Computers & Graphics 36 (2012) 50–69

Contents lists available at SciVerse ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Virtual Reality in Brazil 2011

RPR-SORS: Real-time photorealistic rendering of synthetic objects into real scenes

Saulo A. Pessoa [a,*], Guilherme de S. Moura [a], Joao Paulo S. do M. Lima [b], Veronica Teichrieb [b], Judith Kelner [a]

[a] Virtual Reality and Multimedia Research Group, Informatics Center, Federal University of Pernambuco (UFPE), Brazil
[b] Voxar Labs, Informatics Center, Federal University of Pernambuco (UFPE), Brazil

ARTICLE INFO

Article history:
Received 8 June 2011
Received in revised form
29 November 2011
Accepted 5 December 2011
Available online 29 December 2011

Keywords:
Photorealism
Augmented reality
Authoring toolkit
Pipeline
Computer graphics

ABSTRACT

This paper presents a review of the Photorealistic Augmented Reality field and proposes a solution for interactively rendering virtual objects into dynamic real scenes in a photorealistic way. This solution features a rendering pipeline that comprises techniques regarding illumination, reflectance model, shadowing, composition, and camera effects. The techniques are chained in a flexible way, allowing the user to choose which techniques are to be enabled. An environment map generation procedure was developed and allows virtual objects to exhibit coherent effects such as color bleeding and specular reflection, even when the real objects are moved. The range of materials that can be rendered was widened by extending Lafortune's Spatial BRDF. The implemented infrastructure is offered as an authoring toolkit that consists of an API and a material editor tool. The aim of this authoring toolkit is to increase development productivity of Photorealistic Augmented Reality applications. The proposed solution was evaluated by taking into account visual and performance metrics. It allowed consistent rendering of dynamic scenes and photorealistic materials. The frame rate obtained was suitable to Augmented Reality applications when there were few virtual objects in the scene.

1. Introduction

Augmented Reality (AR) aims to display virtual objects registered with the environment in real-time. In many AR applications (such as games [1], cultural heritage [2], and interior design [3]), it is desirable that the virtual objects appear as real as possible. Through the object's appearance it is possible to deduce many of its properties (material, value, age, etc.), and therefore, the realism plays an important role in Computer Graphics [4]. This problem can be named Real-Time Photorealistic Rendering of Synthetic Objects into Real Scenes (RPR-SORS). In order to tackle this, the application has to cope with features such as coherent illumination, shadows and occlusions.

The goal of this paper is twofold. First, it surveys existing photorealistic AR initiatives. On the other hand, it describes an authoring toolkit for photorealistic AR conceived by the authors, comprising aspects such as illumination, reflectance model, shadowing, composition, and camera effects. The toolkit is named RPR-SORS and consists of an Application Programming Interface (API) and a material editor tool.

The main contributions of this work are: (1) a literature review of the area of photorealistic AR; (2) selection, chaining and implementation of techniques inserted into a photorealistic rendering pipeline for AR systems; (3) punctual extensions to Lafortune's Bidirectional Reflectance Distribution Function (BRDF) and the environment map generation step of the pipeline; and (4) an authoring toolkit that supports the development of photorealistic AR applications. Some previous publications have approached these contributions before [5–8]. However, this paper provides more in-depth details about how the chosen techniques were implemented.

This paper is organized as follows. Section 2 presents some works related to photorealistic AR. Section 3 presents an overview of the developed authoring toolkit. Section 4 details the rendering techniques implemented in the pipeline, emphasizing the contributions made by the current work. While Section 5 explains the RPR-SORS API, Section 6 is focused on the RPR-SORS Material Editor. Section 7 discusses the results obtained regarding visual appearance and performance, as well as an experimental application developed with the toolkit. Section 8 presents a user evaluation. Finally, Section 9 draws some conclusions and future work.

2. State of the art

Despite being a recent research topic, there is an extensive amount of published works about photorealistic AR. The majority of these publications were restricted to a specific problem, for instance, occlusion, illumination, or shadowing. Fournier et al. [9] were one of the first to research about this subject, proposing a

* Corresponding author.
E-mail addresses: sap@cin.ufpe.br, saulopessoa@gmail.com (S.A. Pessoa).

0097-8493/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2011.12.003

method to approximate the reflectance, geometry, and light sources from the real world to coherently insert virtual objects in the scene. In their work, the reflectances of the objects' surfaces were estimated directly from the video stream, while the geometry and light sources had to be manually approximated. Afterwards, this information was used in a classic Radiosity algorithm to compute the illumination (in this case the light sources were considered emissive surfaces). Finally, virtual objects were rendered using a Ray Casting technique. Considering the mentioned work as pioneer, it has some serious problems. First, the required execution time reached 10 min per frame, which excludes its use in interactive applications. Another issue is the time required to manually approximate the real geometry and illumination, limiting its use in many different scenarios.

Later, Drettakis et al. [10] proposed some improvements to the work of Fournier et al. In order to speed up the real scene modeling and camera calibration, a semi-automatic vision-based technique was used. The time required for the scene rendering was also reduced through the use of a hierarchical and incrementally updated Radiosity technique. Besides these improvements, they have also increased the field of view of the captured scene through the use of mosaics, which improved the illumination calculation. Loscos et al. [11] proposed other improvements, where they implemented the removal of real objects, modification of real light sources, and insertion of new virtual light sources. Their work was based on a set of photos taken on different light conditions (manually adjusted by the user).

Although the previously mentioned works have used methods to estimate the illumination and geometry of the real scene, the process still required a considered amount of manual work. Sato et al. [12] developed a way to acquire such information automatically. The geometry was built from a pair of omnidirectional images generated by a fisheye lens. The radiance was estimated from a sequence of images in the same format, captured with different exposure times. The advantage of using omnidirectional images is a wider field of view (compared to [10]), which contributed to better illumination estimation. However, the acquired geometry was imprecise and contained only information about the distant scene. This way, in order to coherently insert shadows, the local scene still had to be manually modeled.

Despite the last three mentioned works proposed new features and optimizations, they did not achieve interactive frame rates. While Refs. [10,11] spent seconds to generate a single frame, Ref. [12] did not expose the results obtained. However, since a Ray Casting technique was used in the latter, it was expected that it did not achieve interactive rates either. In order to overcome this problem, Loscos et al. [13] implemented a solution on GPU for better performance. Besides that, they also refined their solution to use just one image for illumination calculation.

Debevec and Malik were pioneers using image-based lighting (IBL) in virtual scenes, though its first use was not directly related with AR. First, they proposed a method to precisely recover real world radiance from usual low dynamic range (LDR) photos [14]. Using this approach, the photos were combined into a single high dynamic range (HDR) image, called radiance map. Afterwards, this map was used with a global illumination algorithm to illuminate virtual objects inserted in a real environment [15]. To perform correct illumination, the environment was divided into three components: the distant scene, which works as the primary light source; the local scene, which is used to capture the changes caused by virtual objects insertion; and the virtual scene, which is the scene augmentation. Debevec also proposed a Differential Rendering method to compose the real and virtual worlds. With this method, the changes caused by virtual objects insertion could be correctly handled in the final composition. This solution generated good visual results, but did not achieve interactive rates.

Although Debevec did not directly approach the problem of occlusion, other researchers had published some solutions regarding this topic. Breen et al. [16] developed techniques to interactively generate occlusion and physics simulation between the objects. The presented techniques could be classified either as model-based or as depth map-based techniques. They were implemented and evaluated in an AR system that used the GPU in order to achieve interactive rates. Although the solution is designed to AR systems, it is not clear about the possibility of generating the depth maps interactively, thus the camera must remain static. Wloka and Anderson [17] also developed occlusion treatment for an AR system. They proposed an algorithm that receives as input a pair of stereoscopic images and generates the depth maps. This algorithm almost reached interactive rates, however, the visual results obtained were imprecise, and the virtual objects blinked due to failures on occlusion tests.

Given the importance of the knowledge about real world illumination to achieve a natural merge between virtual and real worlds, many solutions were developed around this topic. One way to approximate this illumination is using a set of point light sources. Stauder [18] proposed an automatic method to identify the real light source and illuminate the virtual objects accordingly. Such method consists of: detecting and segmenting a determined real object from a set of images from the real environment; generating an ellipsoidal wireframe around the segmented object; and estimating the light source parameters in order to render the virtual objects. Although it is automatic, Stauder's method has a series of limitations. Scenes must be simple, animated, containing primarily rigid objects, with a single direct light source, and the segmented object cannot be occluded. In addition, because it is based on LDR images, the generated illumination parameters are imprecise.

