.g. dinosaurs from Jurassic Park and Gollum from The Lord of the Rings series, as shown in Figure 2-1. Since this area has received many development efforts, when watching the most recent movies, it can be noticed that good results were already achieved merging any kind of digital entities (characters, vehicles, environments, etc.) with real ones. However, the insertion of these elements is made in a post-processing step, after the real scene was recorded.

Following the same path, another problem that has come out is Real-Time Photo-realistic Rendering of Synthetic Objects into Real Scenes (RPR-SORS). This problem is relatively new and appears in the AR context [3]. Coarsely, RPR-SORS is just a problem of PR-SORS with some particularities. Since AR deals with interactive applications, the main difference between RPR-SORS and PR-SORS is that in the former everything must be executed in real-time.

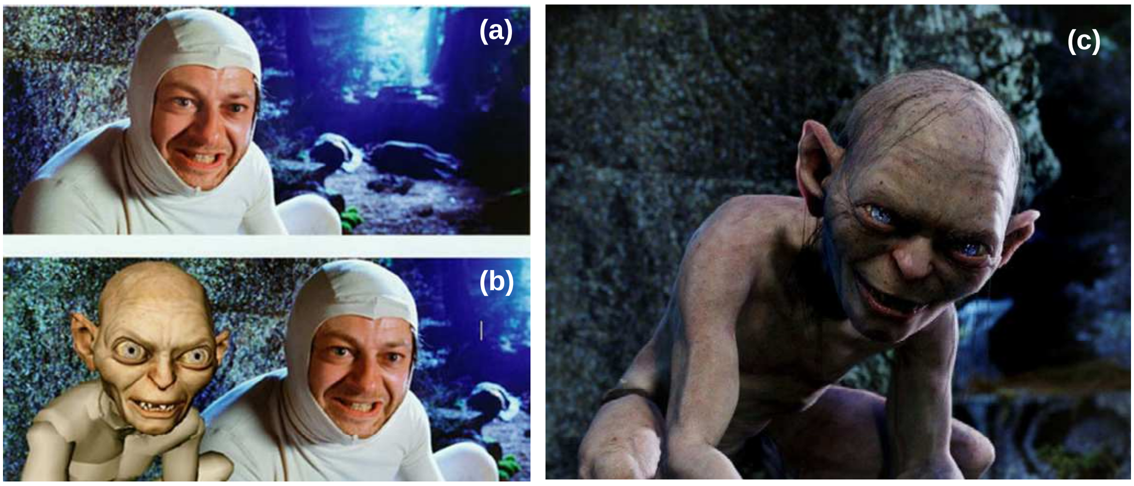


Figure 2-1. Example of PR-SORS application, illustrating the stages of capturing (a), post-processing (b) and final result (c).

The quality of the results in both, PR-SORS and RPR-SORS will be determined by the way three major topics are handled: shape, appearance and behavior of the virtual objects. If one of these topics is not properly dealt with, the virtual objects can appear highlighted, permitting users to easily notice that they are unreal.

The shape of a virtual object is an important visual cue because it can reveal when its proportions are not coherent with real objects measures. A common example for this case can be noticed in cartoons, where characters members' proportions are intentionally distorted to transmit a comic feeling.

The appearance of an object is determined by its albedo (material's reflectance) and by the amount of light reaching its surface. The albedo can either be modeled utilizing simple ways, such as using textures, or more sophisticated ones, using Bidirectional Reflectance Distribution Functions (BRDFs) [4]. The complexity of the function used does not necessarily indicate that the object will appear realistic, but it is linked to the type of material it should look like. For example, if a metal pan is modeled using a simpler function that supports only plastic materials, it will seem artificial.

Finally, the behavior of the virtual objects is directly related to objects movements and how they interact. The complexity of an object's behavior can vary drastically depending on its nature. Deformable objects tend to have more complex behaviors than rigid ones. Figure 2-2 shows an illustration about the difference between shape, appearance and behavior.

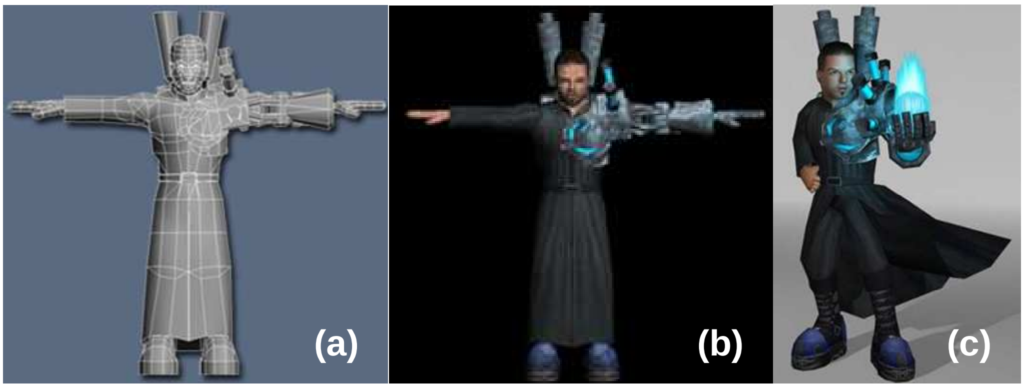


Figure 2-2. An example of a character and its shape (a), appearance (b) and behavior (c) [5].

In this work, the problem of RPR-SORS will be focused on the appearance. Even though the solution proposed could handle some aspects of shape and behavior of virtual objects, they are not detailed here.

3. STATE OF THE ART

This chapter details the problem of using PR in AR, explaining some techniques that are used to achieve the best results. Most of the concepts applied to AR are valid also to VR, furthermore the former requires a more strict real-time concern, since the real environment is being visualized simultaneously. This way, the rest of the dissertation will focus on an AR application. Basic concepts of material editors, and their role as authoring tools will also be described in this chapter.

3.1. PROBLEM

AR is defined as the research area that uses the computer to add virtual information or objects into a real environment [6]. An inherent feature from AR applications is the need for tracking and registry. In case any of these stages is not completed in real time, the application is not suitable for AR.

Although classic AR does not require information beyond the position and orientation of real world objects to be acquired from the environment, new tendencies have emerged along the publication of recent research articles. Nowadays, researchers also study ways to acquire real world information such as: the distribution of scene luminance, the geometric shape of the objects and the reflectance properties of their materials. Such studies are motivated by the fact that these new information are necessary when a seamless merge between the two worlds (virtual and real) is desired. A seamless merge refers to when the user of an application with these features could not perceive the difference between what is real and what is unreal (virtual).

It is important to highlight that such goal must be achieved treating exclusively the virtual scene rendering, without applying filters to the captured real scene to make it similar to the result obtained by the render, as shown in Fischer et al [7] and illustrated by Figure 3-1.

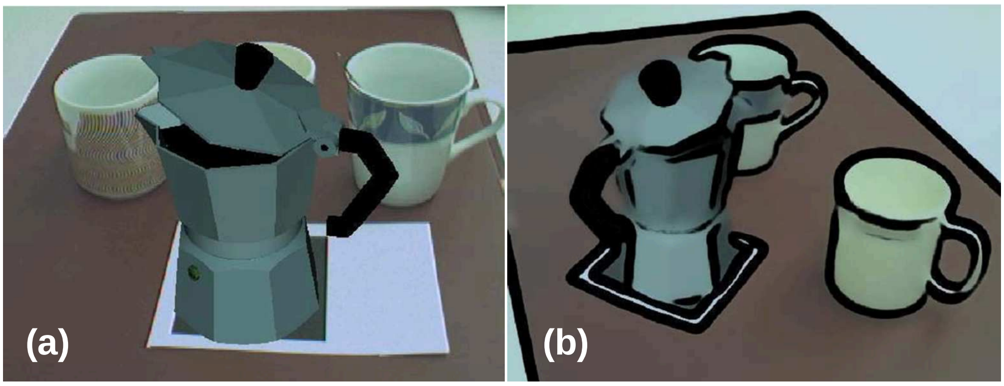


Figure 3-1. Example of filter to alter the original scene (a) and make it similar to the virtual objected rendered (b) [7].

Summarizing, the essence of the problem involving photo-realistic AR is how to proceed to accomplish a combination between the real and virtual worlds where the user perceives the scene as completely real.

3.2. BASIC CONCEPTS OF AR

A conventional AR system can be basically divided in two parts. The first part consists of capturing images and inferring information about the camera and scene structure parameters. This information is called registry [8] and is used to establish real world system coordinates and object positioning, in order to estimate the position and orientation of virtual objects to be inserted in the environment. Figure 3-2 shows the pipeline of AR.

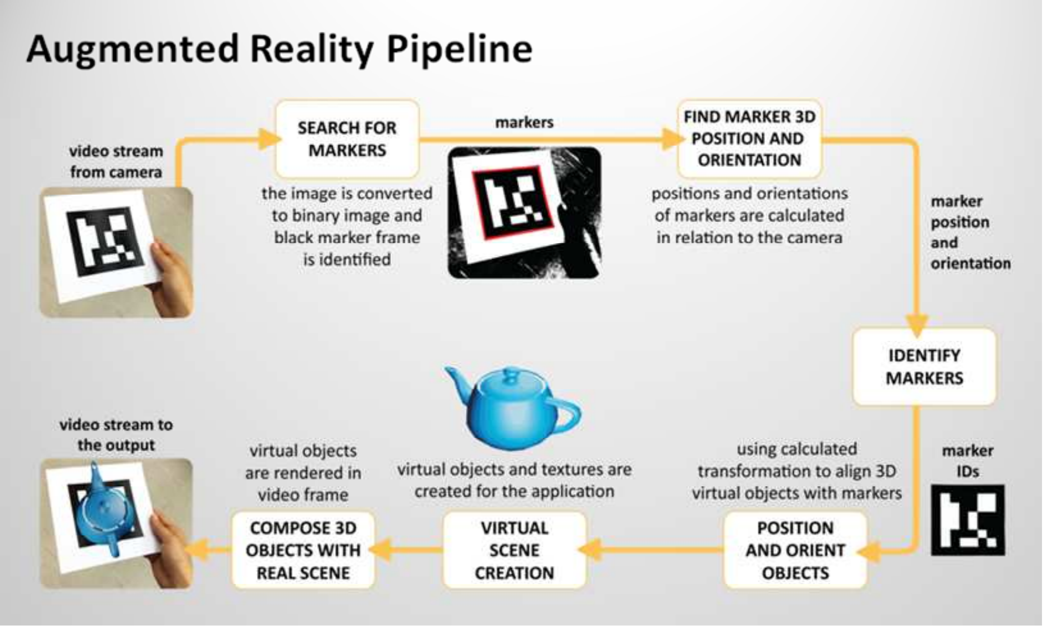

Augmented Reality Pipeline
video stream from camera
SEARCH FOR MARKERS
the image is converted to binary image and black marker frame is identified
markers
FIND MARKER 3D POSITION AND ORIENTATION
positions and orientations of markers are calculated in relation to the camera
marker position and orientation
IDENTIFY MARKERS
marker IDs
using calculated transformation to align 3D virtual objects with markers
POSITION AND ORIENT OBJECTS
virtual objects and textures are created for the application
VIRTUAL SCENE CREATION
virtual objects are rendered in video frame
COMPOSE 3D OBJECTS WITH REAL SCENE
video stream to the output

Figure 3-2. Augmented Reality pipeline.

The second part consists of the standard CG pipeline [9], which begins with the virtual model acquisition, followed by the visualization transformations, and finally rasterization. In AR context, the real objects present in the environment can influence the final result of this pipeline.

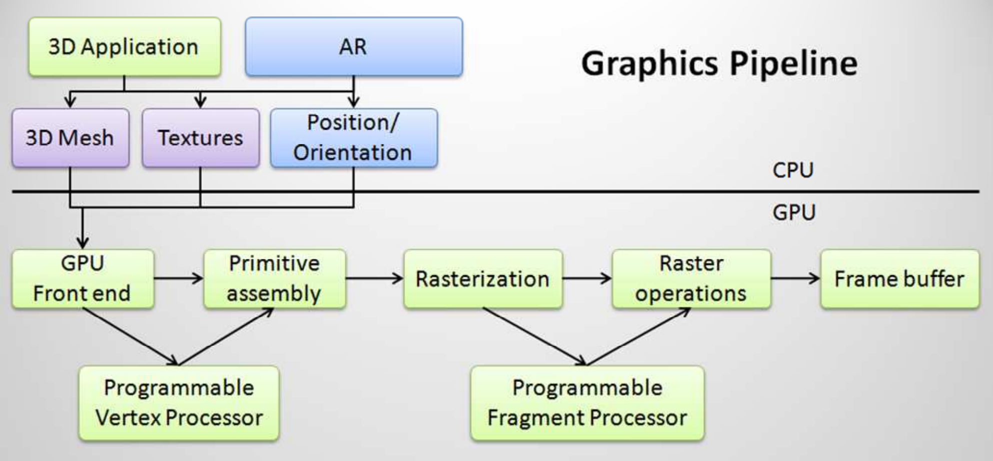


Figure 3-3. Computer Graphics pipeline.

Regarding the registry, there are various techniques developed to infer camera parameters. These techniques can be divided in two ample categories: marker-based and markerless tracking.

Marker-based tracking techniques are the ones which make use of any intrusive visual cue in order to calculate the position and orientation of real world objects. The marker itself can be any object that, when detected by the camera, provides tracking relevant information. Usually paper markers with a unique id are used [10], along with infrared lights, and retro-reflective spheres [11]. Figure 3-4 shows examples of these markers,

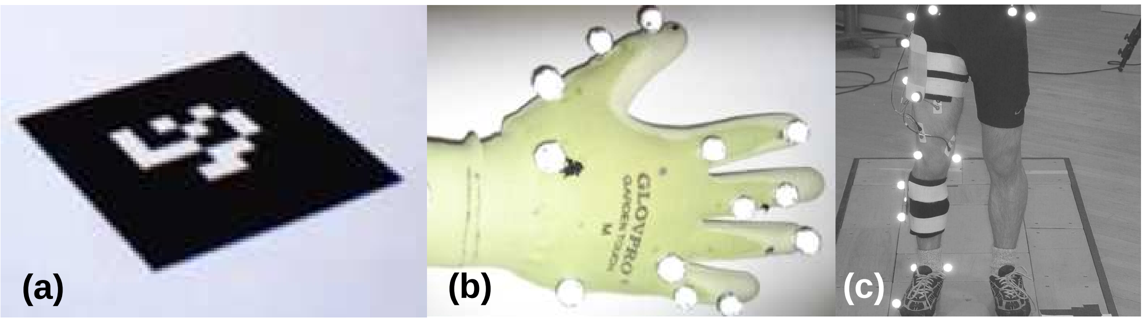


Figure 3-4. Different types of AR markers: (a) paper maker, (b) infrared lights and (c) retro-reflective spheres.

On the other hand, markerless techniques are based exclusively on the information already existent in the real world to calculate the registry [12]. No additional object is inserted into the real world, and the scene is maintained as original. Techniques used to accomplish this task include tracking the most relevant edges of the objects, as well as the optical flow or texture-based.

Regarding the object rendering, most recent techniques exploit the information about illumination and occlusion, in order to alter the final result of the virtual object, or even the real objects that interact with the virtual object. For instance, information about the real environment illumination can be inferred through a light probe [13]. Besides that, real objects can also be modified by the shadows from virtual objects [14].

This dissertation is focused on the object rendering; in other words, subjects related to the registry will be not described.

3.3. BASIC CONCEPTS OF PHOTO-REALISM

The research for new techniques that raises the sense of realism of computer-synthesized images has been one of the main goals of CG. However, even if realism is a highly desired feature in CG applications, it has been the center of controversy. One of the reasons for this discussion in the scientific community is the lack of well-defined standards for measuring realism. Along history, criteria based on human perception, physical precision or ad hoc methods have been used, but none of them is considered a formal standard.

Ferwerda, in his work [15], formally described three types of realism important to CG: physical realism, where the aspect and behavior of scene elements must be precise according to Physics laws; photo-realism, where the images must produce the same visual response as the scene, even if it is not physically coherent; and functional realism, where the main concern is to provide the user with useful information for a task execution. Physical realism requires the physical elements of the scene to be physically simulated, without focusing on the appearance. However, many times the correct simulation of the physical elements produces also a photo-realistic image. Figure 3-5 illustrates these three types of realism.

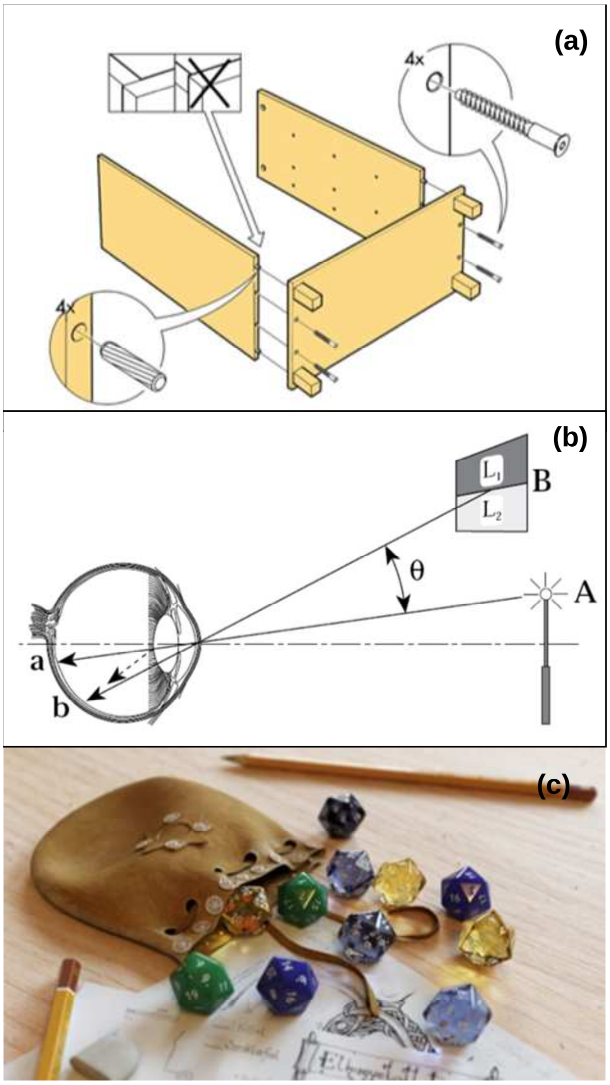


Figure 3-5. Examples of the three types of realism: (a) functional realism; (b) physical realism; (c) photo-realism.

There are various questions about the development of photo-realistic AR applications. Although these questions have predominant interest in Computer Science, concerns of human perception also have their relevance, as for instance, questions related to the user and tasks to be performed. The simple decision of using or not PR is the reason of many discussions in this community, however, the consensus is that even if visual realism cannot be directly linked to the productivity in augmented environments, it improves the sense of perception [16]. Studies show that certain realistic rendering effects, such as shadows, have great importance when the position of virtual objects must be determined [17]. However, it is important to highlight that researches about perception-based rendering showed that humans are less sensible to certain visual effects [18]. With this knowledge, it is clear the need for more detailed researches that evaluate the questions related to perception and the necessity of using or not realistic rendering effects in AR applications.

Taking into consideration the studies mentioned in this chapter, photo-realism is the most relevant variety of realism for photo-realistic AR. In this dissertation, the following aspects will be considered: material shading and reflectance, shadows, lens effects and composition, illustrated in Figure 3-6.

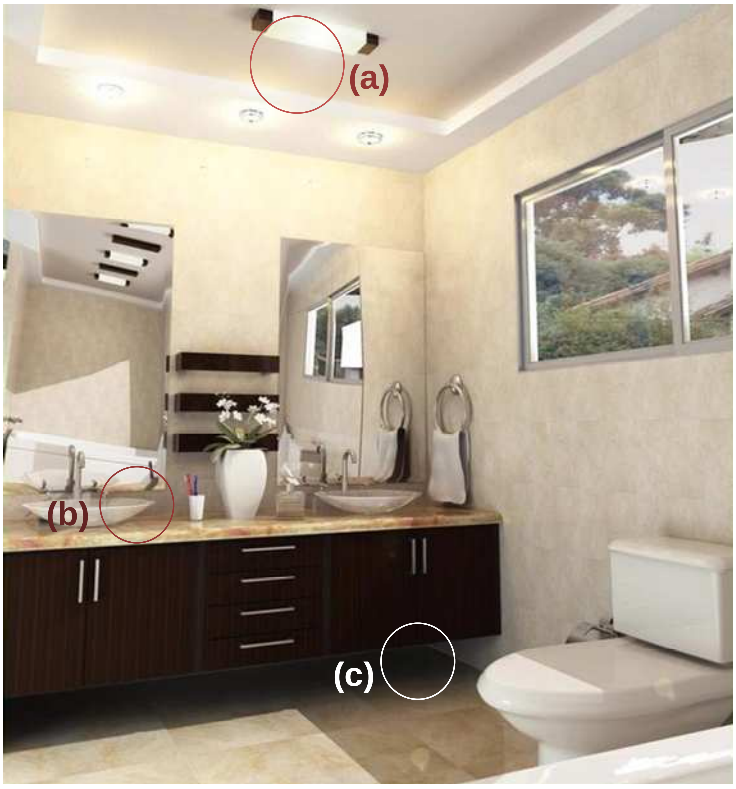


Figure 3-6. Photo-realistic rendering with three important aspects highlighted: (a) lens effects, (b) material shading and reflectance and (c) shadows.

3.4. EVOLUTION OF PHOTO-REALISM IN AR

3.4.1. Material Shading and Reflectance

The visual appearance of a given object is determined by two main factors: the incident light and its material reflectance properties. In CG, these two factors are usually dealt with simultaneously, defining the final visual result. Observing what happens when a light ray hits one given point on the surface of the object, it will be observed that part of the light is reflected immediately on the surface, another part penetrates the object and is re-emitted in another point and direction, and the remnant is absorbed. This behavior can be synthesized in a reflectance model [19], generally called Bidirectional Reflectance Distribution Function (BRDF), illustrated in Figure 3-7.

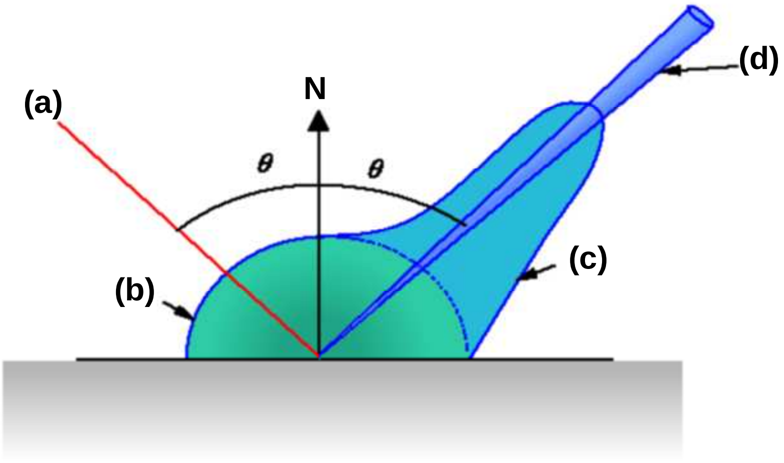


Figure 3-7. Bidirectional Reflectance Distribution Function illustration. (a) represents the incoming light; (b) completely diffuse component (albedo); (c) glossy factor, observed as an imperfect reflection; (d) completely specular component.