On the other hand, Kanbara and Yokoya [19] managed to interactively approximate the illumination of multiple light sources and on the exact spot where the virtual object is inserted. In order to accomplish that, a reflective sphere was placed above the marker associated to the virtual object and, by analyzing the region of the image where the sphere is, they could estimate color and direction of real light sources. In order to minimize the problems of using LDR images, the sphere was painted in black, thus reflecting only the light spots. The limitation of this approach is that it cannot handle indirect illumination, and even direct illumination is imprecise because of the small size of the sphere in the image (20×20 pixels).

Grosch et al. [20] proposed a more precise approach based on a set of images captured in the real environment to perform a goniometric reconstruction of a single light source. This type of reconstruction is more precise because it captures the radiance in various points of the scene. Using the reconstructed light source and an estimated diffuse radiance, virtual objects were seamlessly inserted in the scene, though the solution did not achieve interactive rates. Wang and Samaras [21] went a step further and estimated multiple light sources with a single image. This approach was based on shading and shadowing information. It is important to highlight that the real geometry was previously modeled and considered with Lambertian reflectance. Finally, using the estimated illumination, it was possible to coherently insert virtual objects in the real scene, and in addition virtually relight it.

Another common approach to deal with global illumination is using environment maps, which has the advantage to naturally capture indirect illumination and is supported by graphics hardware. Based on Debevec's approach [15], Gibson and Murta [22] proposed a solution based on environment maps to interactively insert virtual objects into real images. Objects were illuminated by a set of pre-filtered HDR spherical maps. Shadows from real light sources were also added using a Shadow Mapping technique, which had its parameters configured from a reference image generated via Ray Tracing.

Later, Gibson et al. [3] proposed improvements to the illumination and shadowing calculations, which were implemented in an interior design application. Although the solution still used environment maps to simulate specular reflections, the use of irradiance volumes allowed the assumption that the light sources were not infinitely distant, thus it could deal with more general lighting conditions. In order to rapidly discover which light sources are occluded (causing shadows), a hierarchical data structure was created. Spherical Harmonics were used to reduce the processing and storage size of the irradiance volume. Although this solution proposed important improvements, it did not support moving the real objects in the scene and, except for shadows, virtual objects did not cause any changes in the local scene.

Another solution based on irradiance volume was published later by Grosch et al. [23]. The case study of such an approach consists of a real Cornell box (where virtual objects should be inserted) positioned in a place with direct sun light. The external illumination was captured interactively by an HDR camera with fisheye lens, and this information was used to simulate indirect lighting within the box. In order to synchronize the indirect illumination with external light conditions, a special type of irradiance volume that supports interactive updates had to be used. In addition, shadows generated by the outdoor illumination were also simulated using Importance Sampling. The limitations of this solution come from the restrictions of irradiance volumes. Consequently, the indirect illumination was not changed when virtual objects were inserted in the box.

Agusanto et al. [24] also used pre-filtered HDR environment maps to deal with illumination on AR systems. These maps had different blur levels in order to represent materials with various levels of glossiness. On the other hand, Heymann et al. [25] filtered the environment maps on the fly. Like Kanbara and Yokoya [19], they placed a reflective sphere above the virtual object marker. However, in this case, they used a larger and perfectly reflective sphere in order to achieve better results. The surrounding scene observed over the sphere was taken as a spherical map, which was filtered every frame in order to simulate indirect illumination. However, it is worth to highlight that this solution did not use Spherical Harmonics for filtering, which could improve both the quality and speed of the solution.

Environment maps were also the path chosen by Hughes et al. [26] to simulate realistic illumination. In order to quickly render the virtual objects, the illumination from the scene was precomputed for each vertex. This solution also supported virtual point lights to be added in the scene, and both virtual and real objects responded coherently to the changes. One issue detected in this solution was the inconsistency on dynamic scenes, since the illumination from the scene was precomputed.

Nishina et al. [27] proposed a method to interactively capture HDR environment maps without using an HDR camera. The map was adaptively generated depending on the virtual object's albedo. The adaptation was performed estimating the radiance necessary to render the object in the next frame. Then, the HDR map was generated varying the amount of images captured and their exposing levels. Finally, after the virtual object is rendered using the generated map, the resulting image was tone mapped to a lower dynamic range according to the exposure time of the real camera.

A specific pipeline that deals with external scenes, supporting daylight and dynamic illumination, was proposed by Jensen et al. [28]. This pipeline consists of an offline and an online step. Using the HDR environment map and the real scene geometry as inputs, the offline step generates environment maps that describe: the albedo, accessibility, and orientation of surfaces in the real scene. The online step receives these maps as input together with the image captured by the camera. Using this approach, it was possible to estimate a primary directional light source (sun), and an ambient component (light scattered in the clouds). These estimations were used to illuminate and cast shadows on the virtual objects. It is worth mentioning that the used shadowing approach took into account the estimated light and the scene geometry in order to not generate shadow where it already exists.

Feng [29] interactively estimated the real illumination by using two spheres with Lambertian properties. First, the two spheres were segmented from the captured image. Then, using the segmented region, the location and intensity of the primary light source was estimated, as well as an ambient component. Using this information, a light source was created to illuminate the virtual objects. However, this approach is limited to scenes with a single primary light source.

Madsen and Nielsen [30] also interactively estimated the real environment illumination, using a specific approach for external scenarios with hard shadows. No intrusive object was necessary, such as mirrored spheres. Their proposed solution consists of: analyzing the captured image to identify shadow regions; determining the ratio of the sun irradiance to the sky irradiance; then, converting this information to radiance levels. The sun position is obtained based on the date, hour, and geographic position (using a compass and a GPS). Using this approach, it was possible to build an illumination model that was applied to the virtual objects.

Shadows also have an important role for photorealistic AR. They give strong cues about object positioning making it easier for the user to understand how the scene is organized. Haller et al. [31] adapted a Shadow Volume technique and used it on an AR solution. In order to guarantee the correct shape of the projected shadows, phantom objects had to be previously modeled. These objects also provided correct occlusion treatment. Both the shadow generation and the occlusion were implemented on GPU (using, respectively, the stencil and depth buffers). Although this solution achieved interactive rates, it presented some problems. Besides the light source parameters, which had to be manually set by the user, there is no concern about inserting shadows or not, possibly causing virtual shadows to be inserted where there were already real shadows.

Madsen [32] based his approach on real shadows in order to add virtual shadows. He segmented and classified the image captured from the real environment into three regions: directly illuminated, penumbra, and completely shadowed. Using these sets, it was possible to coherently insert virtual shadows. This solution did not require the use of any intrusive object.

Kakuta et al. [2] proposed a different approach trying to improve performance. They used a set of base images to approximate soft shadows generated by the real illumination. These images were previously rendered and later composed in order to generate shadows. This approach has the disadvantage of being applicable only to static scenes.

Jacobs et al. [33] also presented a solution for interactive rendering of virtual shadows consistent with the real ones. This solution consists of three steps: shadow detection, which is performed using the texture information and an initial estimate of the shadowed region; shadow projection, in order to avoid the insertion of virtual shadows where real ones already exist; and shadow generation, which is performed using a technique based on Shadow Volumes. This solution requires that the real world geometry and light source information are manually provided to the system. Besides that, it supports a single light source to project hard shadows.

Approaches based on environment maps were also used to generate shadows. Supan et al. [34] used the GPU to interactively generate them. Similarly to [25], this approach used a reflective sphere positioned on the real scene to capture an environment map. This map was later sub-sampled in order to generate up to 64 light sources, which were used by a Shadow Mapping technique.

On the other hand, Madsen and Laursen [35] pre-captured an HDR environment map and segmented it using a Median Cut algorithm. Each segment represented a light source and its intensity. In order to avoid duplicated shadows, they interactively estimated the scene's diffuse albedo and reilluminated the areas of interest before adding virtual shadows. However, this approach is not aware of changes in the real world illumination.

Summarizing, the approaches presented by [34,35] used multiple light sources in order to obtain a softer shadowing. These approaches also used environment maps in order to illuminate virtual objects.

Photorealism was also incorporated into Markerless AR systems. Frahm et al. [36] estimated camera pose using a Structure from Motion (SfM) technique. Light sources were identified observing the saturated regions of an image captured by a second camera with fisheye lens. Using these sources, they projected shadows (using Shadow Mapping) and illuminated virtual objects.

Bimber et al. [37] proposed an approach for Optical See-through systems. This approach used a set of cameras and projectors to illuminate and capture the diffuse reflectance of the real environment. The scene geometry was previously modeled and provided to the system. For the indirect illumination effects (such as color bleeding), an offline multi-step radiosity algorithm was used. On the other hand, direct illumination effects used GPU-based techniques. In addition to these effects, stereoscopic rendering of the virtual objects was also supported. Although they supported indirect illumination effects, they did not execute it interactively, thus the scene had to be static to preserve consistency.

Bradley and Roth [38] proposed a method to show real world illumination in dynamic augmented scenes using only one camera. The scene consisted of a piece of cloth that was augmented with a textured virtual plane (cloth deformation was considered). This way, it was possible to modify the real piece of cloth appearance. The virtual plane illumination was computed using a simple approach that only required the intensity values of the real image. It was not necessary for extra processing such as 3D reconstruction or the creation of virtual lights. However, this approach presented some failures in the virtual plane (caused by tracking errors performed by ARToolKit) and shadowing errors. In order to solve these problems, Bradley et al. [39] proposed new improvements.

Besides traditional images with restricted field of view, panoramic images were also coherently augmented. Grosch [40] proposed an interactive method to add virtual objects and light sources to omnidirectional images. This method requires only an HDR environment map from the scene of interest. Using this map, scene geometry and reflectance were estimated. The geometry was estimated using box and plane primitives; this way the real scene was restricted to contain only these kinds of shapes. With the reconstructed scene, the user could rotate and move the camera through the augmented scene. The restriction of this method is that it cannot interactively capture the omnidirectional image, thus the stored image can only be augmented by adding new virtual elements.

Beyond illumination, reflectance estimations, and shadows, other aspects were also considered in order to achieve photorealism in AR. Okumura et al. [41] proposed a method based on the blurring of the real image. This method interactively analyzed the blurring due to depth of field and motion blur, and rendered virtual objects exhibiting the same effects.

Finally, trying to achieve better results, the use of Ray Tracing was proposed by Scheer et al. [42]. This approach used an HDR environment map to capture the real world illumination. This map was interactively captured by an HDR camera, and then filtered (using Spherical Harmonics) generating an irradiance environment map. This last map was used to illuminate virtual objects. Shadows were also considered in this approach. It was achieved by using a reduced number of light sources that were estimated according to their importance. However, this technique did not fulfill the AR performance requirements.

3. RPR-SORS toolkit overview

The toolkit proposed by this paper was named RPR-SORS. It was conceived by considering the needs of both programmers and artists. This way, the toolkit was divided into two major components, an API and an authoring tool, providing appropriate tools for each professional so that they may perform their tasks more easily.

The RPR-SORS API intends to make easier and speed up the coding process of photorealistic applications. This is possible since many of the essential features are accessible to the programmer in a few code lines. On the other hand, the provided authoring tool is a material editor, which uses the RPR-SORS API for photorealistic rendering. It allows designers to easily create photorealistic materials and export them for later use.

4. Implemented techniques

This section presents the pipeline techniques. Besides visual quality and performance, each technique was chosen regarding an aspect in order to achieve photorealism. The regarded aspects were: illumination, surface reflectance, shadowing, composition, and camera effects.

4.1. Illumination

In order to make virtual objects seem like real, they must receive the same illumination present in the real scene. This is achieved through the use of an image based lighting (IBL) technique. This kind of technique stores the illumination into high dynamic range (HDR) images (which may also be called radiance maps) and usually achieves more realistic results. Regarding the AR applications, a real world HDR environment map is captured from the scene of interest so that the virtual objects can be properly illuminated. Since IBL techniques are based on real images, it can naturally handle both direct and indirect illumination (differently from techniques based only on primary point light sources).

Due to the difficulty involved in interactively capturing HDR environment maps from real scenes, this work used pre-captured maps. However, in order to simulate the influence of the surrounding scene in virtual objects, a new environment map is captured from the center of each virtual object. The idea is to apply the pre-captured map to a skybox (see Fig. 1a) and, from the center of each virtual object, capture their respective environment maps (Fig. 1b). In order to avoid self-illumination artifacts, each virtual object must be hidden during its environment map capture. At the end of this process each virtual object will have its own environment map.

As illustrated in Fig. 1b, the map captured from the center of each virtual object is in the cubic format. Six rendering passes were used to create it (by using Geometry Shaders, one pass would be enough [43]). The cubic format was chosen since it is

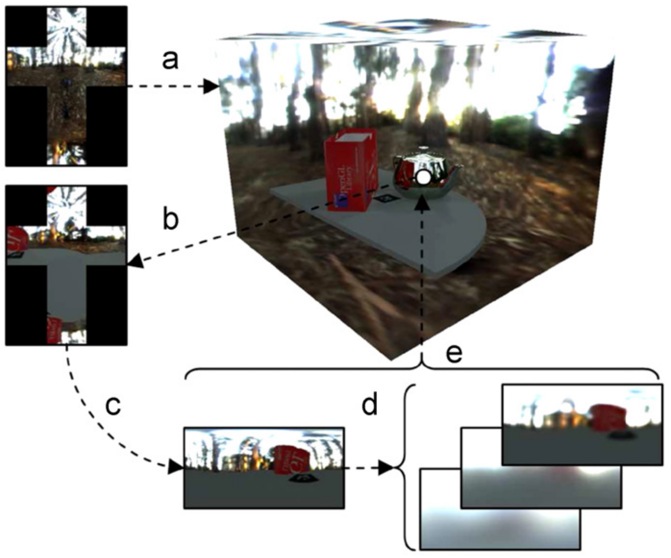


Fig. 1. Idea employed to illuminate virtual objects. In (a), the pre-captured environment map is applied to the skybox, in (b), a new environment map is captured from the teapot center, in (c), the new captured map is converted to the latitude–longitude format (the specular map), in (d), the converted map is filtered generating three new maps (diffuse, glossy1, and glossy2 maps), and in (e), the four latitude–longitude maps are used to illuminate the teapot.

natively supported by modern GPUs. However, the latitude–longitude format was preferred to do the illumination of the objects. Since the latter can be stored in conventional 2D textures, it is cheaper to be handled. Thus, the cubic environment map is converted to the latitude–longitude format prior to the illumination process (Fig. 1c). This converted map will be also called as the specular map.

The specular environment map is enough to achieve only mirrored materials. In order to achieve diffuse and glossy materials, three new latitude–longitude maps are generated with different levels of blurring (Fig. 1d). These new maps are generated from the specular map and are called diffuse, glossy1, and glossy2 (from the most blurred to the least one). At the end, the resultant four latitude–longitude maps are used to illuminate the virtual object (Fig. 1e). If a diffuse material must be rendered, then the most blurred map is used (the diffuse map), on the other hand, a mirrored object would use the most sharpened map (the specular map). Intermediate materials are achieved interpolating two of the four maps. As these maps are generated for every frame, interactive reflections over the virtual objects can be achieved.

4.1.1. *Diffuse and glossy maps generation*

There are two ways to filter environment maps. The first and simplest way is to apply a low-pass filter through a convolution operator. The second way is more complex and is performed in a frequency domain. Since each approach has its own advantages and disadvantages, both were used (while the former is expensive to produce large blurred areas, the latter is expensive to produce sharpened results). Considering both characteristics, the convolution method was used to generate the glossy2 map (since it is the second most sharpened map), and the frequency domain method was used to generate the glossy1 and diffuse maps (since they are the most blurred ones).

The convolution process used to generate the glossy2 maps is executed regarding the domain of the image in question, i.e., the set of every 3D direction. This means that the input image (the specular map) must be sampled based on 3D directions, not on 2D coordinates as usual. The following expression generates the glossy2 map:

$$f(\kappa)=\sum_{\Omega}L_s(\omega)\max((\kappa\cdot\omega)^{128},0), \qquad (1)$$

where $f(\kappa)$ is the pixel with direction κ on the glossy2 map, Ω is the set of every direction ω in the 3D space, $L_s(\omega)$ is the pixel with direction ω on the specular map, and $(\kappa\cdot\omega)^{128}$ is the filtering function. The max function guarantees that negative values are ignored.