Over the years, many approaches have been used to model different types of material that can be found in the real world. These approaches can be divided in two main categories: local and global illumination models [20].

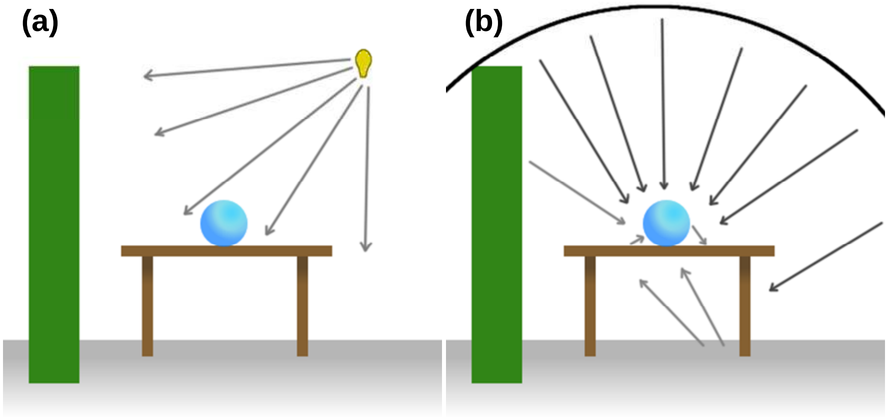


Figure 3-8. Light from local illumination (a) comes from discrete spots, while light from global illumination (b) is distributed through the environment.

Local illumination models deal with point light sources and, in general, produce non-realistic visual results. The reason that there are many algorithms that handle this specific case is their simplicity in relation to global models, because integrals are not necessary to calculate the resulting shading. Among local illumination models, there are many studies that aim the improvement of the BRDF model in order to represent more complex materials [20]. Between the existent models, Lafortune et al. [21], based on previous researches, achieved an elegant and representative solution, capable of representing a wide variety of materials, such as anisotropic, retro-reflective, etc.

The disadvantage of synthetic BRDF based models is the incapability for representing materials that vary the reflectance properties along the surface of the object, which occurs frequently in real cases. In order to solve this problem, Dana et al. [22] introduced the concept of Bidirectional Texture Functions (BTF), where an apparent BRDF is stored in each texel of an image, as seen in Figure 3-9. The capture of these data requires a device called gonioreflectometer [23], which captures also information about self-shadowing and interreflections. This approach, even if used with local illumination, produces realistic visual results. Such textures, however, are large and need heavy pre-processing in order to be used in real-time. Sattler et al. [24], Müller et al. [25] and Schneider [26] proposed approaches based in Principal Component Analysis (PCA) to compress the BTF data. Meseth et al. [27] made a brief evaluation about existing BTF compressing methods. The conclusion is that even if the use of BTF can achieve high quality and realistic images, there are at least two main limitations to the technique application: high memory requirements and the need for texture sampling using the gonioreflectometer.

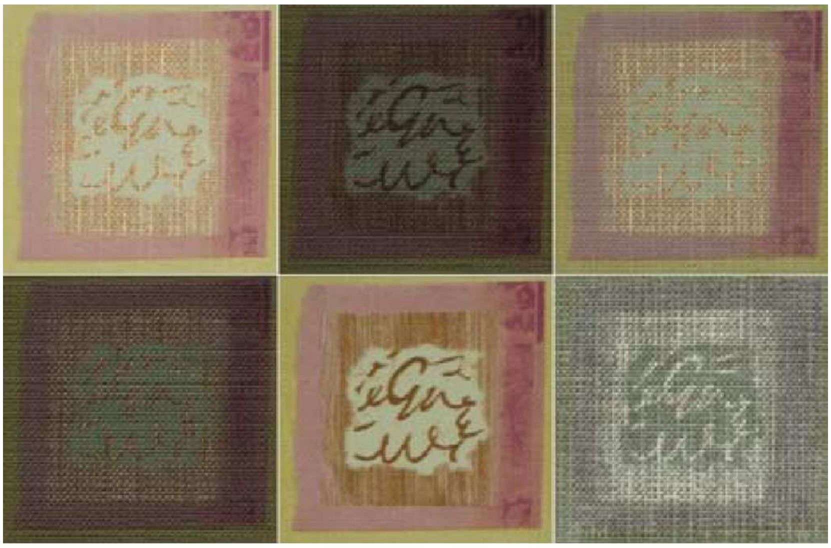


Figure 3-9. Six views of one wallpaper from different views and light directions, as stored in a BTF [27].

On the other hand, global illumination models are capable to represent with higher fidelity real world light sources, where the light does not come from a single point, due to the various reflections in the environment. Image Based Lighting (IBL) is an example of one technique able to approximate the global illumination effect through the use of HDR environment maps (see Figure 3-10). Ramarmoorthi and Hanrahan [28] showed one efficient representation for environment irradiance maps (also called diffuse environment maps) using spherical harmonics. Trying to improve this method performance, King [29] ported such representation to the GPU, using a sampling scheme dependent of the environment map format, and that requires a balancing procedure during the spherical harmonics transformation. Another problem of this approach is the lack of flexibility for representing more complex materials, since the only products of this technique are the original environment map (completely specular) and the diffuse environment map generated. That way, only material completely diffuse (e.g., plaster, chalk) and material completely reflective (e.g., mirror) can be accurately reproduced. Other materials can be only approximated from a combination of these two maps.

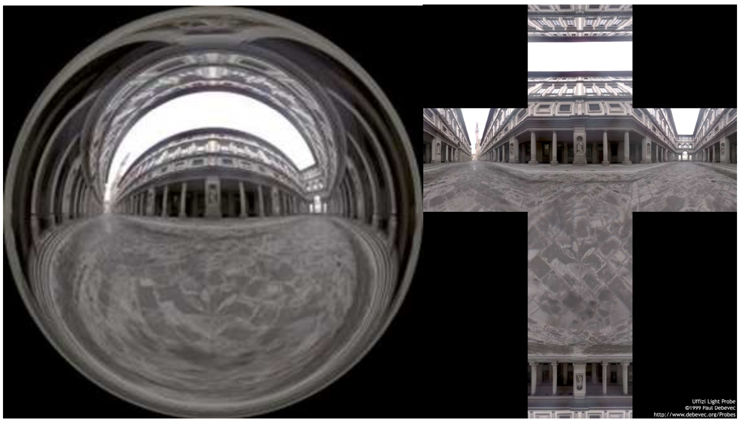


Figure 3-10. Light probe environment maps used in IBL.

McAllister et al. [30] combined the BTF level of detail, the simplicity of Lafortune's analytic BRDF, and the concepts of global illumination generating representative and realistic visual results, shown in Figure 3-11. Their work consisted in pre-process a BTF in order to generate parameters compatible to Lafortune's BRDF. In order to solve the problems of the environment maps, McAllister proposed that some maps need to be generated with different filter levels, beyond the diffuse map (corresponding to the highest level). This way, it is possible to reproduce materials with a wide range of roughness levels. This solution appeared adequate to photo-realistic AR, for it combines the power of the techniques mentioned in this chapter, with the optimization necessary to execute them in real-time.

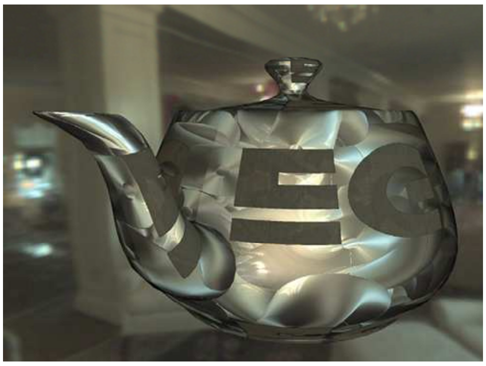


Figure 3-11. Results obtained by McAllister combining Lafortune's BRDF with a texture map [30].

3.4.2. Shadows

Shadows have a fundamental role in the perception of the object position in a given scene. It's through shadows that, for instance, the observer can estimate how high the object is in relation to the ground, as seen in Figure 3-12. This is one of the most studied subjects within CG, consequently resulted in a massive number of proposed methods. Since global illumination algorithms are more suitable for RPR-SORS (as mentioned previously), it is natural that photo-realistic shadow algorithms are also based in such concept.

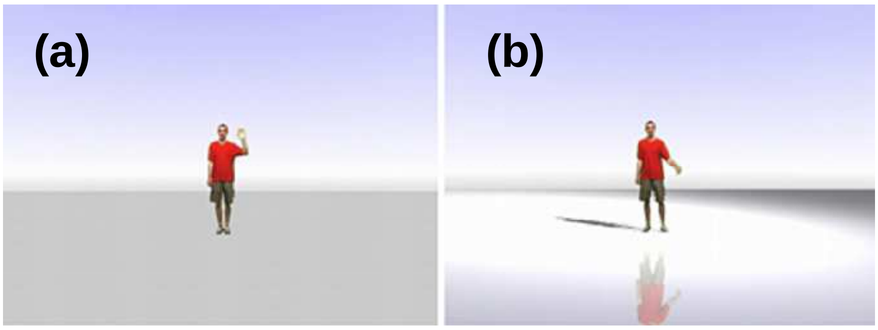


Figure 3-12. Importance of shadows to determine the height of an object. It is not possible to determine if the person in (a) is touching the ground, while in (b) the shadows provide the necessary cue.

Lately, due to the heavy growth of GPUs computation capacity, the self-shadowing generation method known as Ambient Occlusion [31] has increased in popularity among computer games and similar real-time applications. Figure 3-13 shows an example of a model with an Ambient Occlusion technique being executed. The main idea of this method relies in approximating the occlusion of a given point due to the neighbor points. There are many ways to compute that approximation, each one with particular advantages and drawbacks. Bunnell in [32] proposed a method based on object vertices, converting the vertices into surfels and computing the occlusion between them. Even if this method reduced the computational cost from $O(n^2)$ to $O(n \times \log n)$, the performance of this technique was very sensitive to the amount of objects in the scene, which makes it unsuitable for geometrically complex scenes. Besides, since the calculation is performed by vertex, high level tessellated meshes are need for good results.

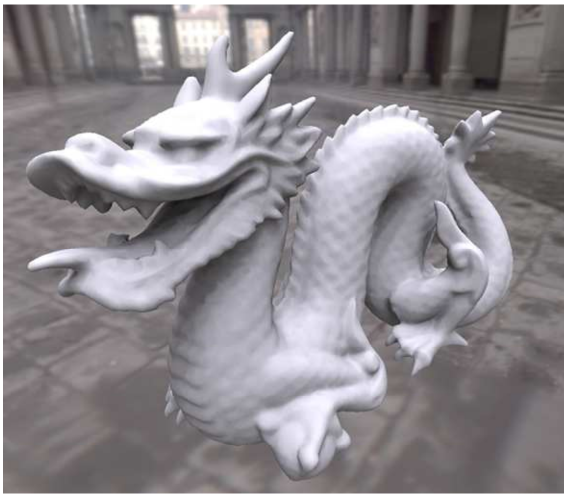


Figure 3-13. Ambient Occlusion technique result.

As interesting alternatives to Bunnell's proposal are the techniques that estimate the occlusion in screen space. The real advantage of such methods related to the one proposed by Bunnell are three features: the small penalty when the number of objects increase; the ease of integration with the graphic pipeline because it does not deal directly with the objects mesh; and the fact that animated objects are trivially dealt with. Screen space methods have also a stable performance because the computational cost is usually constant throughout execution of the application, since they depend only upon the screen resolution. Shanmugam and Arikan [33] approximated the total occlusion for each pixel in screen space sampling its neighbors and calculating their contribution. However, due to low quality primitives used in the GPU, the obtained results showed undesirable artifacts. Such artifacts appear as darker strokes over the edges, as seen in Figure 3-14.

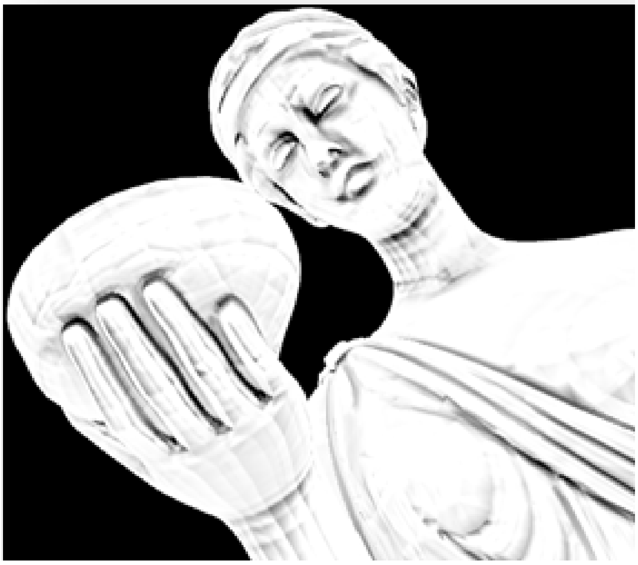


Figure 3-14. Ambient Occlusion dark strokes issue when a low-resolution mesh is used.

Bavoil et al. in [34] also approximated the screen space ambient occlusion; however, the calculation is done based on each pixel horizon. The main advantage of this last technique is the addition of an angle bias, which caused a considerable reduction of the dark artifacts that might occur in low-resolution meshes.

Besides Ambient Occlusion techniques that consider the light uniformly scattered through the environment, there are techniques that use an environment map as light source. Zhou et al. in [35] proposed a method to generate shadows from environment maps in dynamic scenes, however, the only object animations supported were rotation, translation and scaling. They based their work in shadow fields' concept in order to deal with the occluders and the light emitters. The main advantage of this algorithm consists of the generation of shadows from arbitrary shaped light sources, including environment maps. The main issue in this method is related to the shadow fields, because the memory usage and the time spent to generate them are too high. Tamura et al. in [36] developed an adaptive generation of shadow fields, which reduced the problems mentioned before.

Tamura et al. also proposed a method for shadow generation based on environment maps [37]. Their approach consisted of: first render the scene taking in consideration only self-shadows, identify which regions are occluded, and finally remove the light energy erroneously added, generating thus shadows. The amount of light radiation to be removed is computed by rendering one image where each light source is segmented from an environment map. Finally, this work also dealt with the particular case where shadow must be added in specular surfaces.

3.4.3. Lens Effects

Lens Effects are visual artifacts generated together with the virtual object to be rendered, aiming a better integration with the real world, or simply to improve the scene realism. Such effects are necessary because the digital object is free from any physical interference existing in the real world, while the light information captured by the camera lens or by human eyes may have many deviations along its trajectory [38], [39].

Kawase [40] implemented many of the effects that occur in real cameras, e.g., Glare, Bloom and Exposure Control, in real-time; however, he does not apply them to AR. Another characteristic in his solution is that it does not use a physically correct approach, (in spite of Spencer et al. [38] and Kakimoto et al. [39]) aiming exclusively the visual result. In addition, one effect that is not present in Kawase's implementation is the common noise generated in the camera charge-coupled device (CCD), especially low quality ones [41].

Luksch [42] also tried to produce such effects in real-time, using OGRE graphics engine [43]. His results are substandard to Kawase's, both in visual quality and variety of effects. In their work, both authors use a Tone Mapping algorithm to simulate the scene exposure control. Lucksh uses Reinhard's algorithm [44], while Kawase does not specify which algorithm is used in his solution.

3.4.4. Composition

Besides the previously mentioned effects, another main factor to the quality of the final result is the method used to merge the virtual and real worlds. This process is called composition and consists of mixing the image obtained from the real world (camera stream) with the image containing the virtual object rendering. The most naive way to create this composition is overlaying the real image with the virtual image. Even if this technique seems overly simple, it has been used as default in many AR applications. Due to such simplicity, the changes caused by the insertion of the virtual object into the real world are not considered in the process. The ideal method should take into consideration the most notorious changes perceived by the observer, such as shadows, occlusion and color bleeding, involving the two worlds.

Haller et al. [45] proposed a method using the GPU to perform the composition; however, they could only calculate occlusion and shadows. This method needs the local scene to be previously modeled and the phantom objects to be associated with their respective real objects. Despite it works efficiently, this method has some downbeat features: duplicated shadow generation when the virtual objects enter into the real objects shadow, and the uniform darkening is not enough to deal with specular reflections in the real scene. Figure 3-15 shows two examples of the mentioned effect.

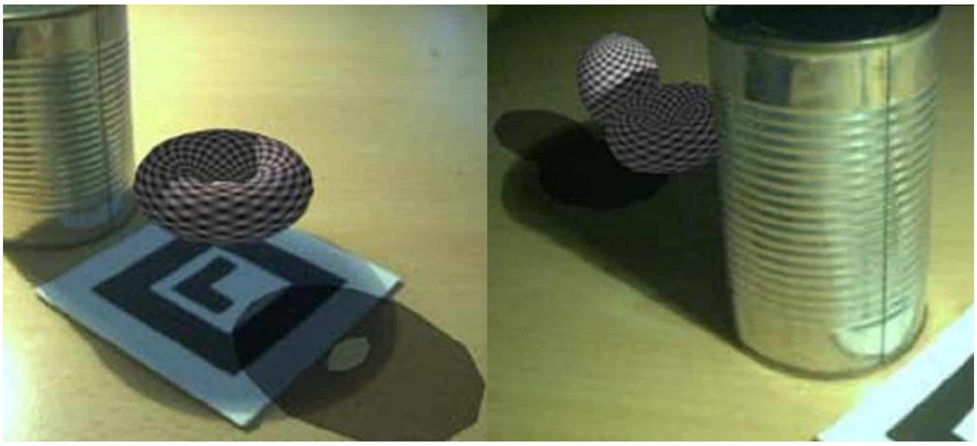


Figure 3-15. Incorrect shadow generation, when the virtual object enters the real object shadow [45].

In [14], Debevec proposed a more general and sophisticated method. Besides occlusion and shadows, he can deal with other effects, such as caustics, indirect lighting and specular reflections. This method is based on the rendering of two images, so that the difference between them can be evaluated. At the end, this difference is added to the real world image, composing the final image. The first rendered image contains the local scene with the virtual objects placed correctly, and the second image contains only the local scene. The resulting image (generated by the difference between the two images) stores the lighting changes caused by the virtual objects. Although the original method proposed by Debevec has an elevated computing cost and is initially proposed to non-real-time applications, Gibson et al. [46] modified it to be used in AR applications.

3.5. Scene Editors

Each one of the techniques mentioned early in this chapter requires their own parameters to be adjusted for the best results. Summing with common variables in the usual rendering pipeline, such as diffuse, specular and ambient colors; as well as textures and animation parameters, they become an uneasy set of parameters to be tuned simultaneously.

Such need generated a demand for sophisticated scene editors that could aid the manipulation of all the parameters. During the last years, commercial applications –such as 3D Studio Max [47] and Maya [48] – and open-source applications – such as Blender [49] – have dominated the market of commercial solutions. However, this chapter will focus specifically on Material and Scene Editors which use OGRE, detailing applications that try to fulfill a restrict demand. One problem with such editors is that they are often a result of an individual effort, or even a small group of developers, and frequently do not achieve the final level of development containing all the desirable features.

Ogre Material Editor [50] for example is a material editor, developed using wxWidgets library, which is cross-platform, and features a text editor for material scripts and shader programs. The limitation of this project is the use of a text editor to manipulate the parameters, similar to edit directly the text file. There are no visual aids, with one exception for the code completion and highlight. Trying to focus on the material properties, oFusion [51] is a plugin developed for 3D Studio Max that creates an Ogre render window to present the final results. This plugin has support for mostly every parameter from Ogre materials, but it relies on the power of 3D Studio Max to edit the scene objects.

Ogitor Scene Builder [52], on the counterpart, is a stand-alone application designed with Qt [53], which theoretically makes the project multi-platform. The solution is an undergoing project, but features important attributes. For example, it supports various mesh formats and has a plugin-based architecture, which makes possible to add physics and sound to the application without changing the scene editor.

A missing feature in every scene editor pointed out by this dissertation is the support for advanced photo-realistic materials. The best results they can achieve are the same results Ogre library can provide. In order to overcome this drawback, this dissertation focuses on a scene editor that provides an integrated solution for photo-realistic rendering, as explained in the next chapter.

4. PROPOSED SOLUTION: RPR-SORS EDITOR

This chapter will describe the RPR-SORS Editor (Real-time Photo-realistic Rendering of Synthetic Objects into Real Scenes) and the solution it brings in the development process of an AR application. Figure 4-1 illustrated the revised pipeline, highlighting the steps tackled by the proposed solution. The RPR-SORS project aims integrating synthetic objects into real scenes in an imperceptible way and in real-time. In order to achieve this, a framework was implemented in order to facilitate the development of applications with such requirements [54]. The main functionalities of this framework are: RPR-SORS API and RPR-SORS Editor.

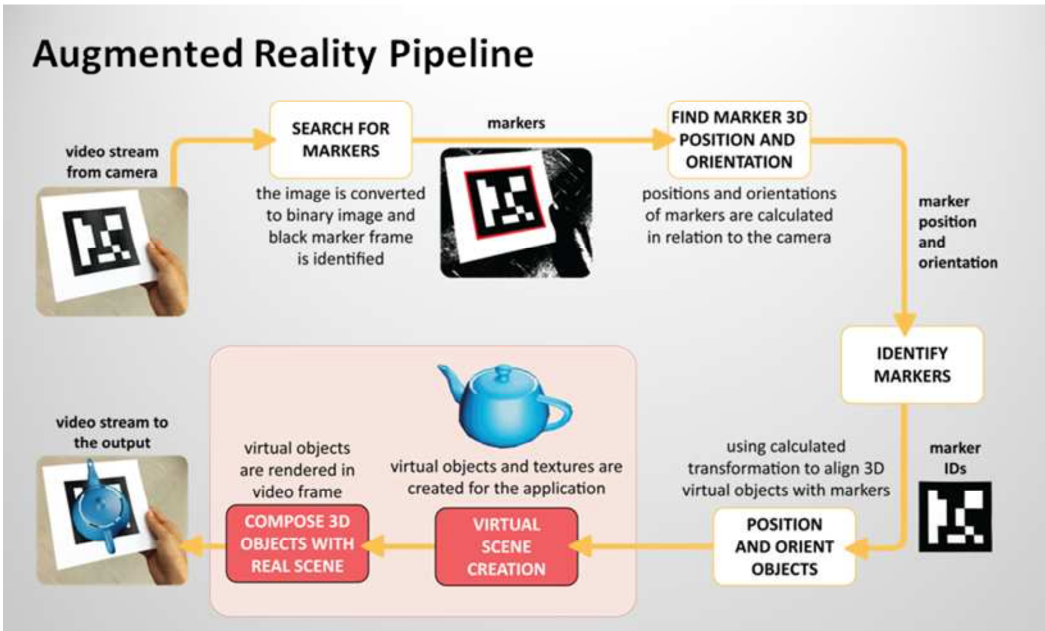

Augmented Reality Pipeline
video stream from camera
SEARCH FOR MARKERS
the image is converted to binary image and black marker frame is identified
markers
FIND MARKER 3D POSITION AND ORIENTATION
positions and orientations of markers are calculated in relation to the camera
marker position and orientation
IDENTIFY MARKERS
marker IDs
using calculated transformation to align 3D virtual objects with markers
POSITION AND ORIENT OBJECTS
virtual objects and textures are created for the application
VIRTUAL SCENE CREATION
virtual objects are rendered in video frame
COMPOSE 3D OBJECTS WITH REAL SCENE
video stream to the output

Figure 4-1. Revised Augmented Reality pipeline.