Since the diffuse and the glossy1 maps tend to be very blurred, they can be well and efficiently made in the frequency domain. This is achieved through the use of the Spherical Harmonics. According to Ramamoorthi and Hanrahan [44], nine Spherical Harmonics coefficients (degree 2) are enough to generate a diffuse map. By extending this idea, this paper also generated the glossy1 map, but using 256 coefficients (degree 15). The implemented Spherical Harmonics filtering consists of three steps: the transform (that generates 256 coefficients), the filtering (that is done just by multiplying the coefficients by the filtering values) and the inverse transform (that obtains the filtered map). In order to achieve a good performance, this filtering process is carried out in the GPU through the use of full-screen quad render passes, similar to King [45]. However, the way that the input map is sampled was changed in order to make easier the transformation pass. King samples the input map texel by texel. The main difficulty encountered by such a sampling scheme is that each sample will be subtended by a different solid angle, which consequently implies in using a different weight for each sample. In order to avoid such difficulties and generalize the sampling operation, a uniform spaced sampling scheme was used. The uniform sampling directions are generated by the expression:

$$(x,y)\rightarrow(2\arccos(\sqrt{1-x}),2\pi y)\rightarrow(\theta,\phi), \qquad (2)$$

where (x,y) are 2D points evenly spaced over a unit square and (θ,ϕ) is the desired direction expressed in spherical coordinates. This process generates directions similar to the intersection points between parallel and meridian lines in the geographic coordinate system, but with a sparser distribution of directions close to the poles (see Fig. 2b). As the proposed sampling scheme does not guarantee that every texel on the input map will be sampled, some aliasing may occur depending on the amount of samples taken. In order to minimize this problem, this time the glossy2 map is used as input instead of the specular map. Since the glossy2 map is smoother than the specular, it is less prone to aliasing during its sampling. An alternative way to

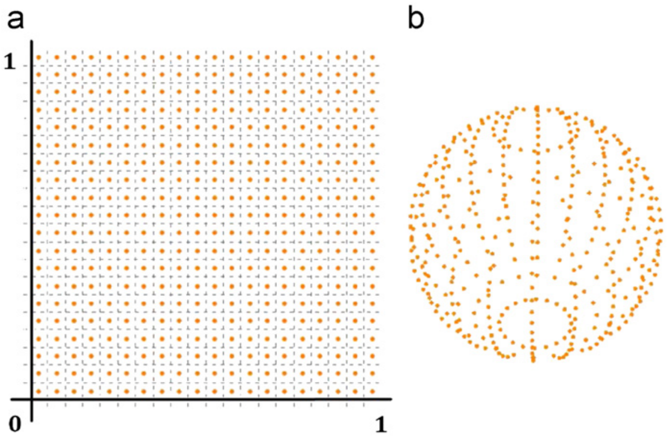


Fig. 2. Mapping from 2D points to 3D evenly spaced directions. In (a), 400 evenly spaced 2D points with their respective bounding regions and the resultant 3D directions in (b).

generate evenly spaced directions could be using an isocube representation [46], however, without any significant visual quality gain and performance penalties due to its more complex expression.

The first step of the filtering process is the Spherical Harmonics transform, which needs to be performed only once per frame in order to generate both the glossy1 and diffuse maps:

$$L_l^m = \sum_{\Omega} L_g(\omega) y_l^m(\omega) \frac{4\pi}{q}, \tag{3}$$

where L_l^m is a coefficient within the range $0 \le l \le 15$ (since only the first 256 coefficients are generated) and $-l \le m \le l$. ω is a direction used to sample both the input map L_g (the glossy2 map) and the Spherical Harmonics basis functions y_l^m. $4\pi/q$ is the constant weight where q is the amount of samples. In order to boost the performance, the Spherical Harmonics basis functions were precomputed using Wolfram Mathematica and stored into a texture. Fig. 3a shows the tile arrangement proposed to store the basis functions. This arrangement stores each degree in a consistent way, allowing the functions to be later retrieved without the need of translation tables. At the end of this step, 256 coefficients are generated into a texture according to the same arrangement (see Fig. 3b).

Next, the filtering and inverse transform are performed together by the following expression:

$$\tilde{L}(\omega) = \sum_{l} \sum_{m=-l}^{l} A_l L_l^m y_l^m(\omega), \tag{4}$$

where $\tilde{L}(\omega)$ is the filtered map, A_l is a filtering value of degree l and y_l^m are the same previously precomputed basis functions. The diffuse map is generated with $0 \le l \le 2$ and the glossy1 map with $0 \le l \le 15$. While the diffuse filtering values were published by Ramamoorthi and Hanrahan in [44], the glossy1 filtering values had to be generated in this work. They were numerically obtained based on the Ramamoorthi and Hanrahan work. Table 1 shows both the diffuse and glossy1 filtering values.

Another idea employed to boost performance was using different sizes for each generated map. As the diffuse and the glossy1 maps are more blurred than the glossy2 and the specular maps, the former may have smaller sizes. Although this reduction boosts performance, it also makes texels' bounds clearly visible. Bilinear texture filtering was used to correct this drawback. Fig. 4 shows the generated maps.

As an alternative to Spherical Harmonics, a Summed-Area Table (SAT) technique [48] was also used in an attempt to enhance the overall performance. However, because of the overhead needed

Table 1
Filtering values used to generate diffuse and glossy1 maps.

	Diffuse	Glossy1
A_0	3.141593	0.694003
A_1	2.094395	0.624635
A_2	0.785398	0.503142
A_3	–	0.362063
A_4	–	0.228609
A_5	–	0.124830
A_6	–	0.055127
A_7	–	0.015344
A_8	–	−0.004141
A_9	–	−0.012064
A_{10}	–	−0.015605
A_{11}	–	−0.018064
A_{12}	–	−0.020676
A_{13}	–	−0.023416
A_{14}	–	−0.026419
A_{15}	–	−0.029469

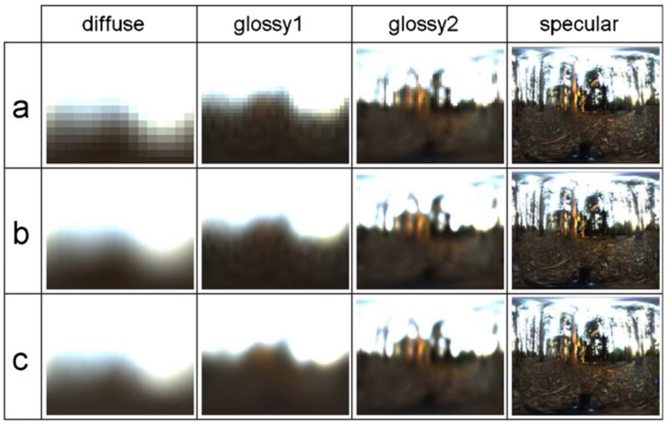


Fig. 4. In (a), the generated diffuse, glossy1, glossy2, and specular maps. Their sizes are respectively 16 × 16, 32 × 32, 64 × 64, and 128 × 128. In (b), the resultant images when maps in (a) are bilinear sampled. In (c), reference maps generated with HDR Shop [47].

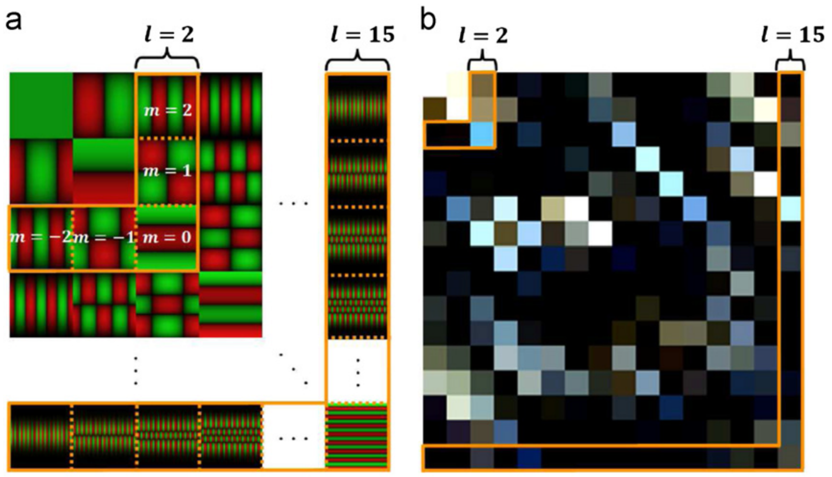


Fig. 3. In (a), tile organization used to store the precomputed basis functions. Each function is in the latitude–longitude format and their values range from −1 (red) to 1 (green). In (b), the generated coefficients are organized in a similar way. The functions with degree $l=2$ and $l=15$ are highlighted with their respective coefficients. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

to generate the table at every frame and the undesirable artifacts generated by the box filtering, this approach was abandoned.

4.2. Reflectance model

Before evaluating the reflectance model, a normal mapping technique is performed in order to simulate surface roughness without the cost of adding new geometry. This technique is illustrated in Fig. 5 and uses normal vectors stored in a texture map (usually called normal map) [49]. The implemented solution features a parameter that controls the contribution of the normal map, resulting in the following expression:

$$N = \text{normalize}(N_g + k(N_m - N_g)), \tag{5}$$

where N_g is the original normal in world space, k is the parameter used to control the normal map contribution, and N_m is the normal provided by the normal map in world space. The normal vector N is used for the subsequent computing.