The RPR-SORS API is a collection of techniques accommodated in a novel pipeline with the objective of providing real-time photo-realistic rendering support. This work is described in details in [55]. It was developed within OGRE's framework, an open-source graphics engine. This framework was chosen because it is versatile and powerful, supporting both OpenGL and DirectX libraries and various shader languages. In addition, its material scripts architecture made possible to integrate the photo-realistic effects with reduced efforts. Applications that use this API are free to choose any tracking library for AR, since the API has no biddings to any specific library.

The development pipeline of an AR application (see Figure 4-2) includes a step of programming, where the application will deal with rendering and interaction. It also has a step of authoring, where the developer designs the virtual scene that will be inserted in the Augmented Reality environment. In this step, it is often required a large amount of time to craft all the elements of the application, and additionally, if the virtual objects are complex and realistic, the creation process is extended even longer. In order to overcome this bottleneck, this dissertation introduces the RPR-SORS Editor, to act as an authoring tool for AR applications that require photo-realistic rendering, making straightforward the creation of complex materials for AR environments.

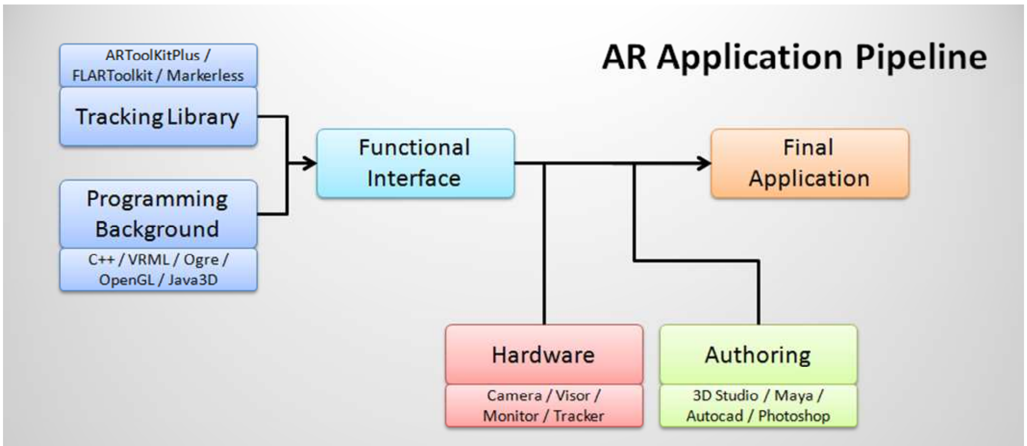

AR Application Pipeline
ARToolKitPlus / FLARToolkit / Markerless
Tracking Library
Programming Background
C++ / VRML / Ogre / OpenGL / Java3D
Functional Interface
Final Application
Hardware
Camera / Visor / Monitor / Tracker
Authoring
3D Studio / Maya / Autocad / Photoshop

Figure 4-2. Most relevant steps of an AR Application pipeline.

The proposed editor integrates the RPR-SORS API with ARToolKitPlus [56] in order to provide real-time editing of material parameters in AR. It was written in C++ and uses the OGRE library for rendering the scene and wxWidgets [57] library for the GUI. It also supports the most relevant parameters from the regular graphics pipeline (e.g. texture modes, shading modes, culling, shading parameters, etc.) and the new parameters added from the RPR-SORS architecture. Figure 4-3 shows a usage example of the RPR-SORS framework. The product of the RPR-SORS Editor can be easily consumed by an application that uses the RPR-SORS API.

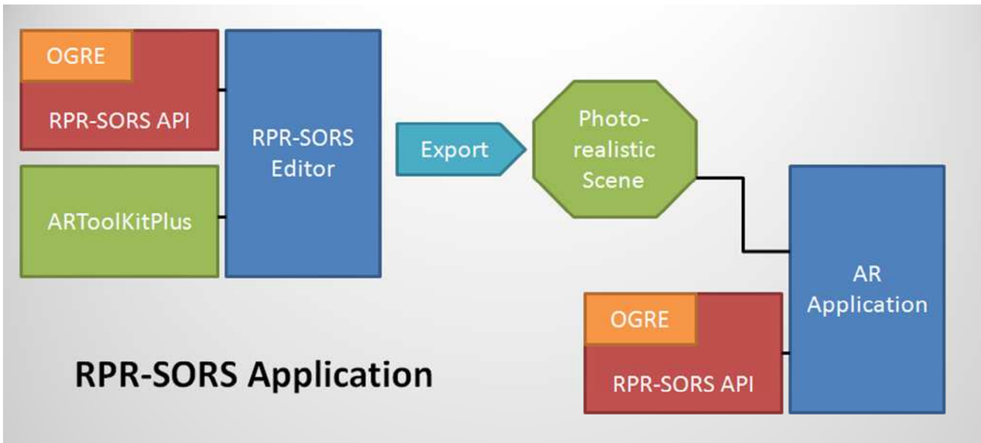
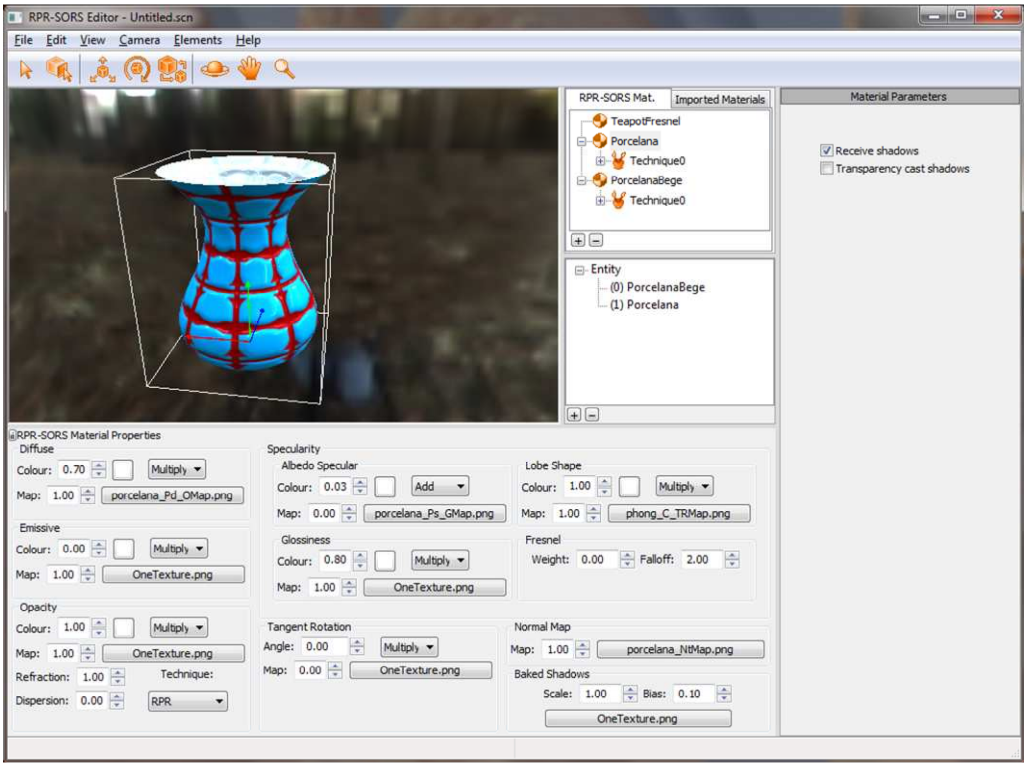


Figure 4-3. Example of an application of the RPR-SORS framework.

The rest of this chapter is concerning to illustrate the main features of the RPR-SORS Editor. Figure 4-4 shows the main window of the proposed solution.



☑ Receive shadows
□ Transparency cast shadows

RPR-SORS Material Properties

Diffuse
Colour: 0.70 Multiply
Map: 1.00 porcelana_Pd_OMap.png

Emissive
Colour: 0.00 Multiply
Map: 1.00 OneTexture.png

Opacity
Colour: 1.00 Multiply
Map: 1.00 OneTexture.png
Refraction: 1.00 Technique: RPR
Dispersion: 0.00

Specularity
Albedo Specular
Colour: 0.03 Add
Map: 0.00 porcelana_Ps_GMap.png
Lobe Shape
Colour: 1.00 Multiply
Map: 1.00 phong_C_TRMap.png
Glossiness
Colour: 0.80 Multiply
Map: 1.00 OneTexture.png
Fresnel
Weight: 0.00 Falloff: 2.00

Tangent Rotation
Angle: 0.00 Multiply
Map: 0.00 OneTexture.png

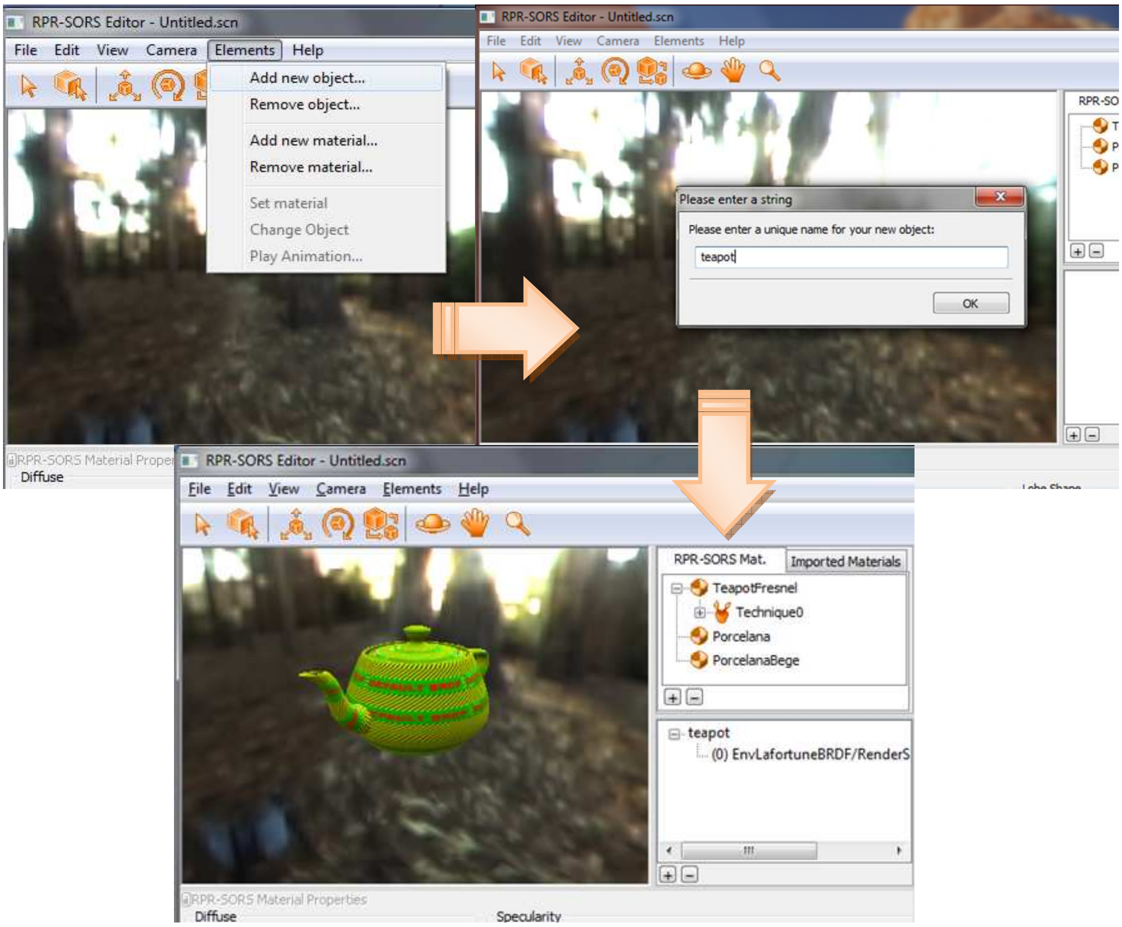
Normal Map
Map: 1.00 porcelana_NtMap.png
Baked Shadows
Scale: 1.00 Bias: 0.10
OneTexture.png

Figure 4-4. RPR-SORS Editor main window.

4.1. BASIC FEATURES

As a scene editor, the RPR-SORS Editor provides support for some basic features. The interface was designed to deal with two main categories that are commonly used by users: Materials and Objects. The following operations can be performed with Objects:

- **Add:** A new object can be created by the user. The interface will query about the desired mesh file (OGRE Mesh) in the system file and a unique name, as seen in Figure 4-5. The object will be created in the world origin $(0,0,0)$ and be associated with a default material.



Please enter a unique name for your new object: teapot

Figure 4-5. Editor interface and result for the add object command.

- **Remove:** The user can remove an object from the list and all the references to that object will be deleted and every resource allocated by the object creation will be freed. The interface for object removal can be seen in Figure 4-6.

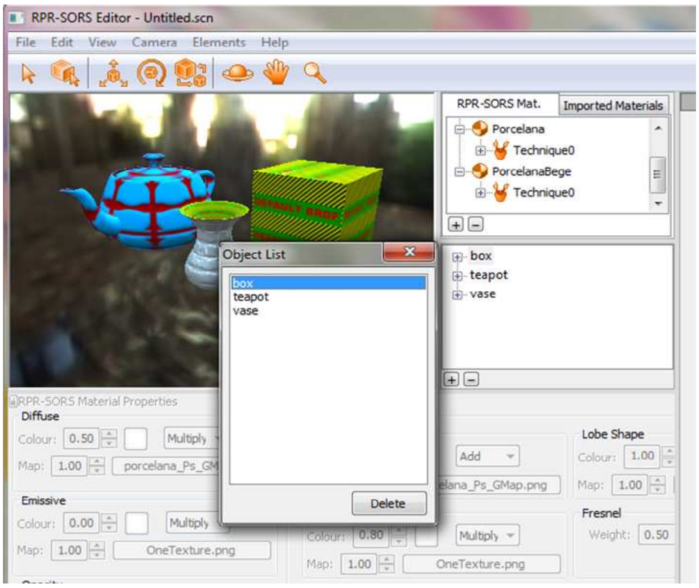


Figure 4-6. Remove object window, showing the list of objects that can be deleted.

- **Translate, Rotate and Scale:** Objects can be transformed through the use of the corresponding tools. Translation and Rotation can be performed separately on each axis (X, Y, Z), but scale is performed uniformly on the three axes. The scale operation is one simple example of how the RPR-SORS Editor deals with the object shape.
- **Select:** The user can visually select an object or a sub-mesh with the mouse pointer. Sub-mesh selection is used to change the materials of selected area. Figure 4-7 illustrates the selection tools mentioned in this topic.

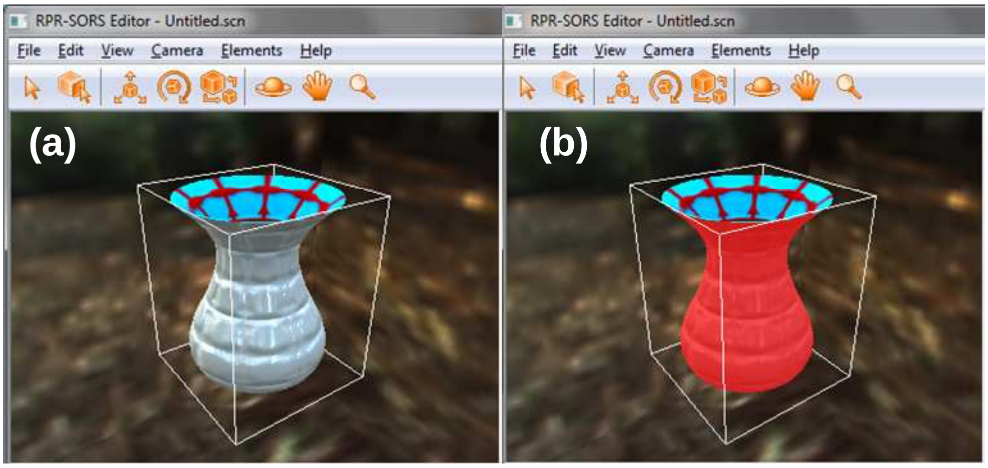
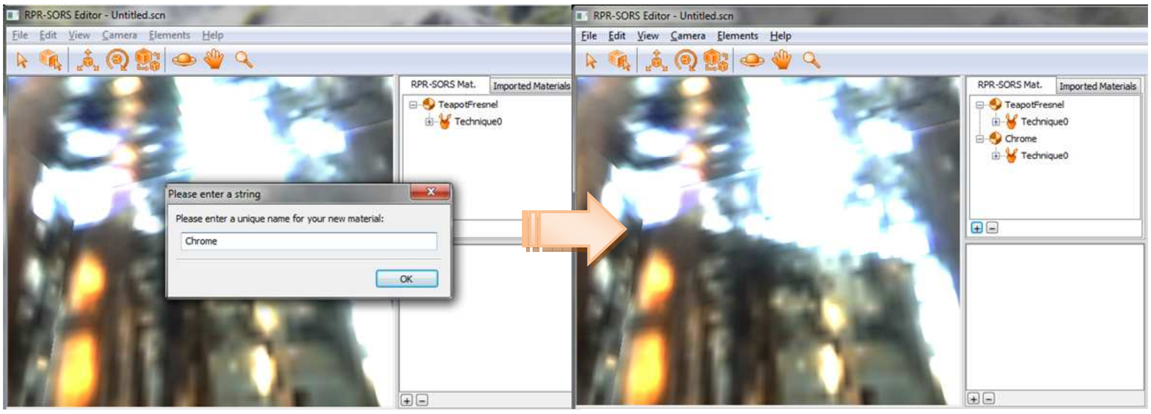

RPR-SORS Editor - Untitled.scn
File Edit View Camera Elements Help
(a)
RPR-SORS Editor - Untitled.scn
File Edit View Camera Elements Help
(b)

Figure 4-7. Selection tool (a) and sub-mesh selection tool (b) usage examples.

- **Change Mesh:** The user can change the mesh of an object without having to redo all the operations with the new object. Every transformation done in the old mesh will not be lost. In addition, the materials from the old object will be transported to the new object, if possible, following the sub-meshes index order.
- **Play Animation:** Objects can be embedded with animations. The RPR-SORS Scene Editor supports bone animations exported along with the mesh. It can list and play them (simultaneously, if needed). The purpose of this feature is to enable the developer to have a glance of the object final state, even if it will be dynamic. The animation control pane is one simple example of how the RPR-SORS Editor deals with the object behavior.

Most of the operations that can be performed with objects can be applied to materials. However, in order to illustrate all the features of the solution, they will be described individually. The following operations can be performed with Materials:

- **Add:** The user can create a new material and the editor will initialize it with default parameters and textures. Figure 4-8 shows an illustration presenting the addition of a new material.



Please enter a string

Please enter a unique name for your new material:

Chrome

Figure 4-8. Editor interface and result for the add material command.

- **Remove:** The user can remove a material from the list and all the references to that material will be deleted and every resource allocated by the material creation will be freed
- **Assign to object:** A material can be assigned to an object sub-mesh, if it is not already being used by another object, as seen in Figure 4-9. In the case of a material already being used, the user must first free the material, changing the old object material, and then assign to the new one. When a new object is created by the editor, a default material is set to all its sub-entities, avoiding any conflict.

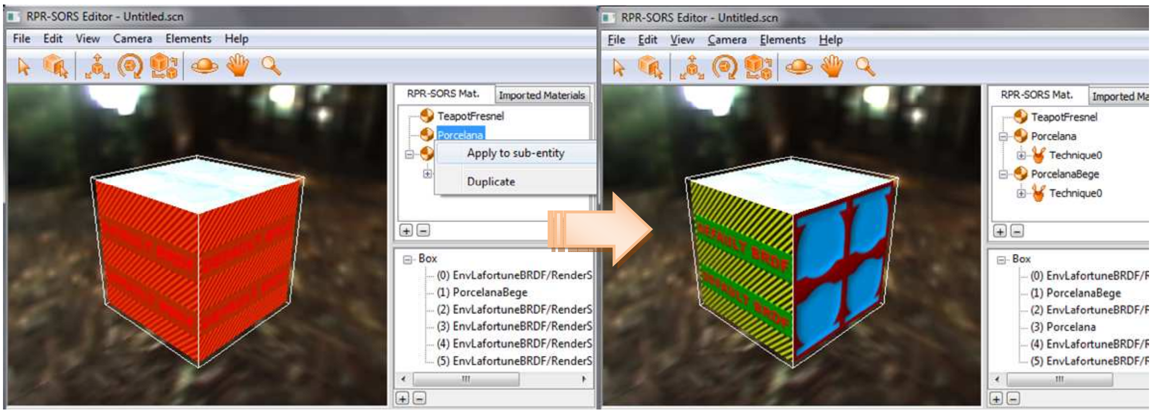


Figure 4-9. Material being assigned to an object sub-mesh.

- **Duplicate:** The user can duplicate an existing material, which will create an exact copy with all parameters and textures. This action can be used to overcome the limitation of one sub-mesh per material. The Editor has this limitation due to the generation of the environment map per entity, as implemented in [55]. Using the duplicate function, the user can rapidly create a new material based on a previous entry and assign it to the new object.

- **Change property:** When the user selects a material, technique, pass or texture unit (OGRE elements that define a material), it is possible to change the RPR-SORS parameters and OGRE parameters for that specific unit. A detailed explanation of the possible parameters will be described in chapters 4.2 and 4.3.

In addition to object and material edition, the user can save the current state of the scene for later use. The RPR-SORS Editor uses a custom file format (scn), which is a zip file, containing a scene description XML file, meshes and textures resources. The XML file stores the camera, objects, materials and scene properties. This way, the saved file is self-contained and can be moved to any computer that runs the application and restore all the configurations to resume work.

Also, it is possible to export the scene for use in external applications. The RPR-SORS Editor can save mesh, material, shader program and the scene parameter files, which are the core of the object appearance. This way, the composed scene can be used in a final application with the same visual quality as presented in the editor.

4.2. Integration with OGRE

OGRE is the RPR-SORS Editor and API main engine. In fact, all the photo-realistic techniques from the RPR-SORS framework were implemented using OGRE's resources, such as material scripts, shader loader, compositor chains and texture buffers. The RPR-SORS Editor is, essentially, a new proposal of an OGRE scene editor; furthermore some additional parameters were added in order to compose the photo-realistic results. Currently it supports only OGRE mesh files as input objects. If the user needs to use another type of mesh (obj, 3ds), it must be previously exported to an OGRE mesh using another tool, such as the LEXIExporter [58]. It is important to specify that the version of OGRE used in this dissertation was 1.6.

In addition, the RPR-SORS Editor supports full manipulation of OGRE materials parameters [59], visible when the user selects one material, technique, pass or texture unit. However, in the RPR-SORS pipeline some parameters do not have any visual effect in the final result, so they were removed from the interface in order to maintain the editor window clean.

The material parameters that remained were `receive_shadows` and `transparency_casts_shadows`. And the material parameter that was removed from the interface was `lod_distances`. The parameter `lod_distances` specifies the distance where a given technique will be used. The technique has only one parameter in OGRE (`lod_index`). It specifies what technique will be executed depending on the distance to the viewer, for performance purposes. However, regardless of the distance, the proposed pipeline requires the same amount of calculations to create the effects, so these parameters available for material and technique were removed in the RPR-SORS Editor.

The pass parameters that remained were:

Table 4-1. Remaining pass parameters.

scene_blend	depth_check	depth_write	depth_func
depth_bias	alpha_rejection	cull_hardware	cull_software
shading	polygon_mode		

And the pass parameters that were removed from the interface were:

Table 4-2. Removed pass parameters.

ambient	diffuse	specular	emissive
lighting	fog_override	colour_write	max_lights
start_light	iteration	point_size	point_sprites
point_size_attenuation	point_size_min	point_size_max	

The parameters `ambient`, `diffuse`, `specular` and `emissive` were removed because the values for these variables come from the shader program, and consequently configured by the RPR-SORS pane (see chapter 4.3). The parameter `lighting`, also has no effect when a shader program is used. The implementation of fog, through the `fog_override` parameter was removed from the interface because it was considered not relevant for photo-realistic AR applications. The `colour_write` parameter was removed because the only pass that the user will be able to manipulate is the final one, where the object is rendered in the screen. Therefore a parameter that disables the object rendering was not desirable for the proposed solution. The parameters `max_lights`, `start_light` and `iteration` are valid only when point light sources are used. In this dissertation, a global illumination technique was implemented, therefore these parameters were removed. In addition, the parameters `point_size`, `point_sprites`, `point_size_min`, point_size_attenuation and point_size_max are used when mesh vertices are rendered in the screen. This is not the case for this dissertation solution.

The texture unit parameters that remained were:

Table 4-3. Remaining texture unit parameters.

texture	anim_texture	tex_coord_set	tex_address_mode
tex_border_colour	filtering	max_anisotropy	mipmap_bias
scroll	rotate	scale	wave_xform
scroll_anim	rotate_anim		

And the texture unit parameters that were removed from the interface were:

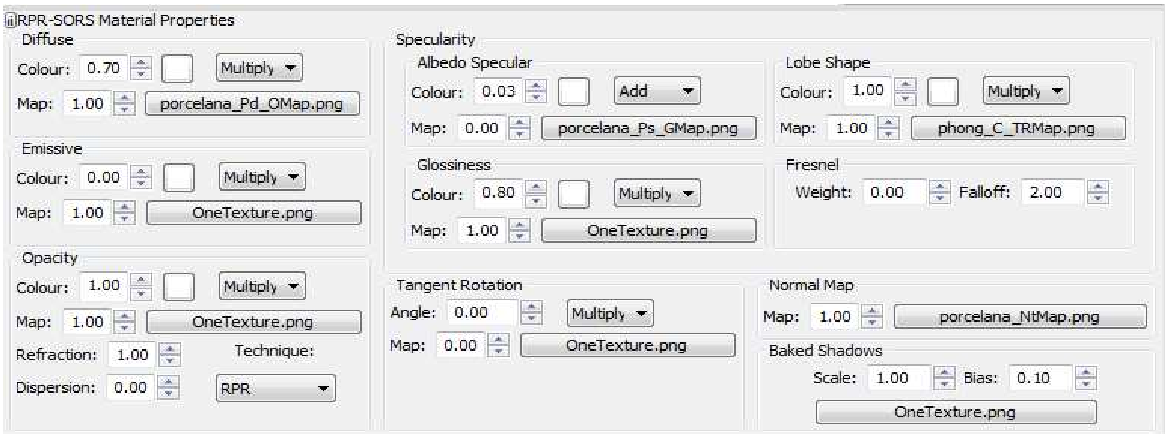
Table 4-4. Removed texture unit parameters.

texture_alias	cubic_texture	colour_op	colour_op_ex
colour_op_multipass_fallback	alpha_op_ex	env_map	transform
binding_type	content_type		

The parameter texture_alias was removed from the interface because it is mainly used when the material is to be inherited by another one. In the RPR-SORS Editor, the materials the user manipulates are already final stages, without need for inheritance. The parameter cubic_texture was removed since the textures applied to objects are simple 2D or 3D textures. For this same reason, the parameter env_map was also removed. The parameters colour_op, colour_op_ex, colour_op_multipass_fallback and alpha_op_ex are used to operate two different texture units. However, in this solution, the texture units used are independent from each other, thus these parameters were removed from the interface. The parameter transform is redundant, grouping the scroll, rotate and scale parameters in a $4x4$ matrix. It was then removed to simplify the interface. Both binding_type and content_type parameters configure behaviors already dealt with by the RPR-SORS API shader program.

4.3. Photo-realistic Material Parameters

In addition to the regular parameters from OGRE, mentioned in the previous chapter, the scene editor proposes additional parameters in order to accomplish the desired photo-realistic effect. These parameters are directly connected to the RPR-SORS API and the techniques implemented on it. Such parameters are associated with the object appearance, thus they were implemented within the material scripts that are applied to the objects. Objects materials are independent from any other object, so the parameters for each material must be set individually. Additionally, because of API limitation, one material cannot be used by more than one object. Then, if two objects share the same material, it must be duplicated with a different name and applied to the second object. Figure 4-10 shows the panel where new parameters can be configured, called RPR-SORS Material Properties. Since the parameters described in this chapter are novel, they will be explained in details.



RPR-SORS Material Properties

Diffuse
Colour: 0.70 Multiply
Map: 1.00 porcelana_Pd_OMap.png

Emissive
Colour: 0.00 Multiply
Map: 1.00 OneTexture.png

Opacity
Colour: 1.00 Multiply
Map: 1.00 OneTexture.png
Refraction: 1.00 Technique:
Dispersion: 0.00 RPR

Specularity

Albedo Specular
Colour: 0.03 Add
Map: 0.00 porcelana_Ps_GMap.png

Glossiness
Colour: 0.80 Multiply
Map: 1.00 OneTexture.png

Lobe Shape
Colour: 1.00 Multiply
Map: 1.00 phong_C_TRMap.png

Fresnel
Weight: 0.00 Falloff: 2.00

Tangent Rotation
Angle: 0.00 Multiply
Map: 0.00 OneTexture.png

Normal Map
Map: 1.00 porcelana_NtMap.png

Baked Shadows
Scale: 1.00 Bias: 0.10
OneTexture.png

Figure 4-10. RPR-SORS Material Parameters pane, containing new parameters for materials.

- **Diffuse**:

When the object surface is not polished or specular enough, the light that reflects on the object is scattered almost uniformly in all directions. This effect results in the objects diffuse color (albedo). However, for many objects in the real world, the diffuse properties (as well as many other properties) can change along the object surface. This way, the scene editor provides the following options to configure this effect:

  - **Diffuse color**: single RGB color that will be used uniformly on the object surface.
  - **Diffuse color weight**: float number $[0, \infty)$ that will multiply the diffuse color. This weight can be used to diminish the color intensity or increase it beyond RGB bounds, simulating an HDR color.
  - **Diffuse Map**: texture image that can be imported from many formats (jpg, bmp, png, etc.). This texture simulates an irregular object surface, where the diffuse color changes along the surface.

- **Diffuse Map weight**: float number $[0, \infty)$ that will multiply the diffuse map texture.
- **Diffuse operation**: This parameter is a function which will be used to compose the two types of diffuse input (color and texture). It can perform two operations: add and multiply.

Figure 4-11 shows an example of the parameters usage.

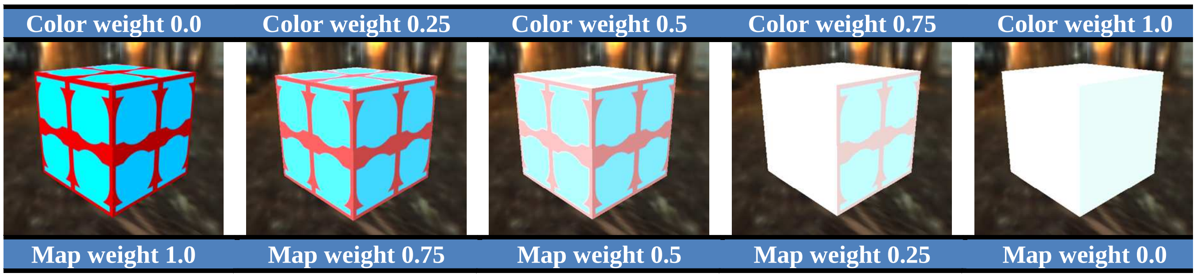


Figure 4-11. Different diffuse values used in a box.

- **Emissive**:

There are some objects that instead of reflecting incoming light, they emit their own light, such as light bulbs, fluorescent objects, etc.:

- **Emissive color**: single RGB color that will be used uniformly on the object surface.
- **Emissive color weight**: float number $[0, \infty)$ that will multiply the emissive color.
- **Emissive Map**: texture image that can be imported from many formats (jpg, bmp, png, etc.). This texture simulates an irregular object surface, where the emissive color changes alongside the surface.
- **Emissive Map weight**: float number $[0, \infty)$ that will multiply the emissive map texture.
- **Emissive operation**: This parameter is a function which will be used to compose the two types of emissive input (color and texture). It can perform two operations: add and multiply.

Figure 4-12 shows an example of the parameters usage.

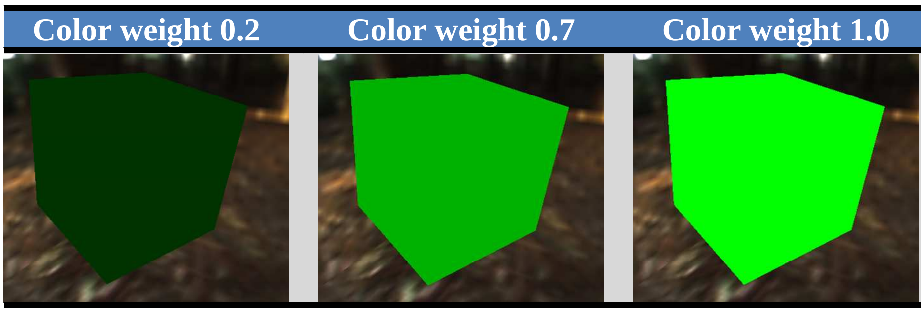


Figure 4-12. Different values of emissive color weight.

- **Specular**:

Depending on the object surface roughness, the light that reaches a certain point on the surface can be reflected to the opposite direction, forming the same angle with the normal as the incident vector. This factor composes the specular color of the object, which can exhibit the color of the environment surrounding the object, similarly to a mirror. The amount of light that is reflected usually is indirectly proportional to the amount of light that is scattered (diffuse component). However, even the light that is reflected to the opposing direction can be slightly scattered. That is the difference between a perfect mirror (which reflects the incoming light perfectly to the opposing direction in relation to the normal) and a glossy object (which reflects the incoming light irregularly). These effects can be simulated by the scene editor through these parameters:

  - **Specular color**: single RGB color that will be used uniformly on the object surface. The input color is an analogy to what color is reflected by the object. For example, a mirror reflects all components of the incoming light equally, but a golden object reflects more the yellow light component, and less other components. This applies to most metallic objects and their respective colors.
  - **Specular color weight**: float number $[0, \infty)$ that will multiply the specular color. This weight can be used to diminish the color intensity or increase it beyond RGB bounds, simulating an HDR color.
  - **Specular Map**: texture image that can be imported from many formats (jpg, bmp, png, etc.). This texture simulates an irregular object surface, where the specular color changes along the surface.
  - **Specular Map weight**: float number $[0, \infty)$ that will multiply the specular map texture.

- **Specular operation**: This parameter is a function which will be used to compose the two types of specular entries (color and texture). It can perform two operations: add and multiply.

Figure 4-13 shows an example of the parameters usage.

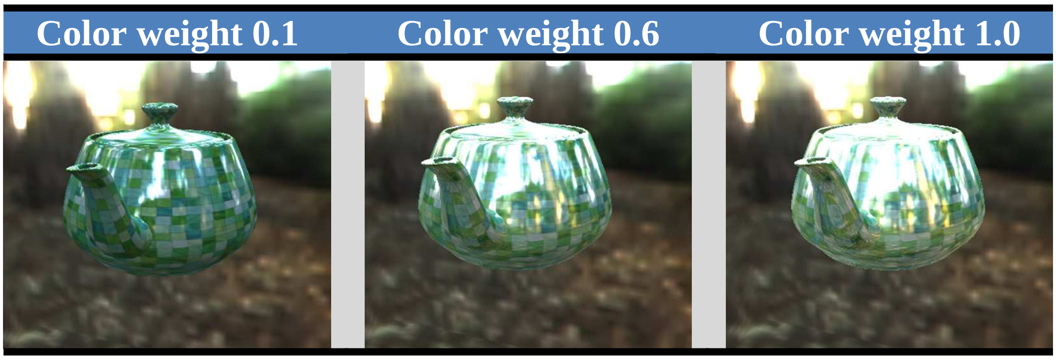


Figure 4-13. Different specular values being applied to the teapot material.

The specular component is slightly more complex than other material components, therefore it requires more functions to define the final specular appearance, such as glossiness, lobe shape and Fresnel.

As mentioned before, a reflective object can reflect the incoming light perfectly as a mirror, or scattered through varying directions. If the object reflects the incoming light imperfectly, the user perceives the reflection as blurred or glossy. The level of glossiness defines how perfect the reflection will be. As the glossy level moves toward zero, the object reflection will appear more like a diffuse object, which in fact reflects the incoming light scattered in all directions. The glossiness component is defined by the following parameters:

- **Glossiness color**: single RGB color that will be used uniformly on the object surface. Values near 1.0 (one) will produce perfectly specular objects and values near 0.0 (zero) will produce diffuse objects.
- **Glossiness color weight**: float number $[0,\infty)$ that will multiply the glossiness color.
- **Glossiness Map**: texture image that can be imported from many formats (jpg, bmp, png, etc.). This texture simulates an irregular object surface, where the glossy level changes along the surface.
- **Glossiness Map weight**: float number $[0,\infty)$ that will multiply the glossiness map texture.

- **Glossiness operation**: This parameter is a function which will be used to compose the two types of glossiness entries (color and texture). It can perform two operations: add and multiply.

Figure 4-14 shows an example of the parameters usage.

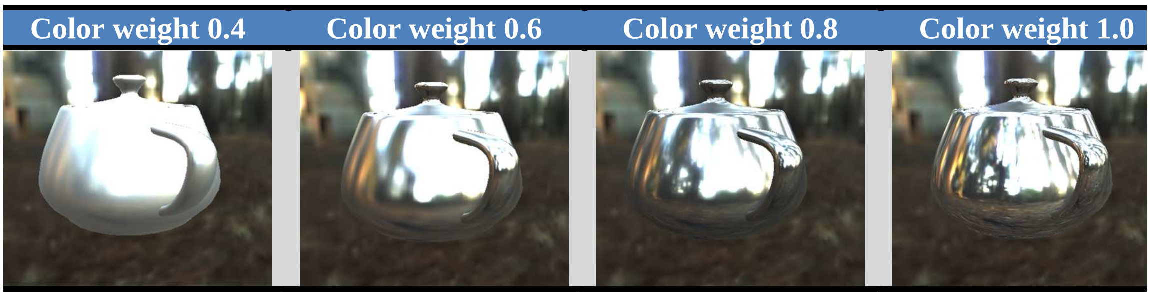


Figure 4-14. Different glossiness levels applied to a teapot material.

When the light reflects on the object surface, it is scattered in many directions. The mathematical abstraction that fits the scattered light directions is similar to a lobe, and its shape is directly related to the object appearance. Modifying the lobe shape allows the creation of unique materials, such as retro-reflective materials and materials with Fresnel effect. In this solution, three numbers, representing the three components of the reflection vector, define the lobe shape. Since the values of color and textures are limited in the range $[0, 1]$, these values were mapped to $[-1,1]$.

- **Lobe shape color**: single RGB color that will be used uniformly on the object surface.
- **Lobe shape color weight**: float number $[0, \infty)$ that will multiply the lobe shape color component.
- **Lobe shape Map**: texture image that can be imported from many formats (jpg, bmp, png, etc.). This texture simulates an irregular object surface, where the lobe shape changes along the surface.
- **Lobe shape Map weight**: float number $[0, \infty)$ that will multiply the lobe shape map texture.
- **Lobe shape operation**: This parameter is a function which will be used to compose the two types of lobe shape entries (color and texture). It can perform two operations: add and multiply.

One of the characteristics of many objects is the increasing reflection at grazing angles. That means when the object is observed directly, forming an angle of 90 degrees between the viewer and the surface tangent, the diffuse component (i.e. the object color) is more visible. When the user observes the same object towards its horizon angle (near 180 degrees with the surface tangent), the object appears to reflect the environment with greater intensity. This effect is called Fresnel effect and it's modeled by two parameters:

  - **Fresnel weight**: float number $[0, 1]$ that determines the strength of the Fresnel effect. Values near 1.0 (one) mean the object is a perfect mirror when observed near 180 degrees angles.
  - **Fresnel falloff ratio**: float number $[0, \infty)$ that determines if the Fresnel function will decline faster or slower according to the observer angle.

- **Opacity**:

Objects can be transparent in some degree and to some color components. In addition to the level of opacity, transparent objects can refract the light through its extension.

  - **Opacity color**: single RGB color that will be used uniformly on the object surface. Values near 1.0 (one) means that the object is completely opaque to that color, e.g. a green glass can be modeled by a $(0.0, 0.3, 0.0)$ color component.
  - **Opacity color weight**: float number $[0, \infty)$ that will multiply the opacity color component.
  - **Opacity Map**: texture image that can be imported from many formats (jpg, bmp, png, etc.). This texture simulates an irregular object surface, where the opacity changes along the surface.
  - **Opacity Map weight**: float number $[0, \infty)$ that will multiply the opacity map texture.
  - **Opacity operation**: This parameter is a function which will be used to compose the two types of opacity entries (color and texture). It can perform two operations: add and multiply.
  - **Opacity refraction weight**: float number $[0, \infty)$ that determines the refraction index of the object relative to the air.
  - **Opacity dispersion weight**: float number $[0, \infty)$ that is used to scatter the incoming light components, simulating a prism.

- **Tangent Rotation**:

Some objects have aligned micro-fissures on their surface, which causes the light to be reflected more in one direction than in the other one. These objects are called anisotropic. Usually, 3D meshes are exported with a tangent vector associated with each vertex. This information is then used as a basis and can be modified by the following parameters:

- **Tangent Rotation angle**: float number $[0, 360]$ that will be used to rotate the basis tangent vector.
- **Tangent Rotation map**: texture image that can be imported from many formats (jpg, bmp, png, etc.). This texture simulates an irregular object surface, where the tangent vector changes along the surface. Since texture values are limited to $[0, 1]$, these values are then mapped into $[0, 360]$ angle values.
- **Tangent Rotation operation**: This parameter is a function which will be used to compose the two types of tangent rotation entries (color and texture). It can perform two operations: add and multiply.

- **Normal map**:

Normal maps are well-established CG textures that allow high frequency surface details without the need for complex meshes. The texture contains the values of each normal vector, which are mapped to the object surface.

- **Normal map weight**: float number $[0, \infty)$ that will multiply the normal map texture.
- **Normal map**: texture image that can be imported from many formats (jpg, bmp, png, etc.).

Figure 4-15 shows an example of the parameters usage.

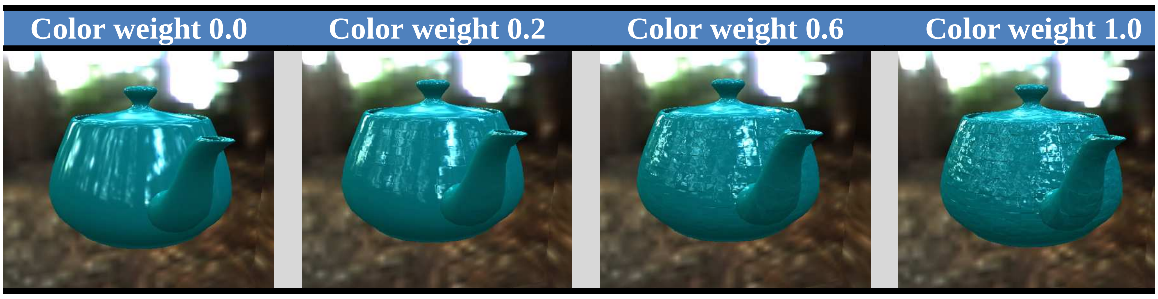


Figure 4-15. Different normal map values applied to the teapot material.

- **Baked shadows**:

When the object and the environment illumination are static, the shadows calculation can be optimized by a pre-computation step. The result of this step is a shadow texture, which is mapped to the object and contains information about self-occlusion. This technique is often used in computer games, since they require a large amount of simultaneous events, and every optimization is welcome.

- **Baked shadows scale**: float number $[0, \infty)$ that will multiply the baked shadows texture uniformly.
- **Baked shadows bias**: float number $[0, \infty)$ that will be added to the result of the baked shadows texture.
- **Baked shadows map**: texture image that can be imported from many formats (jpg, bmp, png, etc.).

Figure 4-16 shows an example of the parameters usage.