Lafortune et al. proposed a compact and general BRDF representation [50]. This representation was later extended by McAllister et al. in order to support spatial variation of surface reflectance properties [51]. This new representation is called Spatial BRDF (SBRDF) and was implemented in GPU. Another feature proposed by McAllister et al. was the use of environment maps as light source. Since interactively filtering environment maps is an expensive task, they used six pre-filtered maps (this work uses only four maps because it is not so expensive to generate and the results were still visually acceptable).

This paper improves the mentioned SBRDF by providing a way to: interactively generate environment maps (explained in Section 4.1), achieve increasing specularity at grazing angles, simulate space-varying direction of anisotropy, and provide an approximation to simulate refractive materials. Although McAllister results suggest support for space-varying direction of anisotropy, the lack of details motivated the creation of an own solution.

The chosen SBRDF has a diffuse and a specular component that can produce faithful materials such as metals, plastics, porcelain, etc. However, since it does not take into account subsurface scattering, it cannot simulate materials such as skin, marble, plant leaves, and wax. The diffuse component has just one term and represents the Lambertian reflection. The specular is a sum of terms where each one represents a lobe. The SBRDF is evaluated in a local coordinate system and its expression is

$$L_r(V) = \rho_d D(N) + \sum_j \rho_{s,j} S(p(V), n_j) g(V). \tag{6}$$

In order to improve its representativeness and performance, the following modifications were made. First an emissive term was inserted so that materials with this kind of property could be simulated. Another modification was the use of just one specular lobe. Since the use of multiple lobes did not significantly improve the visual results (as also mentioned in [51]), this simplification was adopted targeting a better performance. By adopting these modifications, the final expression became

$$L_r(V) = e + \rho_d D(N) + \rho_s S(p(V), n) g(V), \tag{7}$$

where $L_r(V)$ is the radiance reflected to the view direction V. In the diffuse component, ρ_d is the diffuse albedo and $D(N)$ is the irradiance incoming from normal direction N (it is achieved by sampling the diffuse map). In the specular component, ρ_s is the specular albedo and S is a weighted mean of samples taken from two of the four maps. The glossiness parameter n (similar to the Phong exponent) defines the two maps that will be sampled and their respective weights. Fig. 6 exhibits the results obtained for different values of n.

The two chosen maps are sampled according to the peak vector $p(V)$. This vector is explained by McAllister et al. [51] and represents the direction where most of the reflected light (in the view direction V) is coming from. For example, in a Phong like material the peak vector is the vector which respects the law of reflection (incoming and outgoing direction makes the same angles with respect to the surface normal), on the other hand, in a retro-reflective material it is equal to the view vector. The peak vector is evaluated by

$$p(V) = \begin{bmatrix} C_x & 0 & 0 \\ 0 & C_y & 0 \\ 0 & 0 & C_z \end{bmatrix} \cdot \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix}, \tag{8}$$

where V and $p(V)$ are expressed in surface local coordinates. C_x, C_y, and C_z are the parameters that determine the material properties. For example: $C_x = -1$, $C_y = -1$, and $C_z = 1$ produce a Phong like material; $C_x = C_y$ produce isotropic materials; $C_x > 0$ and $C_y > 0$

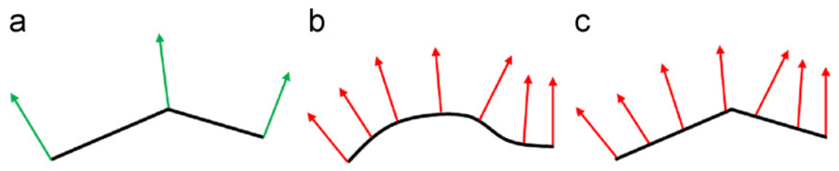


Fig. 5. In (a), original surface and its normals. In (b), desired detailed surface and its normals. In (c), the normals of (b) are mapped to the surface of (a) simulating the detailed appearance.

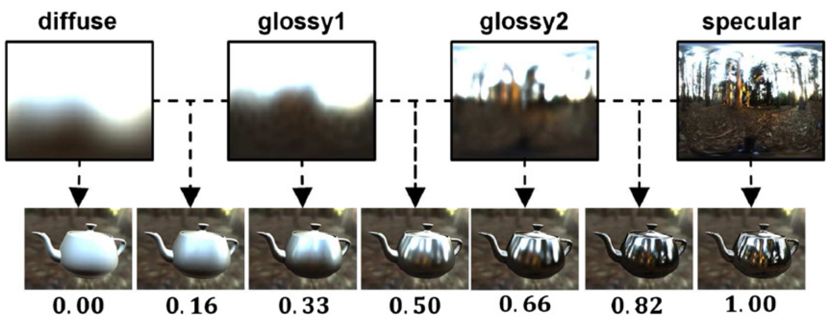


Fig. 6. Resultant specular component for different n values. Observe how these values determine which maps are sampled: for $0.00 \le n \le 0.33$ diffuse and glossy1 maps are sampled; for $0.33 < n \le 0.66$ glossy1 and glossy2; and for $0.66 < n \le 1.00$ glossy2 and specular.

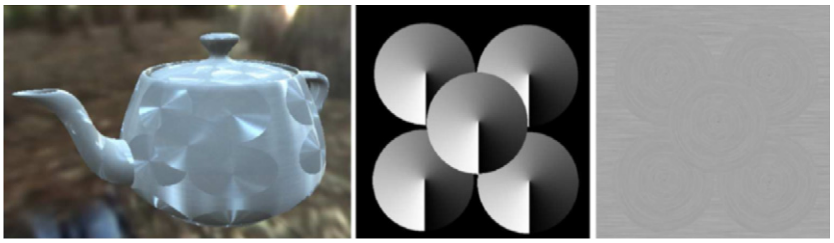


Fig. 7. On the left, the resultant material with space-varying direction of anisotropy. In the middle, the monochromatic texture storing the rotation angles. On the right, the texture used as ρ_d and ρ_s.

produce retro-reflective materials. These parameters can also be measured from real materials [52]. Finally, $g(V)$ is the term that avoids radiance coming from below the surface tangent plane from being included in $L_r(V)$ when low glossiness parameters are used. This last term is given by

$$g(V) = N \cdot p(V). \tag{9}$$

The increasing specularity at grazing angles was achieved through the use of a response function:

$$f(V) = (1-(V \cdot N))^{\varphi} f_w, \tag{10}$$

where $f(V)$ is the contribution ranging in the interval [0,1], φ is the falloff, and f_w is the effect weight. Eq. (10) is then used to generate new parameters for the specular albedo ρ_s, for the glossiness n, and for the irradiance falloff $g(V)$:

$$\rho_s' = \rho_s + (1-\rho_s) f(V), \tag{11}$$

$$n' = n + (1-n) f(V), \tag{12}$$

and

$$g(V)' = g(V) + (1-g(V)) f(V). \tag{13}$$

These new parameters replaced the original ones in order to obtain the final SBRDF:

$$L_r(V) = e + \rho_d D(N) + \rho_s' S(p(V), n') g(V)'. \tag{14}$$

The space-varying direction of anisotropy is achieved by rotating the tangent vector around the normal vector (in order to preserve the orthogonality, the binormal vector must also be rotated). It is done prior to the shading evaluation and according to a monochromatic texture. Since this kind of texture can only store values ranging from zero (black) to one (white), a mapping to the interval $[0, 2\pi]$ is done to achieve a full range rotation. This process ends up generating a new local coordinate system so that the shading will now produce the desired effect. Fig. 7 shows a result with its respective input textures. It is important to mention that the use of bilinear filtering may produce some unpleasant artifacts. It happens mainly on the high gradient areas where the interpolation between different texels produces invalid values (this interpolation is valid only for colors). These artifacts appear as dark or bright points close to the mentioned areas. In order to avoid them, a different interpolation method should be investigated. However, since they can be avoided by disabling bilinear interpolation (although this can produce jagged edges) this work will not go further.