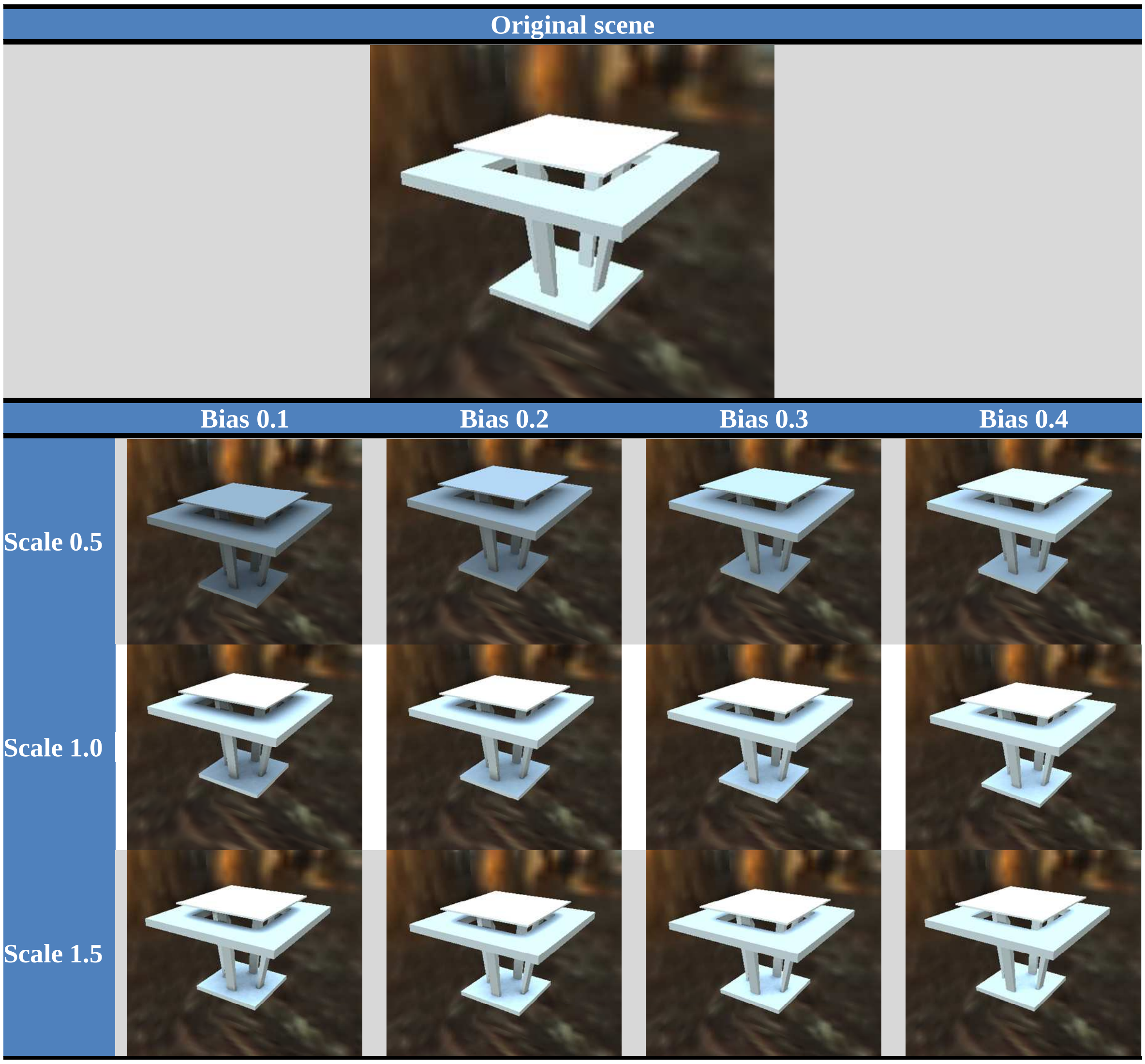
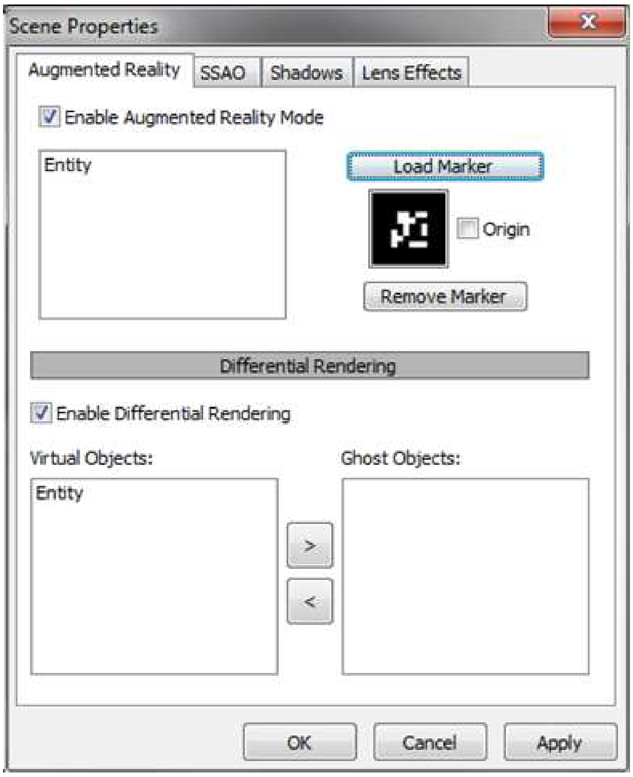


Figure 4-16. Different baked shadows values applied to an object.

4.4. Integration with AR

One important feature of the scene editor proposed in this work is the adaptation of the AR pipeline, in order to support the photo-realistic effects. This feature allows the user to see the final result of the application, while the modeled objects interact with the real world. The ARToolKitPlus [56] framework is used in this case, but the system architecture is modular enough to allow an easy replacement of libraries. ARToolKitPlus uses markers to detect the world orientation and object positioning, but this could be easily changed for a markerless library (although, for the time being, markerless libraries are processing consuming) [12].

The RPR-SORS Editor can configure the AR setting through the Scene Properties window. When activated, it automatically detects the computer webcam, and allocates the necessary resources. Afterwards, the user can manage the objects and markers, associating and disassociating ids as necessary. The scene editor loads the marker ids from the ARToolKitPlus resources directory. This feature allows the user to see the actual marker picture being associated with the desired object, which is an advantage since commonly what happens is that the user holds a printed version of the marker and does not necessarily know which id it is belong to. Furthermore, the user can choose as world origin one of the markers that will be used as reference to the other ones. The AR configuration pane is shown in Figure 4-17.



Scene Properties

☑ Enable Augmented Reality Mode

Entity

☐ Origin

☑ Enable Differential Rendering

Virtual Objects:

Entity

Ghost Objects:

Figure 4-17. Augmented Reality and Differential Rendering pane.

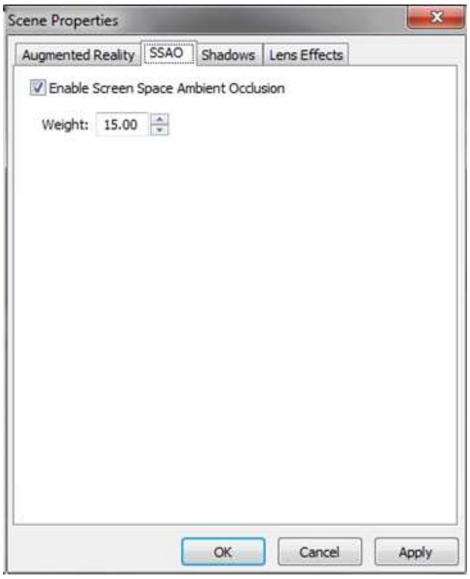
4.4.1. Differential Rendering

Alongside with the AR configuration window, the RPR-SORS Editor also supports the configuration of a Differential Rendering technique, which allows advanced interaction between the virtual objects and the real world, i.e., occlusion, color bleeding and shadow casting between virtual and real objects. This feature, for obvious reasons, is only available when the AR has been activated before. From this window, the user can decide which objects will be nominated virtual and which ones will be phantom. The Differential Rendering configuration pane is shown in Figure 4-17.

Virtual objects are the ones that appear in the final scene, augmenting the reality. Phantom objects, on the other hand, are representations of real objects and act exclusively to provide certain effects in the virtual and real objects, e.g. occlusion and color bleeding.

4.5. Screen-Space Ambient Occlusion

As explained in chapter 3.4.2, Ambient Occlusion is the technique that deals with self-shadows, i.e. the amount of light that is blocked by the object on its own surface. The technique implemented in this solution is a Screen Space Ambient Occlusion (SSAO) and has only one parameter to be adjusted, which is the weight value to determine the strength of the effect, as seen in Figure 4-18.



☑ Enable Screen Space Ambient Occlusion

Weight: 15.00

Figure 4-18. Screen Space Ambient Occlusion configuration pane.

However, the SSAO technique has some integration problems with the HDR images used, generating a less than perfect result at the end, as illustrated in Figure 4-19.

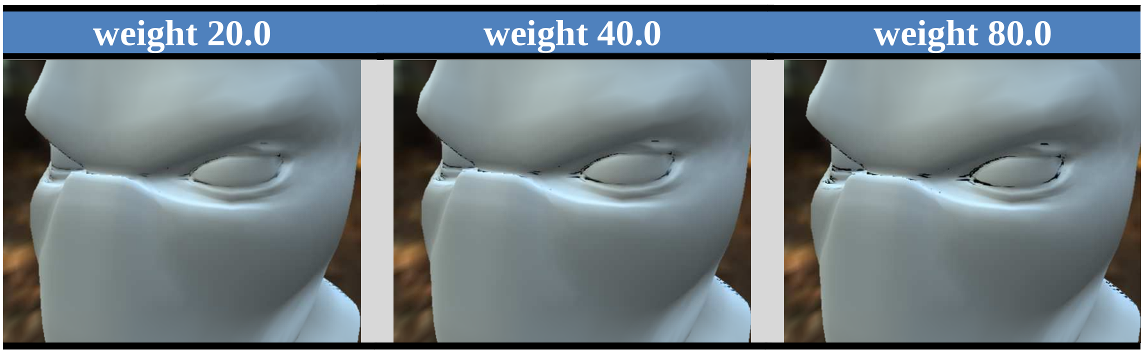
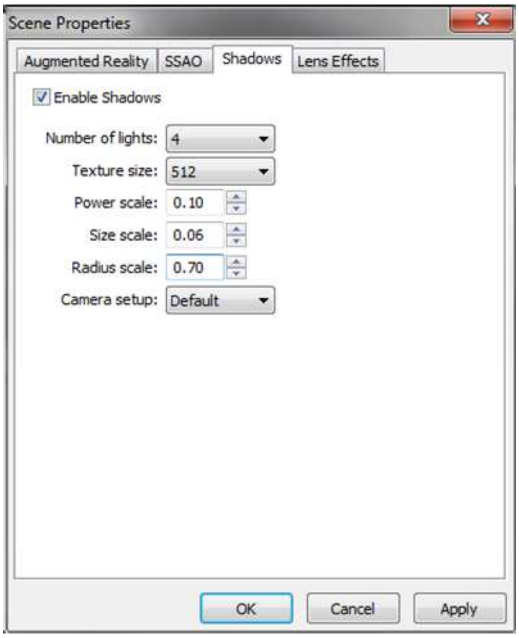


Figure 4-19. Results from the SSAO implemented in this dissertation.

4.6. Projected Shadows

The RPR-SORS Editor features an algorithm to project shadows between objects based on shadow maps. It can infer the most relevant light sources from the environment, calculate the shadows these light project in real object, and reproduce them on the virtual objects. The generated shadows can occlude both virtual objects and real objects through the phantom representation associated with them. Figure 4-20 illustrates the configuration pane used to set the technique parameters:



☑ Enable Shadows

Number of lights: 4

Texture size: 512

Power scale: 0.10

Size scale: 0.06

Radius scale: 0.70

Camera setup: Default

Figure 4-20. Shadows configuration pane.

- **Number of lights**: the number of lights the algorithm will try to infer from the real world. The values it can assume are 1, 2, 4, 8 and 16.

- **Texture size**: the size of the texture used to store the shadow map. The larger the texture, the image will appear less pixelated at the shadow edges.
- **Power scale**: float number $[0, \infty)$ that scales the radiance emitted by each light source.
- **Size scale**: float number $[0, \infty)$ that is used to adjust the size of the light source, varying the results of the generated shadows from hard shadows to soft shadows.
- **Radius scale**: float number $[0, \infty)$ that determines the distance of the light sources. Since the light positions are inferred from a 2D texture, the real distance to the objects in the scene cannot be determined, hence the need for this parameter.
- **Camera setup**: pre-defined camera setups that are used to vary the final shadow result. It can assume three options: default, focused and LISPSM.

The results obtained in this solution are shown in Figure 4-21.

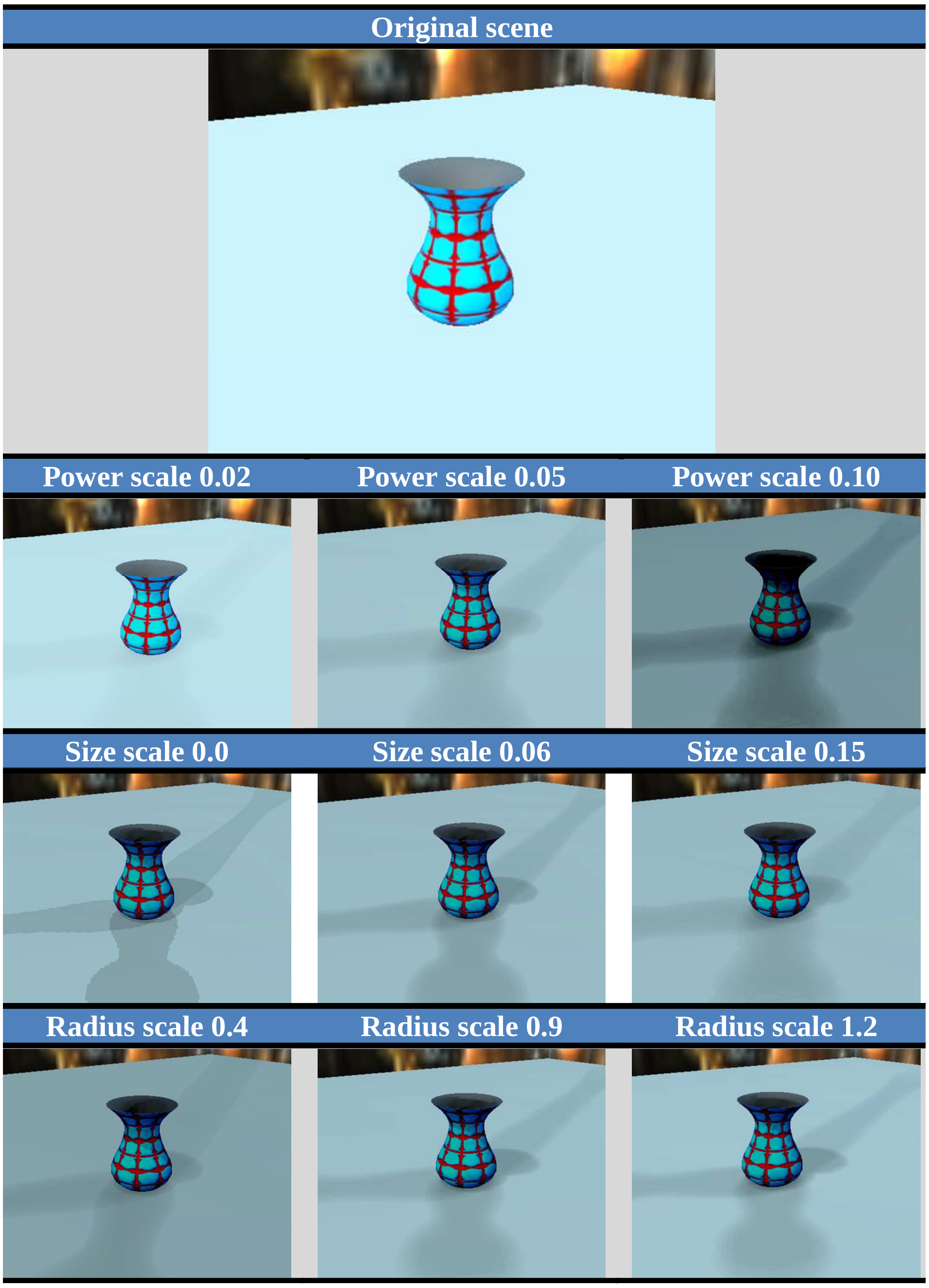
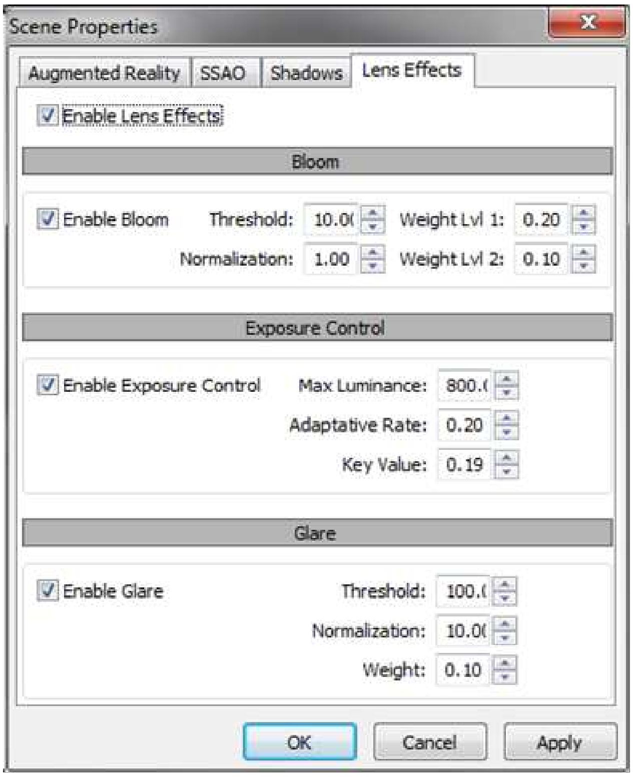


Figure 4-21. Different projected shadows values applied to an object.

4.7. Lens Effects

In the RPR-SORS Editor, three major effects define Lens Effects: bloom, glare and exposure control. Such effects can be activated and configured independently (see Figure 4-22) from each other, according to the scene and user characteristics.



Scene Properties

☑ Enable Lens Effects

Bloom

☑ Enable Bloom
Threshold: 10.0(
Weight Lvl 1: 0.20
Normalization: 1.00
Weight Lvl 2: 0.10

Exposure Control

☑ Enable Exposure Control
Max Luminance: 800.(
Adaptative Rate: 0.20
Key Value: 0.19

Glare

☑ Enable Glare
Threshold: 100.(
Normalization: 10.0(
Weight: 0.10

Figure 4-22. Lens Effects parameters, divided in Bloom, Exposure Control and Glare.

- **Bloom**:

Bloom appears as fringes of light around very bright objects in an image. The physical basis behind the bloom effect is that the light beam that comes from a light source tends to scatter inside the lenses (including human eyes) hitting some spots that would naturally be illuminated by another object. The resulting effects are objects with blurred silhouettes, when visualized against bright background. Figure 4-23 shows the Bloom effect being activated. It is possible to see that the effect is generated by an emissive material, and also by the reflection in the mirror-like material. Bloom, in this implementation, is consists of a two-step blur of the environment light sources, and can be configured by the following parameters:

- o **Bloom threshold**: float value $[0, \infty)$ that is used to determine the least amount of luminance that will be considered bright enough to generate the bloom effect.
- o **Bloom normalization**: float value $[0, \infty)$ that is used to normalize the values from the original image, controlling the luminance that will be considered to the bloom effect.
- o **Bloom weight 1**: float value $[0, \infty)$ that will multiply the first step of the blur in the bloom generation.

- **Bloom weight 2**: float value $[0,\infty)$ that will multiply the second step of the blur in the bloom generation.

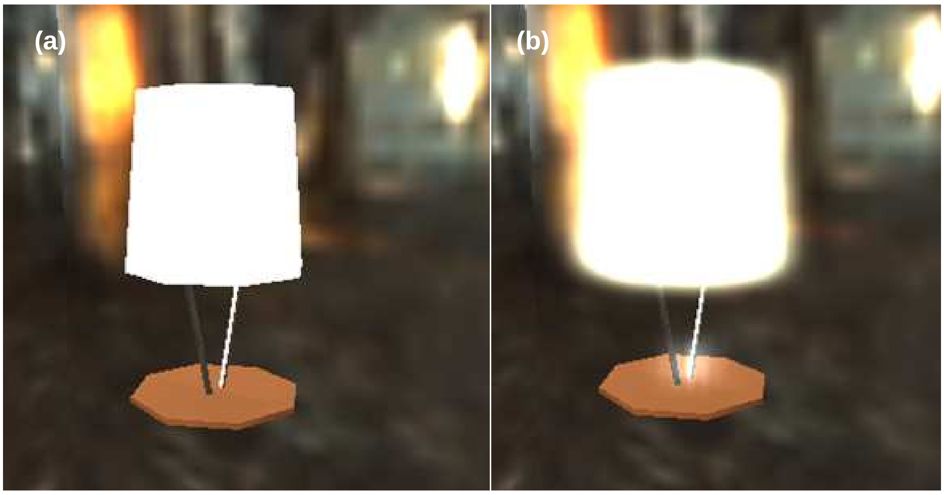


Figure 4-23. Bloom effect before (a) and after (b) being turned on.

- **Exposure Control**:

In real lenses, the size of the aperture, the shutter speed and the brightness of the scene control the amount of light that enters the camera during a period of time. This can determine if the final image will appear brighter or darker. In dynamic scenes, where the illumination changes fast, usually cameras mimic the human eye behavior, which takes some time to adapt to the illumination change. Figure 4-24 shows an example of two view angles, being adjusted by the Exposure Control effect. The RPR-SORS Editor using the following parameters can simulate this effect:

  - **Exposure Control maximum luminance**: float value $[0,\infty)$ that is associated with the brightness of the scene.
  - **Exposure Control adaptive rate**: float value $[0,\infty)$ that controls the speed in which the final image will stabilize in the best exposure value.
  - **Exposure Control key value**: float value $[0,\infty)$ that determines the final exposure level desired.

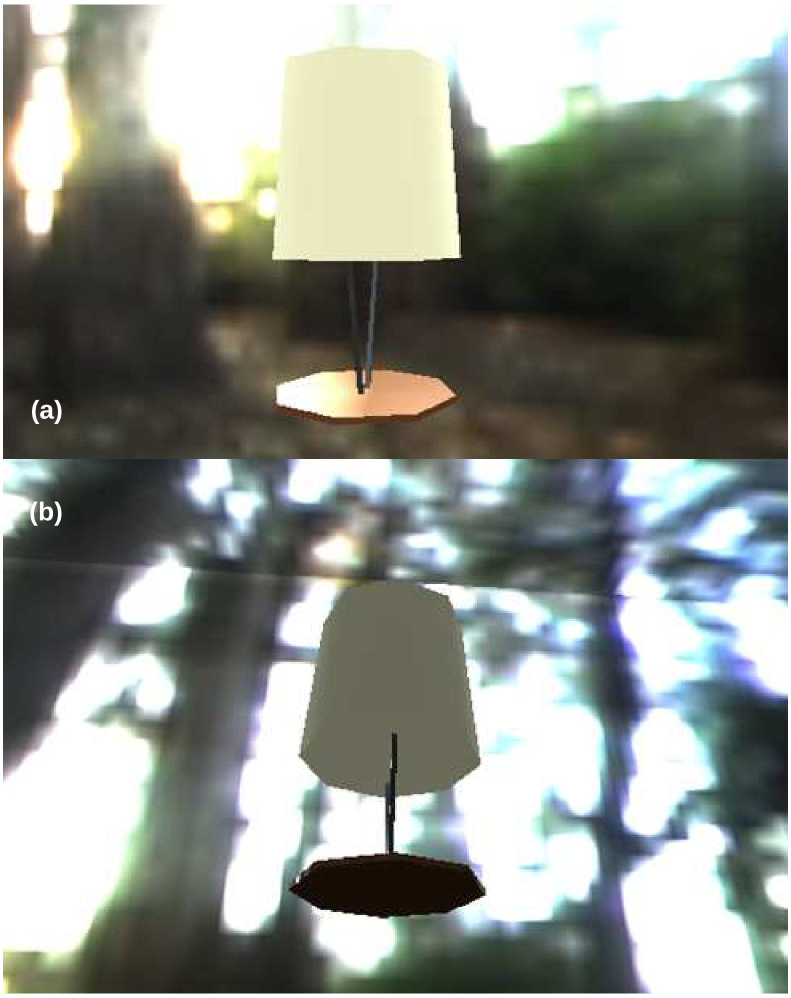


Figure 4-24. Exposure Control demonstration. When the viewer looks from (a) to (b), the brightness of the scene is adjusted.

- **Glare**:

Glare is an effect that is caused by the same reasons as the bloom effect, but appears as light streaks (see Figure 4-25), mostly depending on the lenses and light source intensity. Glare has the following parameters:

- o **Glare threshold**: float value $[0, \infty)$ that is used to determine the least amount of luminance that will be considered bright enough to generate the glare effect.
- o **Glare normalization**: float value $[0, \infty)$ that is used to normalize the values from the original image, controlling the luminance that will be considered to the glare effect.
- o **Glare weight**: float value $[0, \infty)$ that will multiply the final glare effect.

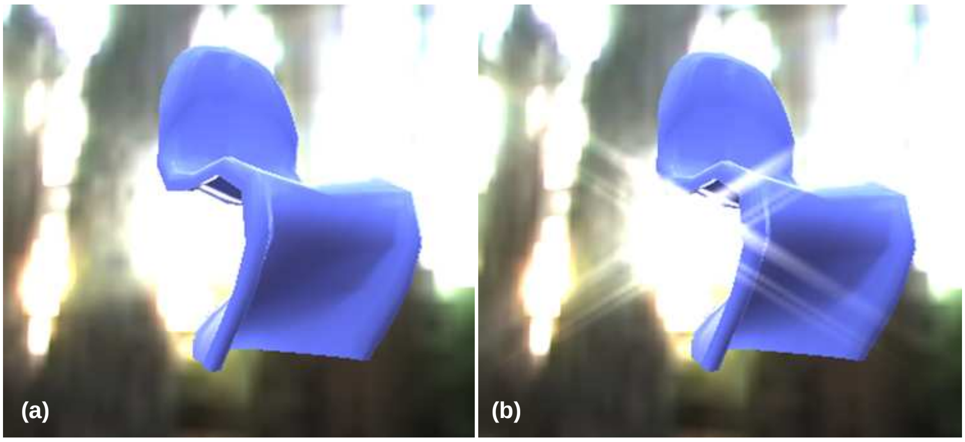
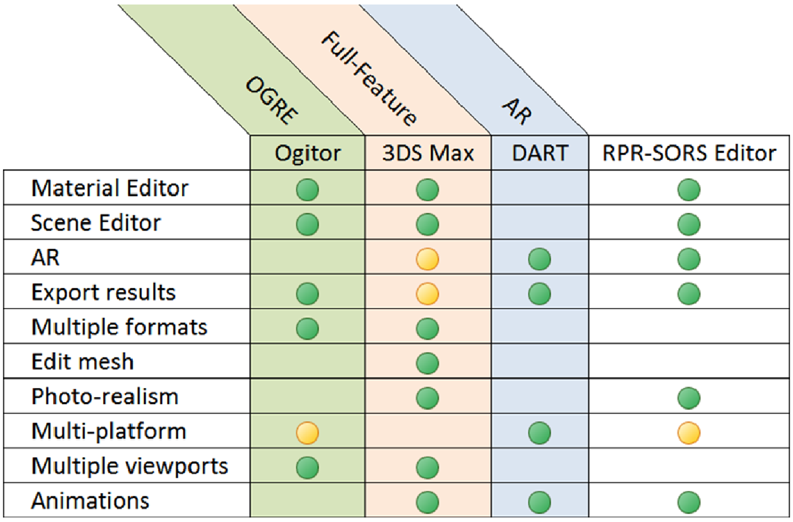


Figure 4-25. Glare effect before (a) and after (b) being turned on.

4.8. Some Considerations

In this chapter, the most important features of the RPR-SORS Editor were described. Such features compose the main experience obtained when using the proposed solution. The mentioned photo-realistic effects were chosen in order to obtain realistic results that fit in this solution scope, however more advanced effects could also become part of the solution, if they can be performed in real-time and integrated within the API.

The proposed solution represents a combination of a Material Editor and an AR authoring tool, containing some features present in both categories. Figure 4-26 shows a comparison chart, featuring three of the most representative applications of OGRE Editors, Scene Editors and AR authoring tools. Green circles represent existing features and yellow circles represent features that could be present in the application through plugins or small changes.



	OGRE	Full-Feature	AR	
	Ogitor	3DS Max	DART	RPR-SORS Editor
Material Editor	🟢	🟢		🟢
Scene Editor	🟢	🟢		🟢
AR		🟡	🟢	🟢
Export results	🟢	🟡	🟢	🟢
Multiple formats	🟢	🟢		
Edit mesh		🟢		
Photo-realism		🟢		🟢
Multi-platform	🟡		🟢	🟡
Multiple viewports	🟢	🟢		
Animations		🟢	🟢	🟢

Figure 4-26. Comparison chart of three most representative applications from OGRE Editor, Scene Editors and AR authoring.

It is possible to observe that the RPR-SORS Editor has many of the features of the three applications. In the following chapter, a case study will be described in order to illustrate these features.

5. Case Study

As mentioned in chapter 4, the RPR-SORS Editor uses the RPR-SORS API in order to accomplish the best photo-realistic results. However, the main goal of the editor comprehends also the ability to transport the created scene to a final application, and not only show the results. This way, an application called Scene Designer was developed, during this dissertation, using the byproducts of the RPR-SORS Editor in order to evaluate the editor features. This chapter will describe the process of creation in the proposed editor, and then how to use the generated results in a custom application.

The Scene Designer application consists of a tool for architects and salesmen to show a possible set up of a house or flat to the possible buyer without the need to physically be in the place. The tool features a realistic rendering of the objects that can be placed in the building, so that the buyer can move them anywhere within the model to find the best layout. For this task, the Scene Designer also uses the RPR-SORS API, which provides the photo-realistic rendering. The initial window of the proposed case study is shown in Figure 5-1.

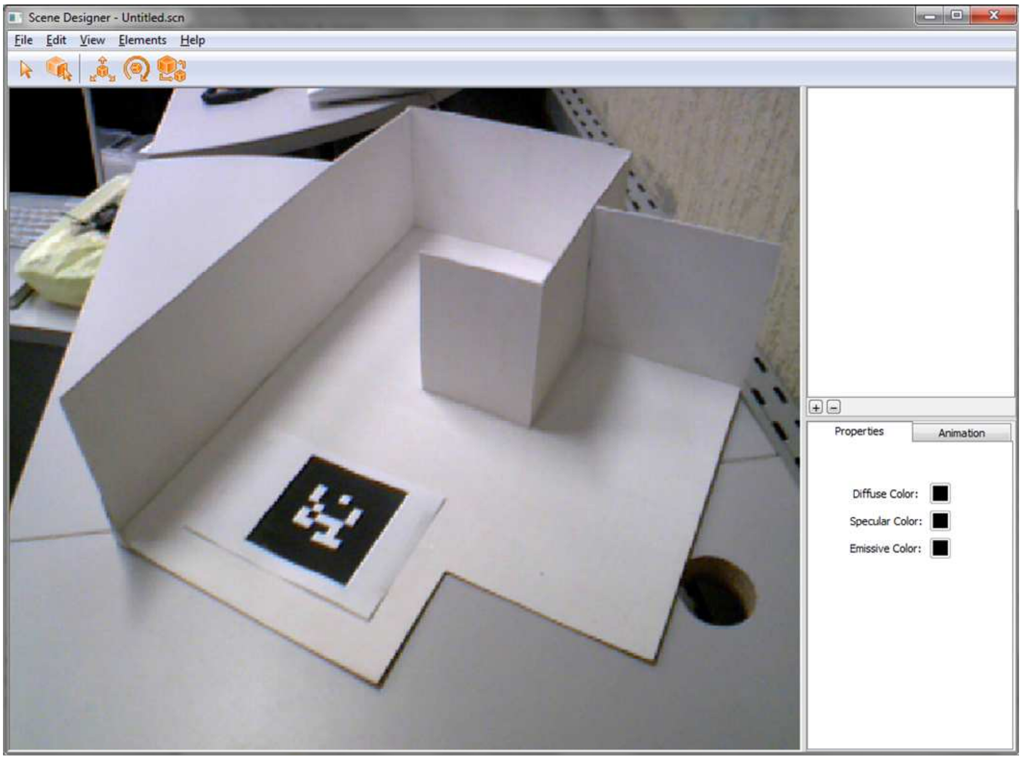


Figure 5-1. Scene Designer main window.

However, before using the Scene Designer, the first step in the process is to create, in the RPR-SORS Editor, the scene that is going to be visualized. In order to do that, it is possible to create a scene from the beginning or open a previous scene from a saved file. In this case study, a photo-realistic object will be created from a clean scene. The user must then add the desired mesh in the editor. It will be loaded with a default material, as illustrated in Figure 5-2. It is possible to add objects in two places in the interface, as indicated by the highlighted areas of the image.

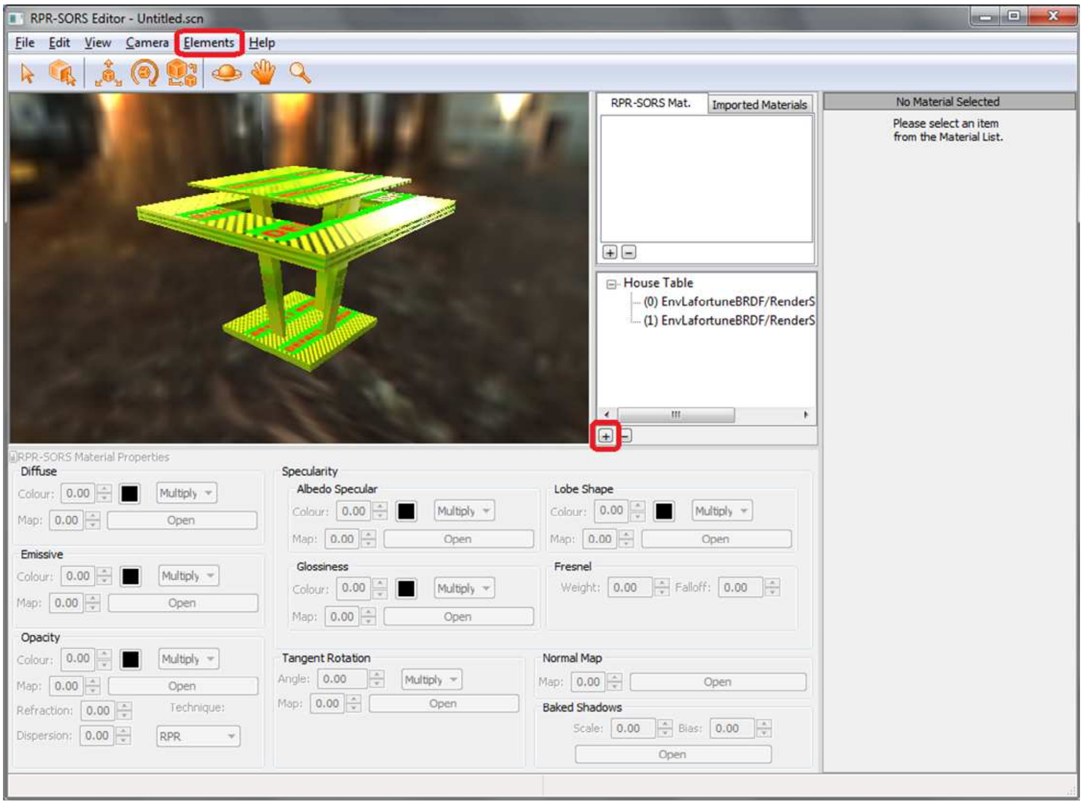


Figure 5-2. RPR-SORS Editor window after adding an object mesh.

After the object is created, it is necessary to create the photo-realistic materials that will be applied to it. In Figure 5-2 it is possible to see that the created object (named House Table) has two sub-meshes. Therefore, two materials were created (Metal Top and Wooden Body). Figure 5-3 contains two green highlighted areas that are used to create materials. In order to apply the material to a sub-mesh, it is necessary to activate the Sub-mesh Selection Tool (blue highlight in Figure 5-3) and select a section of the object. Then, the user must right-click the desired material and select the “Apply to sub-entity” option. Since new materials are allocated with default properties and texture, the appearance of the object does not change when the newly created material is applied. However, every change that is made in the applied material will now appear in real-time.

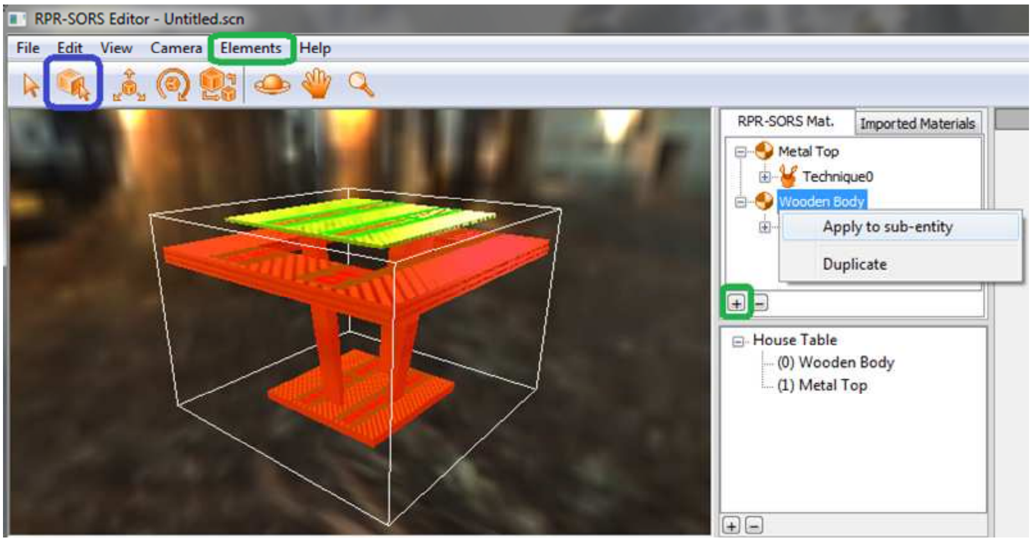
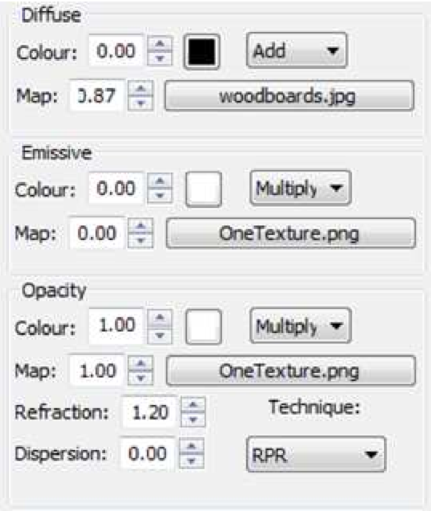


Figure 5-3. Newly created materials being applied to the object.

The next step in the process is to configure every parameter from the materials, in order to achieve the desired appearance. The material named Wooden Body will be described first and in more details, and then some parameters for the material named Metal Top will be described.

The first set of parameters is from the diffuse color. In this case, the final color of the object is provided by the texture. This way, the color weight is set to 0.0 (zero) and the operation set to add. Then the map weight is set to 0.87 and the wooden texture is chosen from a JPG file. This value was chosen in order to make the object appearance darker than the original texture. Emissive parameters were all set to 0.0 (zero), since the material is a usual reflective object, thus do not emit light. Opacity parameters were set to 1.0 (one), since the object is completely opaque. Refraction and Dispersion parameters have no effect in this case. Figure 5-4 illustrates the configuration of the mentioned parameters.

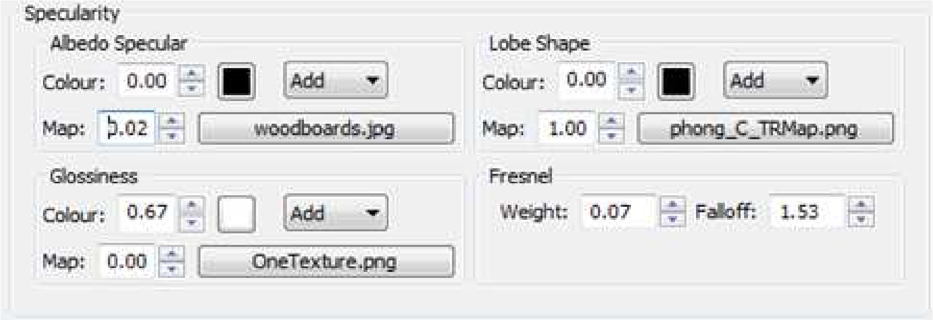

Diffuse
Colour: 0.00 ■ Add
Map: 0.87 woodboards.jpg

Emissive
Colour: 0.00 □ Multiply
Map: 0.00 OneTexture.png

Opacity
Colour: 1.00 □ Multiply
Map: 1.00 OneTexture.png
Refraction: 1.20
Technique: RPR
Dispersion: 0.00

Figure 5-4. Parameter configuration.

The Specularity parameters were set as follows: the specular albedo considers only the texture parameter, similar to the diffuse component. The value set for this component was 0.02 for the map weight, and the texture is the same as the diffuse component. This value defines that the object reflects a small amount of light, simulating what would be a coat of varnish on the surface of the table. The Glossiness factor was defined by a value of 0.67, which indicates that the reflection provided by the varnish is glossy. The Lobe Shape used was defined by a texture that corresponds to Phong's reflective model. In addition, the Fresnel factor has the values 0.07 for the weight and 1.53 for function falloff. These values ensure that there is a little increase of specularity at grazing angles. Figure 5-5 illustrates the specular component configuration.



Specularity

Albedo Specular
Colour: 0.00 Add
Map: 0.02 woodboards.jpg

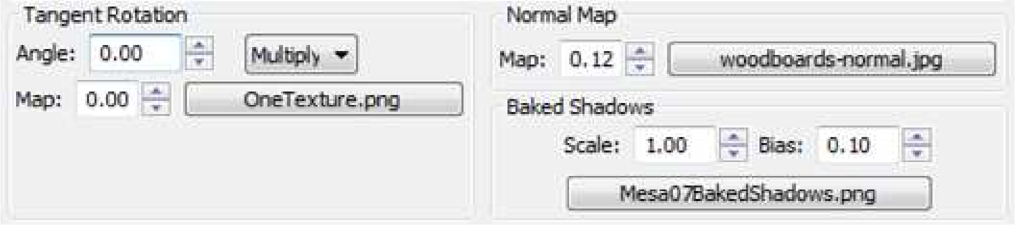
Lobe Shape
Colour: 0.00 Add
Map: 1.00 phong_C_TRMap.png

Glossiness
Colour: 0.67 Add
Map: 0.00 OneTexture.png

Fresnel
Weight: 0.07 Falloff: 1.53

Figure 5-5. Specularity parameters configuration.

Finally, the last photo-realistic parameters to be configured: Tangent Rotation is not applied in this example, so all the parameters were set to 0.0. A Normal Map texture, matching the diffuse texture used, was applied with a weight of 0.12. This value defines that the roughness of the surface is not too high. A Baked Shadows texture was generated for this object, and then applied with the values 1.00 for scale and 0.10 for bias. These parameters are used to define a soft shadow for the object. The mentioned parameters are shown in Figure 5-6.



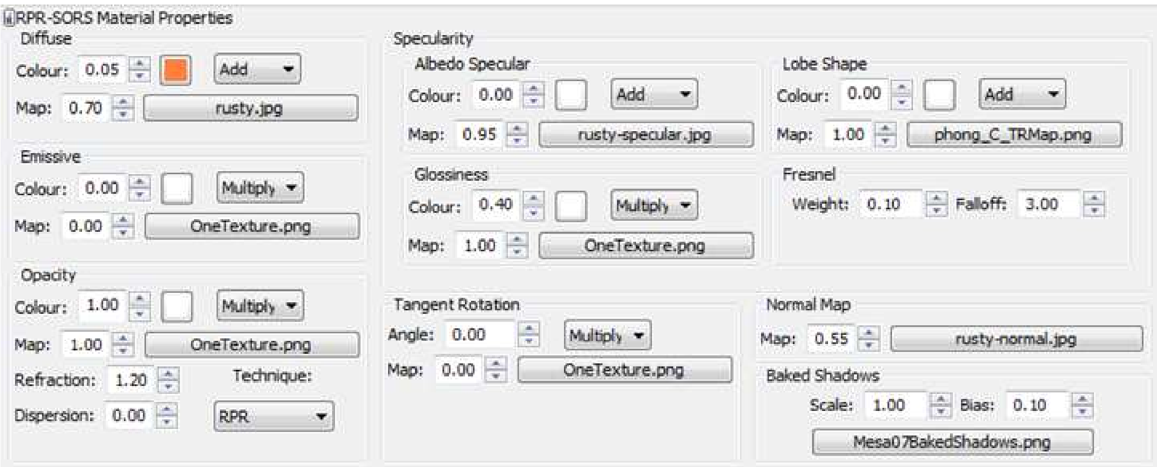
Tangent Rotation
Angle: 0.00 Multiply
Map: 0.00 OneTexture.png

Normal Map
Map: 0.12 woodboards-normal.jpg

Baked Shadows
Scale: 1.00 Bias: 0.10
Mesa07BakedShadows.png

Figure 5-6. Tangent Rotation, Normal Map and Baked Shadows parameter configuration.

These are the necessary parameters to be configured for a photo-realistic material. In addition the material named Metal Top was configured in the similarly, setting textures and parameters in order to obtain a rusty metal appearance. The final parameter configuration for the Metal Top material is shown in Figure 5-7



RPR-SORS Material Properties

Diffuse
Colour: 0.05 Add
Map: 0.70 rusty.jpg

Emissive
Colour: 0.00 Multiply
Map: 0.00 OneTexture.png

Opacity
Colour: 1.00 Multiply
Map: 1.00 OneTexture.png
Refraction: 1.20 Technique: RPR
Dispersion: 0.00

Specularity

Albedo Specular
Colour: 0.00 Add
Map: 0.95 rusty-specular.jpg

Glossiness
Colour: 0.40 Multiply
Map: 1.00 OneTexture.png

Lobe Shape
Colour: 0.00 Add
Map: 1.00 phong_C_TRMap.png

Fresnel
Weight: 0.10 Falloff: 3.00

Tangent Rotation
Angle: 0.00 Multiply
Map: 0.00 OneTexture.png

Normal Map
Map: 0.55 rusty-normal.jpg

Baked Shadows
Scale: 1.00 Bias: 0.10
Mesa07BakedShadows.png

Figure 5-7. Parameter configuration for the Metal Top material.

Afterwards, the Lens Effects were turned on, in order to bring a smooth blending between the object and the background. The final visual result, after configuring all the mentioned parameters is shown in Figure 5-8.