Another implemented feature was an approximation to simulate refractive materials. It was achieved based on the technique presented in [53]. The initial idea was to sample the generated specular map according to refracted vectors. However, this idea was abandoned because the resultant effect showed inconsistency when superimposed onto the background image (the video stream). Sampling the background image achieved more natural results and consequently was adopted. The refraction effect is achieved by first evaluating the refracted vector A accounting: the

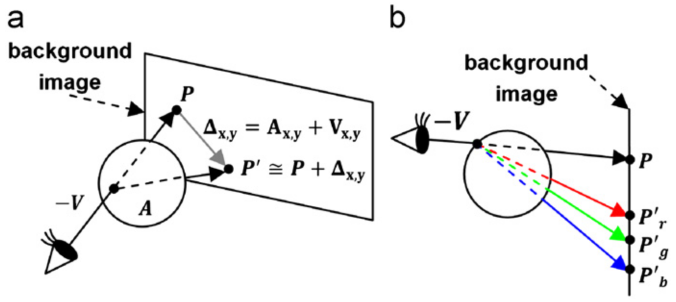


Fig. 8. In (a), the idea employed to achieve the refraction effect. In (b), the idea behind the light dispersion effect.

view vector V, the normal vector N, and a refraction index passed as parameter. This refracted vector could be directly used to sample the environment map. However, it must be translated to a 2D coordinate in order to sample the background image. This 2D coordinate is found by evaluating the difference between A and V regarding only x and y components. This difference is then added to the original point P to find an approximation for the point P'. This last point is where the background image should be sampled in order to achieve the refraction effect (see Fig. 8a). Light dispersion was also simulated. It is achieved by adding a tiny increment to the refraction index for each color component. Therefore a different P' is evaluated for each color component (see Fig. 8b). At the end, the result of the refraction effect is linearly interpolated with the SBRDF result.

4.3. Shadowing

Shadows are fundamental to a correct spatial perception. They provide clues about the shape and the relative placing of the objects. Actually, the lack of shadows may lead to an ambiguous interpretation of the scene. Therefore, shadows were also considered in the pipeline.

Generating shadows according to the illumination stored by environment maps is a complex task. The trivial solution could be considering each map pixel as a light source, although, it would be very expensive. A better solution is to approximate the original illumination by subdividing the map into regions of equal energy. Each region acts as a single light source that represents the average information of the pixels contained by the region (in the proposed pipeline, these will also be called as phantom light sources). After the creation of this approximation, a usual shadowing technique can be more efficiently used.

A Median Cut algorithm [54] is used in order to subdivide the latitude–longitude environment map into regions. This subdivision is done iteratively by generating 2^n regions for n iterations. In the first iteration the whole map is considered as the initial region. So it is split along the longest dimension in the median point (the point that generates regions with equal energy). This process splits one region into two new regions and is iteratively repeated until the desired number of regions is achieved. In order to alleviate the over-representation of the regions near the poles (it is an intrinsic characteristic of the latitude–longitude maps), pixels on the map must be scaled by $\cos(\theta)$. The over-representation should also be taken into account to determine the longest dimension of a region.

After the map subdivision, a centroid and a total intensity are evaluated per region. While the total intensity is obtained by summing its pixels' intensities, the centroid is estimated by a weighted mean of the pixels' positions (in this case the pixels' intensities are used as the weight). Fig. 9 shows a subdivided map with the evaluated elements. After this process each region will have three information: an area (that will correspond to the light source size), a centroid (that will correspond to the light source direction expressed in a spherical coordinate system), and a total intensity (that will correspond to the light source intensity). A 3D position needs also to be generated per region. Since environment maps are supposed to store the illumination coming from an infinite distance, the position information had to be arbitrated based on the light source direction (the region centroid). It is achieved by

$$P = KD, \tag{15}$$

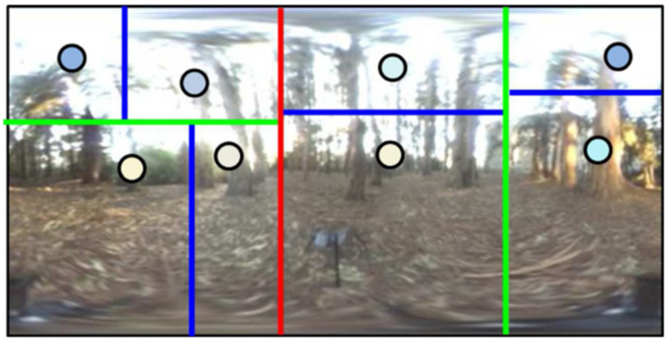


Fig. 9. Regions with their respective centroids and total intensities. The total intensities (represented by the circle internal color) were scaled down in order to not appear white. The red split was created in the first iteration, green in the second, and blue in the third. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

where P is the desired light source position, K is a constant passed as parameter by the user, and D is the light source direction (in this case expressed as a 3D vector).

The chosen shadowing technique was the Percentage-Closer Soft Shadow (PCSS) [55]. It is a Shadow Mapping based technique and was chosen mainly because it can generate penumbrae regions. This feature is important in order to compensate the use of fewer light sources. The penumbrae effect is achieved by filtering the shadow map according to the regions' areas (the light sources size). The filter size is given by

$$w_p = \frac{(d_r - d_o)w_l}{d_o}, \tag{16}$$

where w_p is the filter size, d_r is the distance from the light source to the point being shadowed, d_o is the average distance from the light source to the occluder, and w_l is the light source size.

The crude result of the chosen technique for just one light source is depicted in Fig. 10a. White pixels indicate full visibility from the light source point of view, black ones no visibility, and gray ones intermediate visibility (the penumbrae regions). Since the crude result may exhibit some strange artifacts and an unnatural effect close to the silhouettes, the original visibility result is attenuated according to the light source direction L and the surface normal N:

$$v' = v * \max(L \cdot N, 0), \tag{17}$$

where v' is the attenuated visibility and v is the crude one. The attenuated visibility is exhibited in Fig. 10b.

The final shadowing effect is achieved by subtracting the undue light intensity added to the rendered scene. It is performed according to the attenuated visibilities and the intensities of each light source. In practice, the undue light energy added per light source is

$$(1 - v_i') * L_i, \tag{18}$$

where $1 - v_i'$ is the attenuated visibility of light source i and L_i is the respective intensity. At the end, this result is subtracted from the rendered scene. Fig. 11 illustrates it. Wyman and Hansen proposed an alternative method to generate soft shadows, called Penumbra Maps [56]. However, their technique was discarded because it relies on a silhouette detection algorithm, which is dependant of the object complexity and compromises scalability.

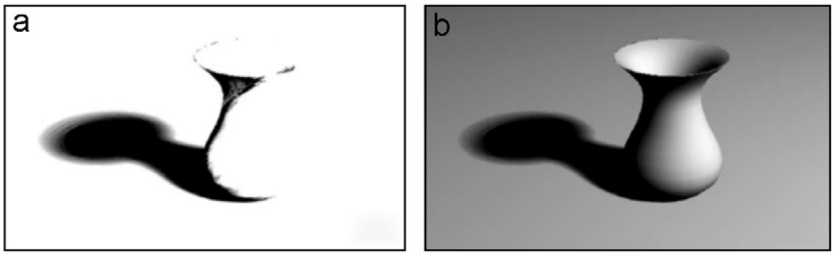


Fig. 10. Result of the shadowing technique for just one light source. In (a), the original visibility result. In (b), the attenuated visibility.

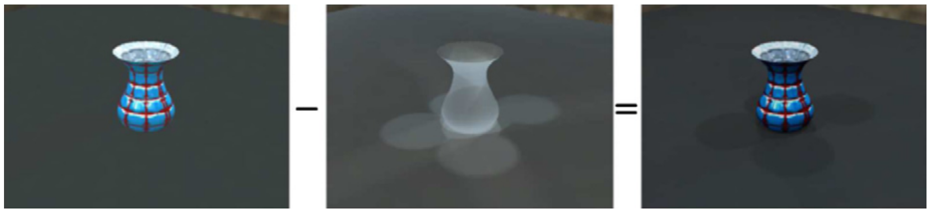


Fig. 11. From left to right: a virtual scene without shadows, the total undue light intensity, the final scene with shadows.

4.4. Composition

The composition of the virtual objects with the real scene is also important for the overall quality of the final result. In order to achieve good results, the visual modifications caused by virtual objects insertion must be taken into account. These modifications may appear as many known effects: shadowing, diffuse indirect lighting, caustic, and occlusion. The idea in the composition stage is to go beyond a simple image superimposition.

Haller et al. proposed a simple and limited approach [31]. It is performed by shading the scene with a black-transparent color blending and can handle only shadows and occlusions. Debevec [15] has proposed a more general and precise approach. This approach is called Differential Rendering (DR) since it is based on the difference between two rendered images. Debevec's approach was chosen to integrate the pipeline.