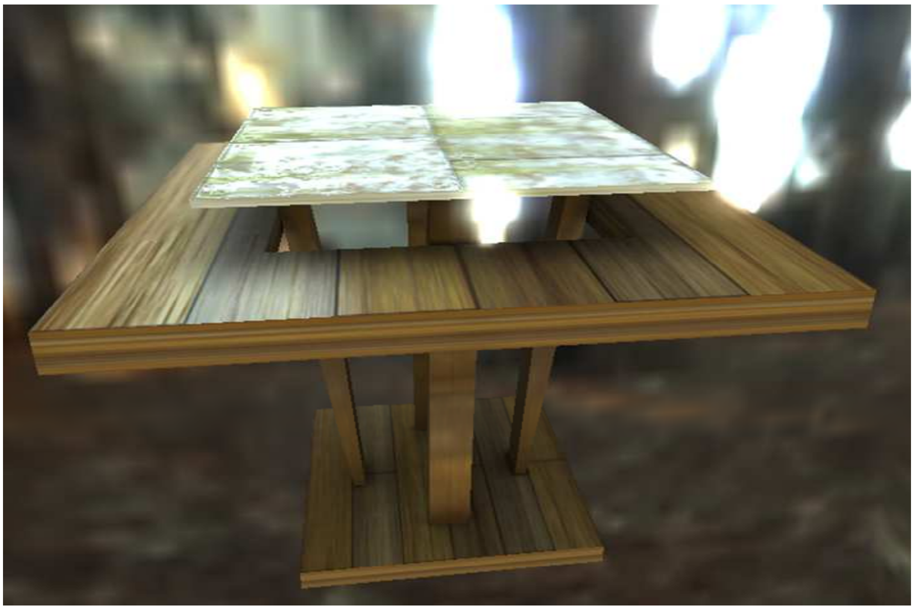
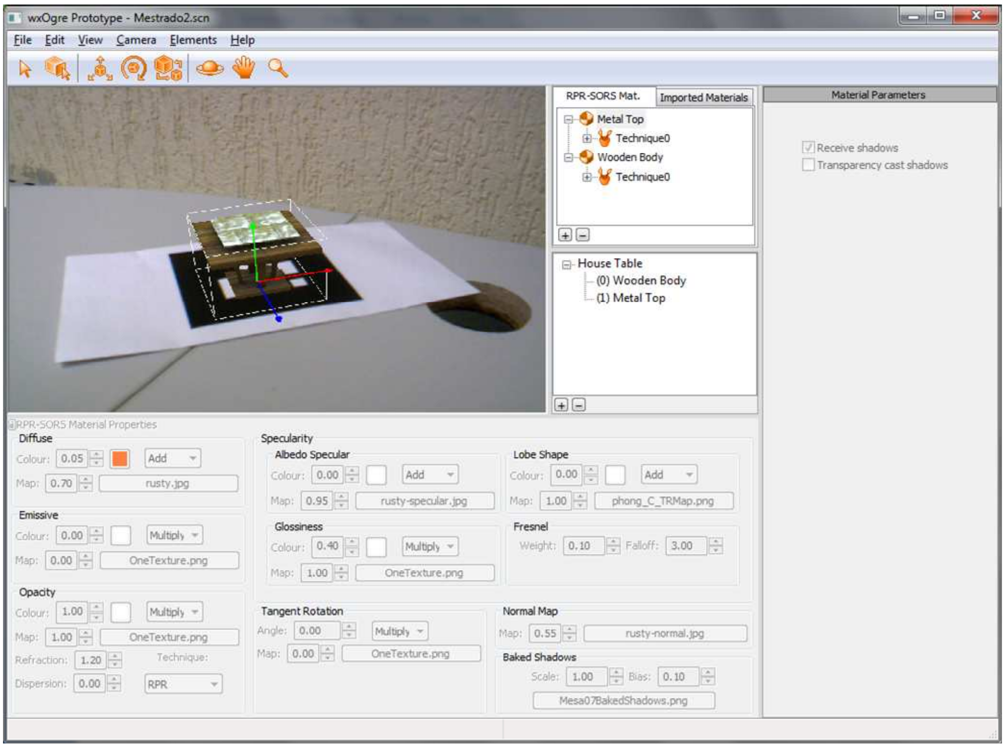


Figure 5-8. Visual result for the RPR-SORS parameters and the Lens Effects.

From this point it is already possible to export the object to be used in the Scene Designer application. However, first the model will be visualized in the AR environment, to verify if every effect is being presented correctly. Figure 5-9 shows the RPR-SORS Editor with AR mode activated. It is still possible to edit the object parameters and see the results interactively.



☑ Receive shadows
□ Transparency cast shadows

RPR-SORS Material Properties

Diffuse
Colour: 0.05 Add
Map: 0.70 rusty.jpg

Emissive
Colour: 0.00 Multiply
Map: 0.00 OneTexture.png

Opacity
Colour: 1.00 Multiply
Map: 1.00 OneTexture.png
Refraction: 1.20 Technique: RPR
Dispersion: 0.00

Specularity
Albedo Specular
Colour: 0.00 Add
Map: 0.95 rusty-specular.jpg

Glossiness
Colour: 0.40 Multiply
Map: 1.00 OneTexture.png

Lobe Shape
Colour: 0.00 Add
Map: 1.00 phong_C_TRMap.png

Fresnel
Weight: 0.10 Falloff: 3.00

Tangent Rotation
Angle: 0.00 Multiply
Map: 0.00 OneTexture.png

Normal Map
Map: 0.55 rusty-normal.jpg

Baked Shadows
Scale: 1.00 Bias: 0.10
Mesa07BakedShadows.png

Figure 5-9. Augmented Reality mode in the RPR-SORS Editor.

After the process of creation and testing, the modeled object and materials were exported to a folder, and then used in the Scene Designer application. The same process described to create the mentioned table object was used to create a set of materials and objects used in the case study.

In order to illustrate most of the features of the proposed solution the Scene Designer application was developed using a real model of an apartment (previously crafted with stiff paper from a blueprint). The replica was pre-modeled intentionally without some walls for the best visualization of the objects within. In addition, it is possible to create the missing walls virtually, and have the full-vision enabled if desired. Afterwards, the virtual model was created, mimicking the real model as perfect as possible in order to be used as a phantom object. The blueprint used as reference, as well as the real and virtual models are shown in Figure 5-10. However, the final match of the virtual model with real model is not ideal – some visualization angles feature a variance between real and virtual walls – because of camera distortion and the real model wall gets slightly curved with time.

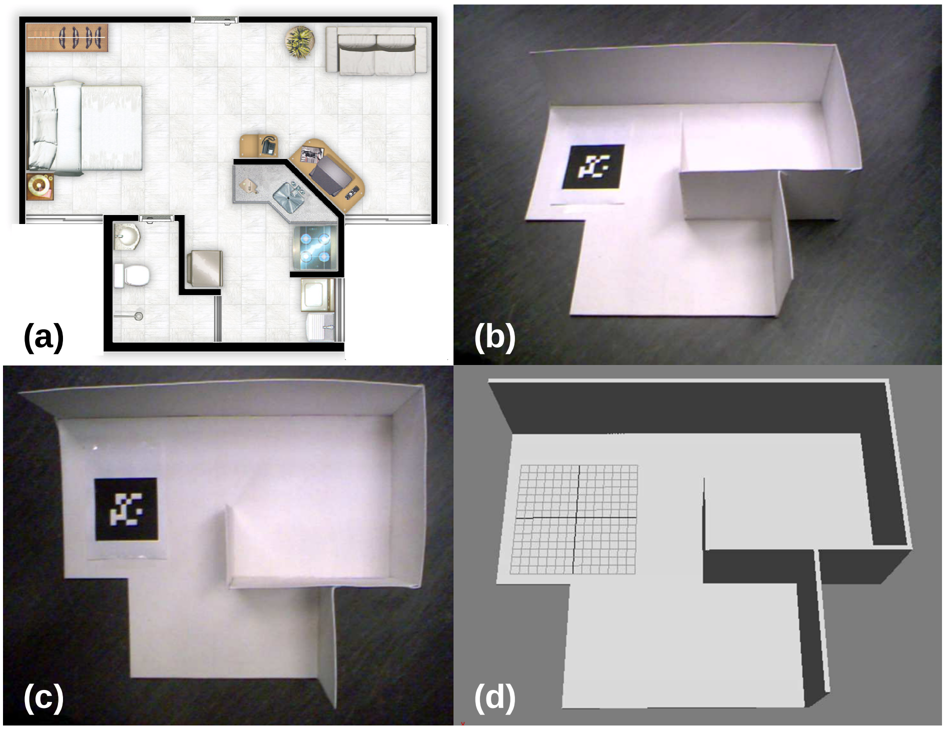


Figure 5-10. Crafted model references: (a) original blueprint, (b) and (c) real model with stiff paper, and (d) virtual model.

Regarding the rendering in the Scene Designer screen, all the effects were turned on, with one exception for the Ambient Occlusion. For the reason that objects already had baked shadows texture, which is cheaper to use in static meshes and produces smoother results, this approach was chosen. In addition, objects' materials were created in the RPR-SORS Editor, aiming the perfect appearance in comparison to real ones, and then exported to straightforward use within the scene designer. Due to time constraints, just the living room furniture was modeled, comprehending TV racks, tables, center tables, shelves, sofas, chairs, luminaries, paintings, and carpets. It is possible to select one of the items from the selection screen, which is split by category.

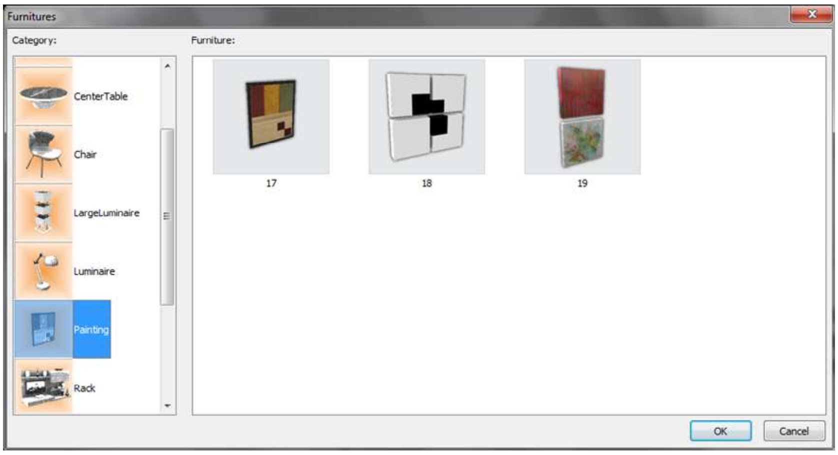


Figure 5-11. Object selection window from the Scene Designer application.

It is important to detail the exporting process to highlight how easy other applications may benefit from the RPR-SORS Editor. It can save the current scene for later use, using the .scn format, which consists of a zip file containing the important resources. This format can be used as long as the editing and refining process is finished, because it simplifies the task of opening the scene from the last state. When the desired result is achieved, the user can then export the chosen types (materials, meshes and/or textures) to any folder. The last step in the process is to add this folder to the resource paths of the final application, if it is not already added. After that, the files are ready to be used by an RPR-SORS API enabled application. The results of a scene composed in the Scene Designer are illustrated in Figure 5-12 and Figure 5-13.

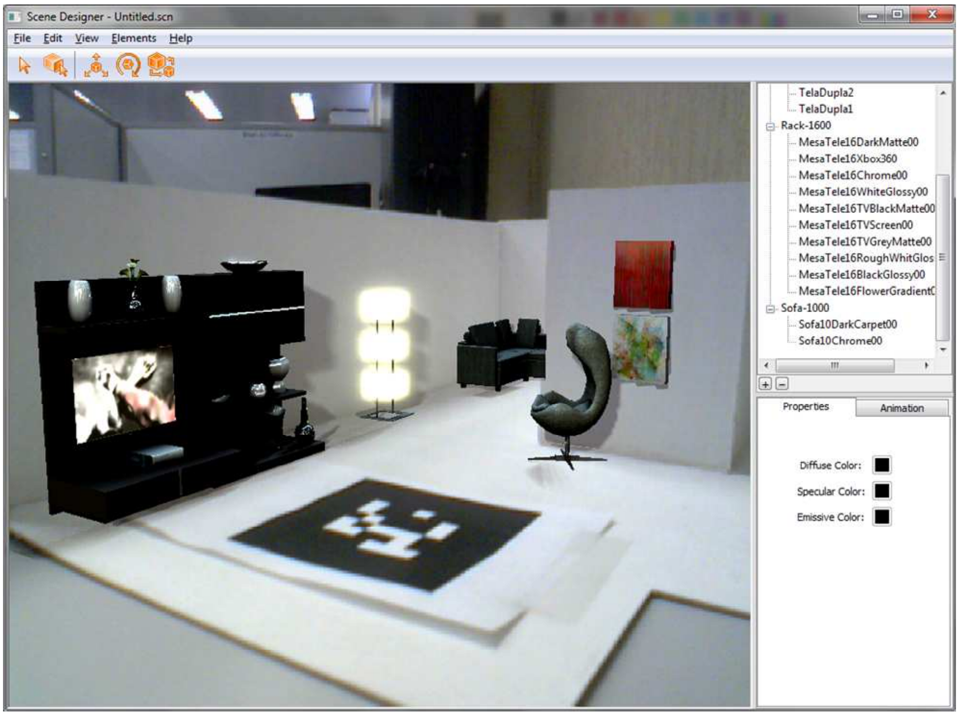


Figure 5-12. Photo-realistic scene composed in the Scene Designer.

The Scene Designer application allows the user to move, scale and rotate the imported objects as well as simple material modifications, such as diffuse, specular and emissive color change. In addition, if the object has animations, e.g. a rotating fan or cabinet door, the Scene Designer supports toggling the animation on and off, as well as transition through any state of the animation.

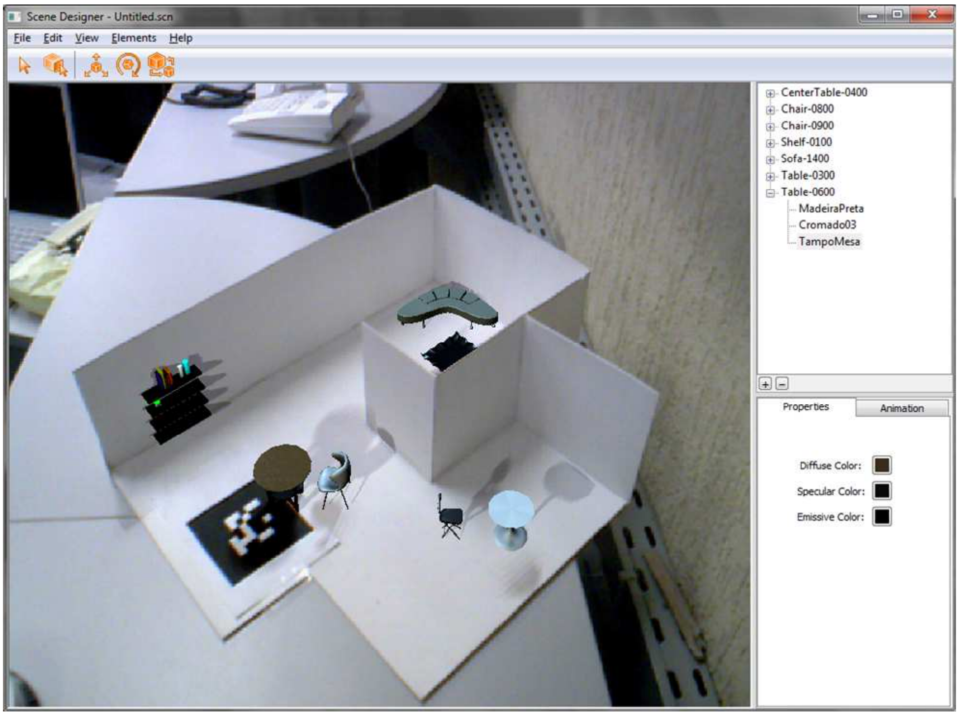


Figure 5-13. Different objects can be placed in the application looking for the best layout.

5.1. FINAL CONSIDERATIONS

This chapter described a case study of the RPR-SORS Editor, detailing the required step to create a simple photo-realistic scene and export the results for further use. The main goal was to illustrate a step-by-step process, where the configurations used could be reproduced by a user of the application. The Scene Designer application is an example of application that uses the exported products of the RPR-SORS Editor to compose a possible commercial solution.

Analyzing the performance of RPR-SORS Editor and Scene Designer applications, it was observed that the amount of objects in the scene is the most relevant factor for the frame rate. Figure 5-14 shows a performance graph illustrating the weight of each technique from the photo-realistic API. It is possible to see that, in exception to the SSAO technique, the enabled effects drop the frame rate by6%-18%. The SSAO technique is the heaviest one, requiring multiple shader steps, and drops the frame rate by almost 50%. It is also possible to observe that even when all the effects are enabled simultaneously, the frame rate is still maintained in real-time values.

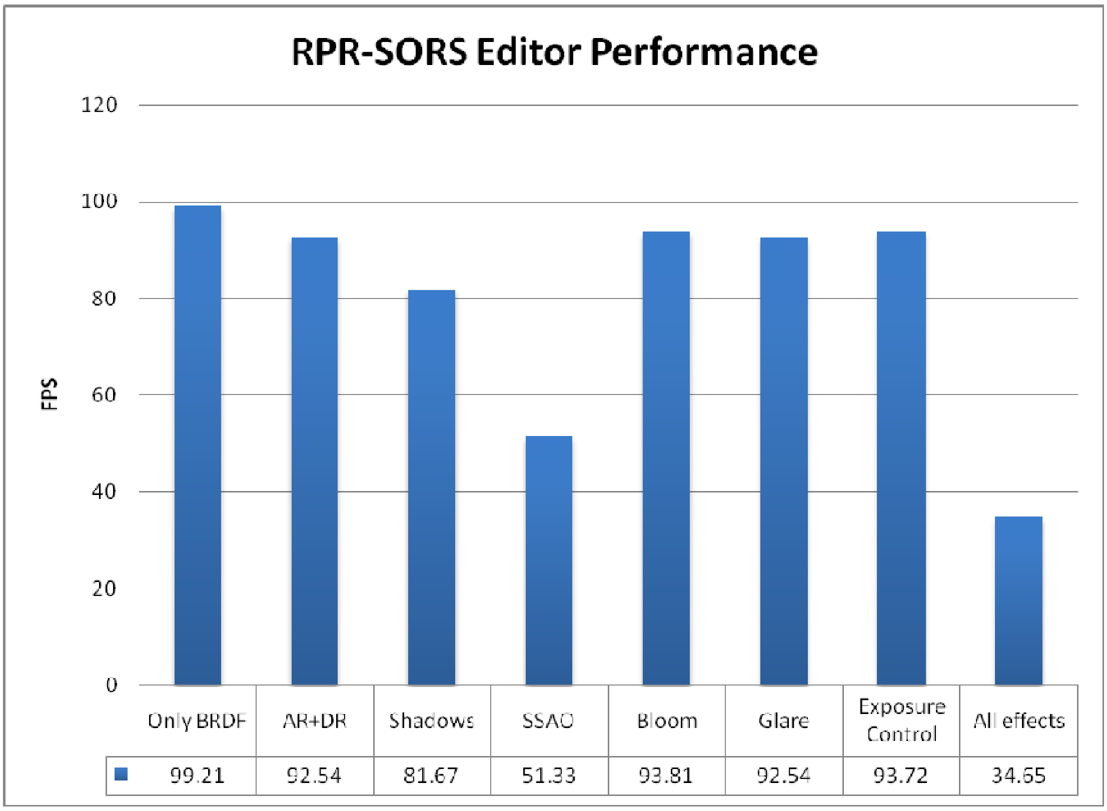
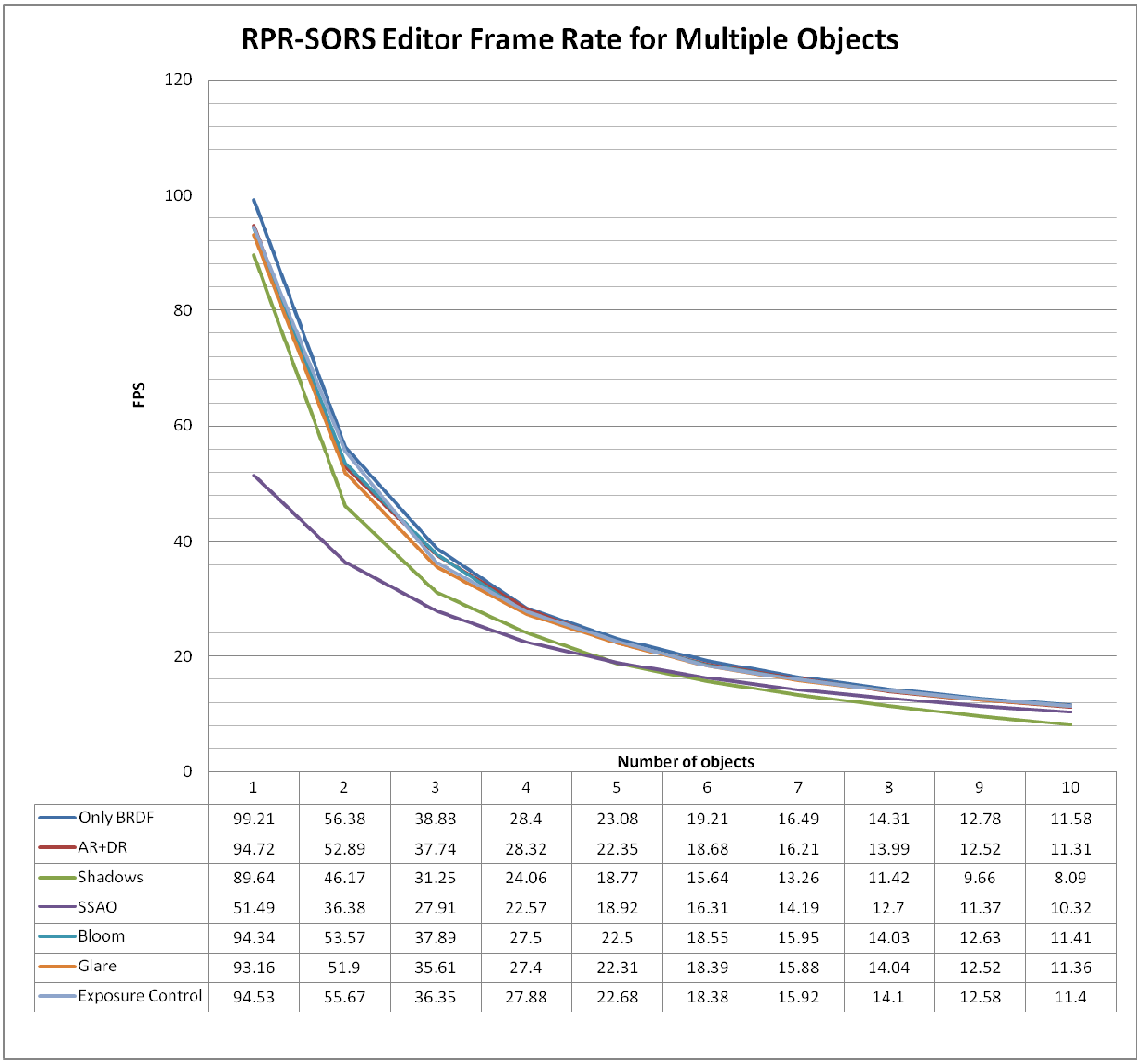


Figure 5-14. Frame rate variation for each independent effect rendering a single object.