Debevec first partitioned the scene into three components: the distant, the local, and the virtual scenes. The distant scene is the portion of the real scene that is not influenced by the other components. It just acts as light source and consists of a rough geometric approximation with an HDR environment map applied to it. The local scene represents the portion of the real scene that may be modified after the virtual objects insertion. This component comprehends the vicinity region where the virtual objects are inserted. In order to achieve good visual results, the local scene must be created with fine geometric models (the so-called phantom objects). However, a rough reflectance model is enough. Since environment maps only capture lighting coming from far, the local scene is also important because it acts as a near-field illumination. Finally, the virtual scene comprehends the new elements that will be inserted into the real scene.

The first step of the composition process is the rendering of two images to off-screen buffers. While the first image exhibits only the local scene, the second exhibits both the local and virtual scenes (Fig. 12b and c). Next, considering only the second image, a mask to the virtual scene regions (i.e., the image regions where virtual objects can be seen) is created in the stencil buffer (Fig. 12d). Since virtual scene regions are disjoint from local scene regions, a complementary operation may be used to obtain the mask to the latter. The following step evaluates the difference between the two rendered images (Fig. 12e) and adds it to the background image. In order to regard only the local scene regions, the complementary version of the mask is used. At last, the virtual scene regions are superimposed onto the background image according to the original mask. Fig. 12f exhibits the final composition.

4.5. Camera effects

The last aspect considered in the pipeline concerns the device used to capture the scene, i.e., the camera. Depending on the device's characteristics, many visual effects may arise. Two well known effects are Bloom and Glare. These effects are important because they increase the perception about the areas of the image exhibiting high levels of energy (while the former produces feathers of light around very bright areas, the latter generates radial streaks). Another well known effect is called Exposure Control and arises due to the cameras' ability to control the amount of light reaching the photosensor. Actually, when the scene is over-illuminated, the camera tends to restrict the light entrance. On the other hand, when the scene is poorly illuminated, it tends to make easier the light energy entrance. Since these effects are usually observed in real images, they had also to be considered into the photorealistic pipeline.

It was preferred to apply the three effects in the whole composed image. Since the real background image already exhibits camera effects, in some way it ends up duplicating the effects. However, it was observed that this approach produced results more consistent than when applying the effects only over the inserted virtual objects.

Since the usual Computer Graphics camera's model cannot produce the desired effects, they were generated through the post-processing of the composed image. The Bloom and Glare effects were based on Kawase's work [57]. They are achieved basically in three steps: first, a threshold operator is used in order to identify the areas with high energy; second, filters are used to generate the effects; and third, the generated effects are added to the composed image. The Exposure Control effect was accomplished through the Reinhard et al. technique [58]. Since it is done after the Bloom and Glare addition, the Exposure Control will also affect these two effects. The Exposure Control effect is also performed in three steps: it first evaluates the average radiance of the composed scene; next, each pixel is scaled according to the obtained average; at last, a tone mapping operator is used to compress the scaled pixels to an appropriated dynamic range.

5. RPR-SORS API

The techniques presented in the last section were placed together in an all-in-one solution in order to achieve a seamless insertion of virtual objects. This solution consists of a photorealistic pipeline

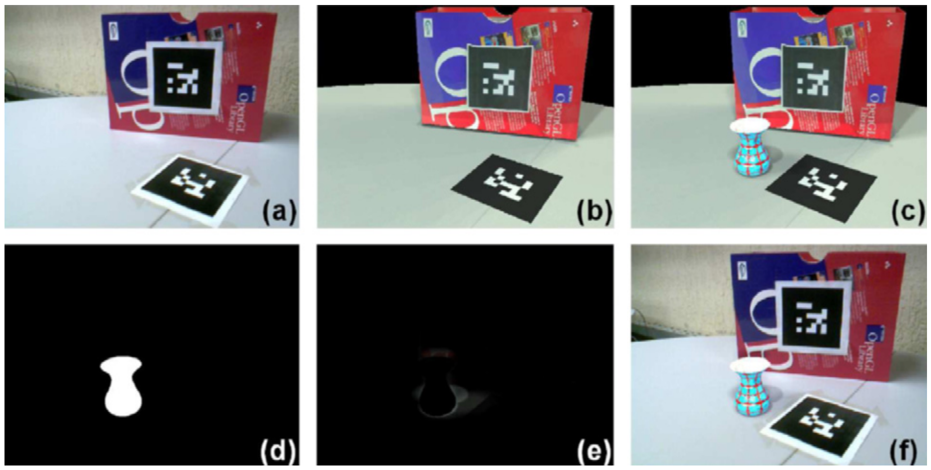


Fig. 12. In (a), captured real background image. In (b), first rendered image exhibiting only the local scene. In (c), second rendered image exhibiting the local and the virtual scenes. In (d), mask to the virtual scene regions. In (e), difference of the two rendered images regarding only the local scene regions. In (f), the final composition.

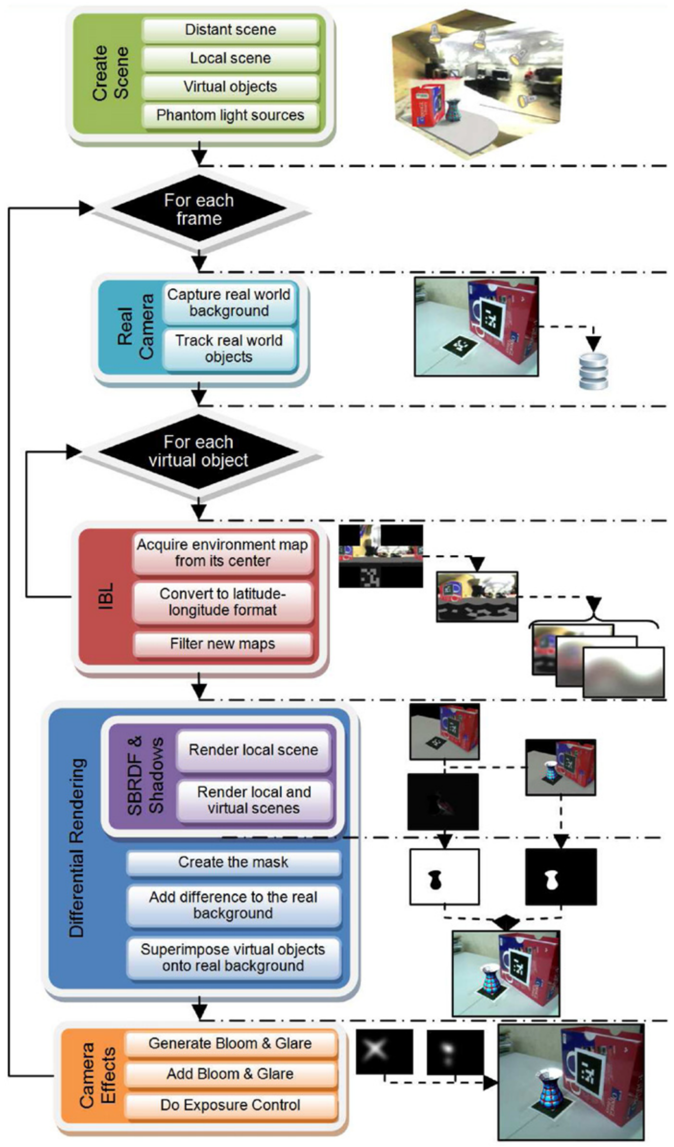


Fig. 13. The proposed chain of stages used to coherently augment real scenes. At the bottom-right, the final result is shown.

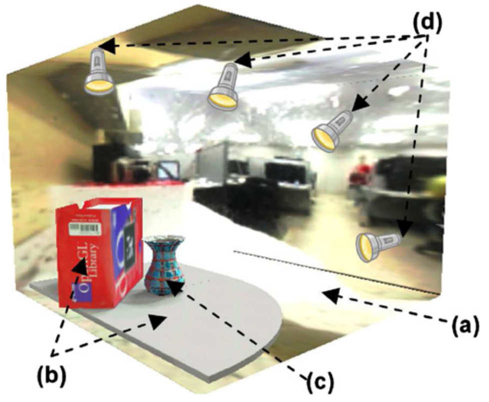


Fig. 14. Illustration of an internal scene setup. (a) is the distant scene, (b) are the local scene objects (an OpenGL red box and a gray table), (c) is the virtual scene object (a porcelain vase), and (d) are the phantom light sources. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

(see Fig. 13) implemented as an AR API. It was implemented in C/C++ using OGRE [59]. The shading language used was Cg. The API also used the OGREAR library [60], which wraps ARToolKitPlus [61] integrating it with OGRE. This allows to easily change the underlying tracking technique if desired.