On the other hand, the number of virtual objects inserted on the scene has a great impact in the performance. The reason of this bottleneck is the environment map that is generated for each object every frame. This map is necessary to update the interactions on the object BRDF (e.g., reflections, indirect illumination). Figure 5-15 shows a performance graph of the different effects being applied to a varying number of virtual objects.

The final conclusions obtained by this dissertation will be described in the next chapter.



Number of objects	1	2	3	4	5	6	7	8	9	10
Only BRDF	99.21	56.38	38.88	28.4	23.08	19.21	16.49	14.31	12.78	11.58
AR+DR	94.72	52.89	37.74	28.32	22.35	18.68	16.21	13.99	12.52	11.31
Shadows	89.64	46.17	31.25	24.06	18.77	15.64	13.26	11.42	9.66	8.09
SSAO	51.49	36.38	27.91	22.57	18.92	16.31	14.19	12.7	11.37	10.32
Bloom	94.34	53.57	37.89	27.5	22.5	18.55	15.95	14.03	12.63	11.41
Glare	93.16	51.9	35.61	27.4	22.31	18.39	15.88	14.04	12.52	11.36
Exposure Control	94.53	55.67	36.35	27.88	22.68	18.38	15.92	14.1	12.58	11.4

Figure 5-15. Performance graph showing the effects being applied to multiple objects.

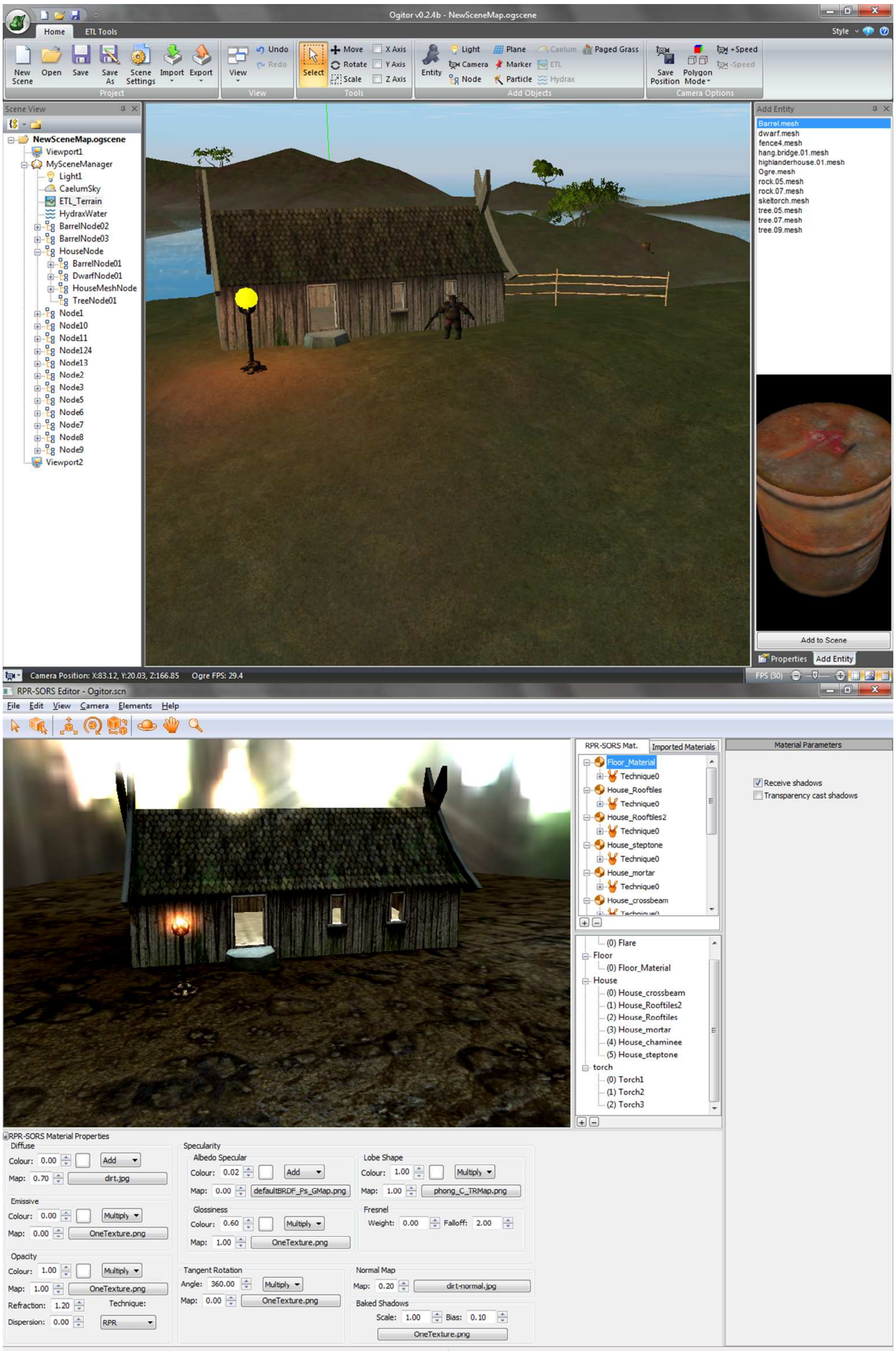
6. CONCLUSIONS AND FUTURE WORK

This work brought a solution to the problem of photo-realistic AR applications and draws some interesting results. It focused the step of object creation and proposed a scene editor capable of editing the parameters of many CG techniques in order to achieve a photo-realistic result. This proposal was evaluated by the development of an application that used the results of the editor as input, corroborating the editor as an authoring tool.

The overall performance of the solution was restrained within interactive rates. It was observed that the application is more sensitive to the amount of virtual objects in the scene than the number of techniques running simultaneously. This happens due to the cost of computing the environment map for each object. However, this restriction is not so relevant, since it was observed that the most common use of the editor is to refine the appearance of a single object, inserting few additional objects to study the interaction between them, although the composition of an extremely complex scene would decrease the frame rate and compromise the user interaction.

It was shown that the use of the RPR-SORS Scene Editor could help the construction of photo-realistic objects for later use in a final application. Figure 6-1 shows a comparison result of a similar scene, composed in Ogitor OGRE editor and in the RPR-SORS Editor. The scene in the RPR-SORS Editor was composed trying to insert most of the elements from a template scene from the Ogitor Editor, adding the photo-realistic effects.

The proposed editor displays most of the relevant parameters on the user screen allowing real-time visual feedback while changing the values. This feature proved useful to the users in order to refine the results by small amounts and help the achievement of the finest visual results. In addition, the techniques implemented in the RPR-SORS API can be changed and upgraded with time, leaving an open path for improvements.



Ogitor v0.2.4b - NewSceneMap.ogscene

Camera Position: X:83.12, Y:20.03, Z:166.85 Ogre FPS: 29.4

RPR-SORS Editor - Ogitor.scn

Material Parameters

☑ Receive shadows
□ Transparency cast shadows

RPR-SORS Material Properties

Diffuse
Colour: 0.00 Add
Map: 0.70 dirt.jpg

Emissive
Colour: 0.00 Multiply
Map: 0.00 OneTexture.png

Opacity
Colour: 1.00 Multiply
Map: 1.00 OneTexture.png
Refraction: 1.20 Technique: RPR
Dispersion: 0.00

Specularity
Albedo Specular
Colour: 0.02 Add
Map: 0.00 defaultBRDF_Ps_GMap.png
Glossiness
Colour: 0.60 Multiply
Map: 1.00 OneTexture.png
Lobe Shape
Colour: 1.00 Multiply
Map: 1.00 phong_C_TRMap.png
Fresnel
Weight: 0.00 Falloff: 2.00

Tangent Rotation
Angle: 360.00 Multiply
Map: 0.00 OneTexture.png

Normal Map
Map: 0.20 dirt-normal.jpg
Baked Shadows
Scale: 1.00 Bias: 0.10
OneTexture.png

Figure 6-1. Comparison of a similar scene rendered in Ogitor (top) and RPR-SORS Editor (bottom).

Many AR applications can benefit from the use of this proposal, which can decrease the overall time spent on the development of an AR application. Only two requirements are necessary: the final application must use OGRE as render engine, and it must include the RPR-SORS API for the advanced CG techniques.

6.1. CONTRIBUTIONS

The main contributions of this dissertation can be resumed as follows:

- Development of a scene editor, which integrates the RPR-SORS API with OGRE and the wxWidgets library, focusing on photo-realistic AR. The solution fits in the creation step of the development pipeline, facilitating the design of realistic virtual objects in Virtual and Augmented Reality applications.
- Scene Designer case study comprehending an application for the architectural business.
- One short paper accepted on international conference titled "A Global Illumination and BRDF Solution Applied to Photorealistic Augmented Reality" in IEEE Virtual Reality 2009.
- Two full papers accepted on conferences of Virtual and Augmented Reality. The first one, titled "Illumination Techniques for Photorealistic Rendering in Augmented Reality", in the national conference Symposium of Virtual and Augmented Reality in 2008, and the second one, titled "Photorealistic Rendering for Augmented Reality: A Global Illumination and BRDF Solution" in IEEE Virtual Reality 2010, international conference.

6.2. FUTURE WORK

Some future works have been identified as evolution of this work and are explained below:

- Addition of more CG techniques into the pipeline in order to increase the realism of some objects. It was observed that there are still some types of objects that cannot be realistic rendered by this solution, such as translucent objects. The study of advanced techniques to integrate the set of effects could provide more sophisticated scenes.
- Performance improvement. The overall performance of the scene editor can be improved to allow the addition of several objects for simultaneous edition, allowing the composition of a complete scene without reducing the frame rates to unusable levels.
- Optimization of exported code. The material and shader files exported by the application currently contain all the possible effects that can be

enabled in the object. However, some objects can be essentially simpler than others (e.g. completely diffuse objects do not require specular textures), and thus allocate unnecessary resources. It is desirable that the application can detect unused resources and removes them from the material and shader files, in order to optimize the performance of code executed in the final application.

- Support for more mesh file formats. Currently the application supports only OGRE mesh files, however there are many other file formats that could be useful in the application without the need to convert them first (e.g., obj, 3ds).
- Case study extension. The case study presented in this dissertation was implemented with a limited set of models to be used, comprising only the living room of the house. There are many other models to be created and extend the experience of the application, such as bathroom, kitchen and bedroom models.
- Further usability study in order to refine the user interface. The current application interface is functional and allows the manipulation of every important parameter. However, formal usability studies are required to make the user interface simpler and more intuitive, without crippling the capacity of editing so many parameters precisely.

7. REFERENCES

[1] J. McGrenere and G. Moore, "Are We All In the Same "Bloat"?". In Proceedings of Graphics Interface, Montreal, pp. 187-196, 2000.

[2] S. Pessoa, E. Apolinário, G. Moura, J. Lima, M. Bueno, V. Teichrieb, and J. Kelner, "Illumination Techniques for Photorealistic Rendering in Augmented Reality". In Proceedings of Symposium of Virtual and Augmented Reality, João Pessoa, pp. 223-231, 2008.

[3] K. Agusanto, L. Li, Z. Chuangui, and N.W. Sing, "Photorealistic Rendering for Augmented Reality Using Environment Illumination", ISMAR, IEEE Computer Society, Los Alamitos, pp. 208-216, 2003.

[4] S. Rusinkiewicz. "A Survey of BRDF Representation for Computer Graphics", 1997. [Online]. Available: Szymon Rusinkiewicz's home page. URL: http://www.cs.princeton.edu/~smr/cs348c-97/surveypaper.html [Accessed: Nov. 30, 2009].

[5] 3D Total Tutorials, "3D Total Home Page". [Online]. Available: http://www.3dtotal.com/team/Tutorials/discmage/discmage1.asp [Accessed: Jan. 13, 2010].

[6] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino. "Augmented Reality: A Class of Displays on the Reality-Virtuality Continuum". Telemanipulator and Telepresence Technologies, SPIE, V.2351, 1994.

[7] J. Fischer, D. Bartz, and W. Straßer. "Reality Tooning: Fast Non-Photorealism for Augmented Video Streams". In Proceedings of the 4th IEEE/ACM ISMAR, pp. 186-187, 2005.

[8] R. T. Azuma. "A Survey of Augmented Reality". Presence, vol. 6, pp. 355-385, 1997.

[9] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. "Computer Graphics: Principles and Practice". Addison-Wesley Publishing Company, Second Edition, 1996.

[10] ARToolKit, "ARToolKit Home Page". [Online]. Available: http://www.hitl.washington.edu/artoolkit/ [Accessed: Dec. 10, 2008].

[11] L. Teixeira, M. Loaiza, A. Raposo and M. Gattass. "Um sistema híbrido para rastreamento baseado em esferas retrorreflexivas e características do objeto

rastreado". In Proceedings of Symposium of Virtual and Augmented Reality, João Pessoa, pp. 28-35, 2007.

[12] J. P. Lima, "Online Monocular Markerless Augmented Reality: A Survey". IEEE TVCG, to be accepted.

[13] P. Debevec, "Image-based lighting," Computer Graphics and Applications, IEEE , vol.22, no.2, pp. 26-34, Mar/Apr 2002.

[14] P. Debevec, "Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography". In Proceedings of SIGGRAPH '98, pp. 189-198, ACM Press, 1998.

[15] J. Ferwerda, "Three varieties of realism in computer graphics". In Proceedings of the SPIE Human Vision and Electronic Imaging Conference '03, Santa Clara, CA, pp. 290-297, 2003.

[16] R. Welch, T. Blackmon, A. Liu, B. Mellers, and L. Stark, "The Effects of Pictorial Realism, Delay of Visual Feedback, and Observer Interactivity on the Subject Sense of Presence". Presence: Teleoperators and Virtual Environments, pp. 263-273, 1996.

[17] N. Sugano, H. Kato, and K. Tachibana, "The Effects of Shadow Representation of Virtual Objects in Augmented Reality". ISMAR, IEEE Computer Society, Washington-USA, pp. 76-83, 2003.

[18] J. Ferwerda, S. Pattaniak, P. Shirley and D. Greenberg, "A Model of Visual Masking for Computer Graphics". SIGGRAPH, New York-USA, pp. 143-152, 1997.

[19] N. Hoffman, D. Baker, and J. Kautz, "Physically based reflectance for games". International Conference on Computer Graphics and Interactive Techniques, ACM SIGGRAPH 2006 Courses, 2006.

[20] J. Kautz, "Hardware Lighting and Shading: A Survey". Computers Graphics Forum 23, pp. 85-112, 2004.

[21] E. P Lafortune, S. Foo, K. E. Torrance, and D. P. Greenberg, "Non-linear approximation of reflectance functions". In Proceedings of the 24th Annual Conference on Computer Graphics and interactive Techniques International Conference on Computer Graphics and Interactive Techniques, New York, NY, pp. 117-126, 1997.

[22] K. Dana, B. Ginneken, S. Nayar, and J. Koenderink, "Reflectance and Texture of Real-World Surfaces". ACM Transactions on Graphics, 18(1):1-34, January, 1999.

[23] S. C. Foo, “A gonioreflectometer for measuring the bidirectional reflectance of materials for use in illumination computations”. Master's thesis, Cornell University, Ithaca, New York, July 1997.

[24] M. Sattler, R. Sarlette, and R. Klein, “Efficient and Realistic Visualization of Cloth”. In Eurographics Symposium on Rendering, pp. 167-178, June 2003.

[25] G. Müller, J. Meseth, and R. Klein, “Compression and Real-time Rendering of Measured BTFs Using Local PCA”. In Proceedings of Vision, Modeling and Visualization 2003, 2003.

[26] M. Schneider, “Real-Time BTF Rendering”. In Proceedings of CESCG 2004, 2004.

[27] J. Meseth, G. Müller, M. Sattler, and R. Klein, “BTF rendering for virtual environments”. In Proceedings of Virtual Concepts 2003 (Nov), pp. 356-363, 2003.

[28] R. Ramamoorthi and P. Hanrahan, “An efficient representation for irradiance environment maps”. In Proceedings of SIGGRAPH ’01, pp. 497-500, 2001.

[29] G. King, “Chapter 10: Real-Time Computation of Dynamic Irradiance Environment Maps”. GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation, Addison-Wesley Professional, USA, pp. 167-176, 2005.

[30] D. K. McAllister, A. Lastra, and W. Heidrich, “Efficient rendering of spatial bidirectional reflectance distribution functions”. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware ’02, Saarbrucken, Germany, pp. 171-178, Eurographics Association, 2002.

[31] M. Knecht, “State of the Art Report on Ambient Occlusion”. Vienna Institute of Technology, Technical Report, 2007.

[32] M. Bunnell, “Chapter 14: Dynamic Ambient Occlusion and Indirect Lighting”. GPU Gems 2: Programming Techniques for High-Performance Graphics and General- Purpose Computation, Addison-Wesley Professional, USA, pp. 223-233, 2005.

[33] P. Shanmugam and O. Arikan, “Hardware accelerated ambient occlusion techniques on GPUs”. In Proceedings of the 2007 Symposium on interactive 3D Graphics and Games, Seattle, WA, pp. 73-80, 2007.

[34] L. Bavoil, M. Sainz, and R. Dimitrov, “Image-space horizon-based ambient occlusion”. SIGGRAPH '08: ACM SIGGRAPH 2008 talks, pp. 1-1, 2008.

[35] K. Zhou, Y. Hu, S. Lin, B. Gou, and Heung-Yeung Shum, "Precomputed shadow fields for dynamic scenes". ACM Trans. Graph, v.24, n.3, pp. 1196-1201, 2005.

[36] N. Tamura, H. Johan, Bing-Yu Chen, and T. Nishita, "A Practical and Fast Rendering Algorithm for Dynamic Scenes Using Adaptive Shadow Fields". The Visual Computer Journal (Proc of PG2006), vol. 22, n. 9-11,pp. 702-712, 2006.

[37] N. Tamura, H. Johan, and T. Nishita, "Deferred shadowing for real-time rendering of dynamic scenes under environment illumination: Image, Colour and Illumination in Animation". Comput. Animat. Virtual Worlds, vol. 16, n. 3-4, pp. 475-486, 2005.

[38] G. Spencer, P. Shirley, K. Zimmerman, and D. P. Greenberg, "Physically-based glare effects for digital images". In S. G. Mair and R. Cook, Eds, Proceedings of SIGGRAPH '95 Annual Conference on Computer Graphics and interactive Techniques, pp. 325-334, 1995.

[39] M. Kakimoto, K. Matsuoka, T. Nishita, T. Naemura, and H. Harashima, "Glare Generation Based on Wave Optics". Computer Graphics and Applications, IEEE Computer Society, Washington-USA, pp. 133-142, 2004.

[40] M. Kawase, "Home Page – Masaki Kawase," May 1999. [Online]. Available: http://www.daionet.gr.jp/~masa [Accessed: Jan. 29, 2010].

[41] C. B. Madsen and R. Laursen, "A scalable gpu based approach to shading and shadowing for photorealistic real-time augmented reality". In Proceedings of International Conference on Graphics Theory and Applications '07, Barcelona, Spain, pp. 252-261, 2007.

[42] C. Luksch, "Realtime HDR Rendering". Graduation Project. Institute of Computer Graphics and Algorithms, TU Vienna, 2007.

[43] OGRE 3D, "OGRE 3D: Open source graphics engine". [Online]. Available: http://www.ogre3d.org [Accessed: Sep. 8, 2009].

[44] E. Reinhard, M. Stark, P. Shirley, and J. Ferwerda, "Photographic tone reproduction for digital images". In ACM Transactions on Graphics, 21:267-276, 2002.

[45] M. Haller, S. Drab, and W. Hartmann, "A Real-Time Shadow Approach for an Augmented Reality Application Using Shadow Volumes". In Proceedings of the ACM symposium on Virtual reality software and technology, Osaka, Japan, pp. 56-65, 2003.

[46] S. Gibson, J. Cook, T. Howard, and R. Hubbold, "Rapid shadow generation in real-world lighting environments". In Proceedings of the 14th Eurographics Workshop on Rendering, ACM International Conference Proceeding Series, vol. 44, pp. 219-229, 2003.

[47] 3D Studio Max, "Autodesk 3D Studio Max home page". [Online]. Available: http://usa.autodesk.com/adsk/servlet/pc/index?siteID=123112&id=13567410 [Accessed: Dec. 3, 2009].

[48] Maya, "Autodesk Maya home page". [Online]. Available: http://usa.autodesk.com/adsk/servlet/pc/index?siteID=123112&id=13577897 [Accessed: Dec. 3, 2009].

[49] Blender, "Blender home page". [Online]. Available: http://www.blender.org/ [Accessed: Dec. 3, 2009].

[50] Ogre Material Editor, "Ogre Material Editor Forum Topic". [Online]. Available: http://www.ogre3d.org/forums/viewtopic.php?f=11&t=48774 [Accessed: Nov. 30, 2009].

[51] oFusion kit, "oFusion technologies home page". [Online]. Available: http://www.ofusiontechnologies.com/index.html [Accessed: Dec. 3, 2009].

[52] Ogitor Scene Builder, "Ogitor Scene Builder home page". [Online]. Available: http://www.ogitor.org/ [Accessed: Dec. 2, 2009].

[53] Qt, "Qt SDK home page". [Online]. Available: http://qt.nokia.com/products [Accessed: Dec. 3, 2009].

[54] Virtual Reality and Multimedia Research Group – GRVM, "GRVM Home Page". [Online]. Available: https://www.gprt.ufpe.br/grvm/ [Accessed: Jan. 24, 2010].

[55] S. A. Pessoa, "Um Pipeline para Renderização Fotorrealística em Aplicações de RA". Master's dissertation. Federal University of Pernambuco – Informatics Center, Recife, March 2010. To be presented.

[56] ARToolKitPlus, "ARToolKitPlus Home Page". [Online]. Available: http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php [Accessed: Jan. 21, 2010].

[57] wxWidgets Cross-Platform GUI Library, "wxWidgets Home Page". [Online]. Available: http://www.wxwidgets.org/ [Accessed: Jan. 24, 2010].

[58] LEXIExporter, "OGRE Exporters Wiki Page". [Online]. Available: http://www.ogre3d.org/wiki/index.php/LEXIExporter [Accessed: Jan. 24, 2010].

[59] OGRE Manual, “OGRE Manual – Table of Contents”. [Online]. Available: http://www.ogre3d.org/docs/manual/manual_toc.html [Accessed: Feb. 25, 2010].

Dissertação de Mestrado apresentada por **Guilherme de Sousa Moura** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**An Authoring Tool of Photo-Realistic Virtual Objects for Augmented Reality**", orientada pela **Profa. Judith Kelner** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Silvio de Barros Melo
Centro de Informática / UFPE

Prof. Alberto Barbosa Raposo
Departamento de Informática / PUC-RJ

Profa. Judith Kelner
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 18 de fevereiro de 2010.

Nelson Souto Rosa

Prof. Nelson Souto Rosa
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.