The first stage of the pipeline is the scene creation. It consists in creating the scene of interest that is a compound of: the distant scene, the local scene, the virtual scene, and the phantom light sources. These light sources are created as explained in Section 4.3 aiming only shadow generation. Fig. 14 depicts a scene setup. This stage is performed only during the application initialization and it loads resources like: 3D models, textures, material scripts, and shaders code.

The next stages are performed in a loop in order to generate each frame. After scene setup, the real scene background is captured using a web camera. This is the image that will be augmented. This image is also used as an input to the OGREAR/ARToolKitPlus in order to track real objects tagged with markers.

In the next stage the illumination is evaluated by generating four illuminant environment maps per virtual object (see Section 4.1). At first, a cubic map is captured from a virtual object center. Then, this map is converted to the latitude–longitude format (the specular map). And lastly, the three remaining maps are filtered. This step by step is performed for each virtual object.

The real and virtual scenes are composed in the next stage using the Differential Rendering technique. First, the two required images are rendered using the shadowing technique (PCSS). The SBRDF technique is used only for virtual objects during the second image rendering. Since a rough reflectance model is enough to the local scene, photos of the real objects were used as textures for the phantom objects. Next, a mask to the virtual objects' regions is created in the stencil buffer and the remaining of the composition is performed as explained in Section 4.4.

The final stage performs a post-processing of the augmented scene in order to simulate the camera effects. First the Bloom and Glare effects are generated. Next, they are added to the composed image. At last, the Exposure Control is performed. After performing these effects, the augmentation is complete (see bottom-right of Fig. 13).

Disregarding the IBL and SBRDF techniques that must work together, each implemented technique may be arbitrarily enabled or disabled. This is an interesting feature to boost performance since not always all the techniques are necessary. Fig. 15 shows what happens when a given technique is enabled or disabled. When the rendering loop starts, the real world background is captured by the web camera and tracked, as mentioned before. Afterwards, for each virtual object with IBL/SBRDF enabled, the four illuminant environment maps are generated. If no virtual object has IBL/SBRDF enabled, then no environment maps will be generated.

Subsequently, Differential Rendering is handled. If it is enabled, the two images are rendered considering if Shadowing is enabled or not. The IBL/SBRDF technique is used only for the second image rendering. When a virtual object has IBL/SBRDF disabled, it is rendered through the fixed pipeline using the phantom light sources, otherwise the environment maps are used. Next, the composition is finished by adding the difference of both images and superimposing the virtual objects onto the real background (according to the generated stencil mask). If Differential Rendering is disabled, composition is done just rendering the virtual objects over the real background.

After the composition steps, camera effects are handled. If all of them are disabled, then the previous result is shown on the screen and application follows to next frame. Otherwise, if at least one of them is enabled, the previous result is placed into

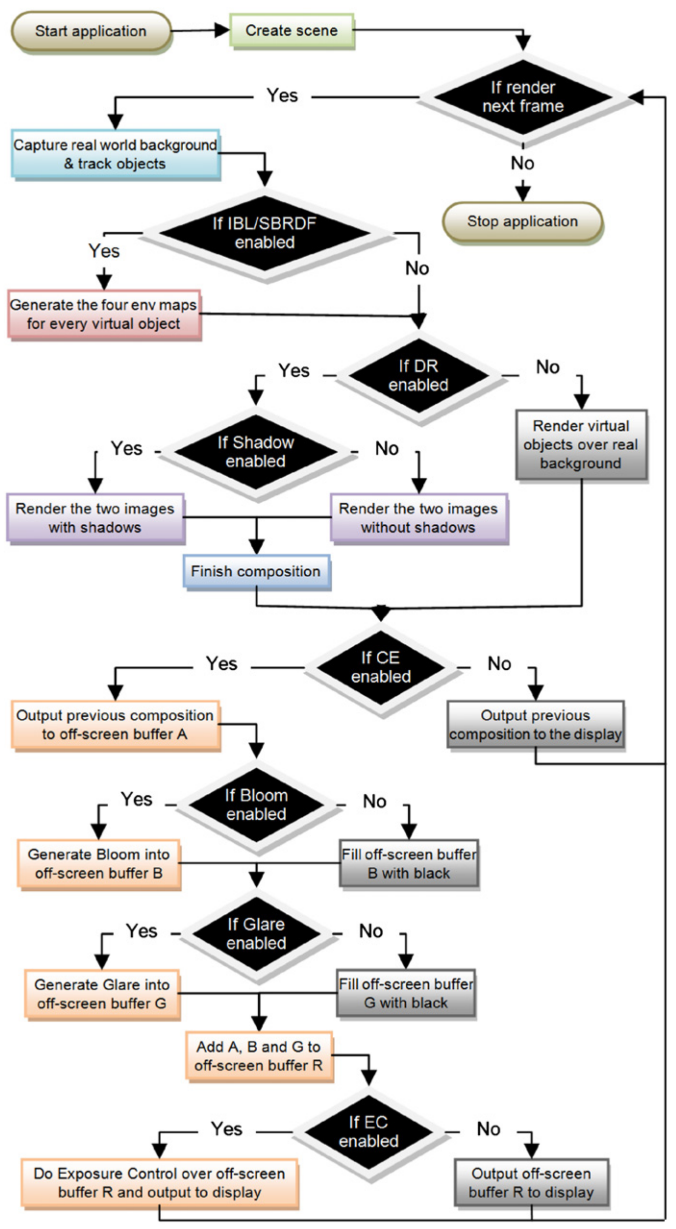

Start application
Create scene
If render next frame
Yes
No
Stop application
Capture real world background & track objects
If IBL/SBRDF enabled
Yes
No
Generate the four env maps for every virtual object
If DR enabled
Yes
No
Render virtual objects over real background
If Shadow enabled
Yes
No
Render the two images with shadows
Render the two images without shadows
Finish composition
If CE enabled
Yes
No
Output previous composition to off-screen buffer A
Output previous composition to the display
If Bloom enabled
Yes
No
Generate Bloom into off-screen buffer B
Fill off-screen buffer B with black
If Glare enabled
Yes
No
Generate Glare into off-screen buffer G
Fill off-screen buffer G with black
Add A, B and G to off-screen buffer R
If EC enabled
Yes
No
Do Exposure Control over off-screen buffer R and output to display
Output off-screen buffer R to display

Fig. 15. Flowchart showing what happens when a given technique is enabled or disabled.

off-screen buffer A and the enabled effects are generated. The first effect handled is Bloom. If it is enabled, then its result is placed into off-screen buffer B, otherwise this off-screen buffer is filled with black. The same idea is employed to the Glare effect although in this case the result is placed into off-screen buffer G. With Bloom and Glare effects handled, a pass that adds both effects and the previously composed image is performed. This pass outputs the addition result to off-screen buffer R. Despite this pass always receiving both Bloom and Glare off-screen buffers even when they are disabled (which may seem inefficient), filling the buffers with a neutral value (black) and using a single code version in the addition pass is a more convenient solution. This way, neither branching instructions (that would be inefficient in the GPU) nor multiple code versions (that would be hard to manage) were necessary. At the end, before the application loops to the next frame, Exposure Control is handled. If it is enabled, Exposure Control is performed with off-screen buffer R content, otherwise it is directly output to the display.

6. RPR-SORS material editor

As mentioned before, the RPR-SORS Material Editor is an authoring tool for designers, helping the creation of photorealistic materials. The goal of this editor is to provide an environment that allows designers to focus exclusively on material creation, and perform fine adjustments to achieve the desired appearance. The advantage of using this editor, compared to other material editors, is the possibility to visualize the results directly in AR, while the virtual objects interact with the real environment.

The purpose of the developed editor is to speed up the parameterization process for photorealistic materials. These materials require many parameters to be adjusted, and frequently they have side effects that can change the results of another parameter. Therefore, in order to achieve the best possible effects, it is an important authoring tool for real-time visual editing.

The RPR-SORS Material Editor was developed using the RPR-SORS API for the photorealistic effects, and therefore uses OGRE for rendering. It was written in C/C++ and currently runs only on the Windows platform. In addition, the application GUI uses the wxWidgets library [62], and marker tracking is performed by OGREAR/ARToolKitPlus. The application main window can be visualized in Fig. 16. Given that these libraries are multiplatform, the proposed editor can also be ported to different operating systems in the future.

The proposed editor focuses on two categories of resources: objects and materials. Objects can be added, removed, selected, and transformed (translation, rotation, scale). However, the object mesh cannot be directly edited, requiring an external application (such as 3DS Max) to model it. Materials can also be added, removed, and in addition, duplicated and assigned to objects. The created materials can be exported as scripts and resource files, which can be easily used by an external application that was developed using the RPR-SORS API.

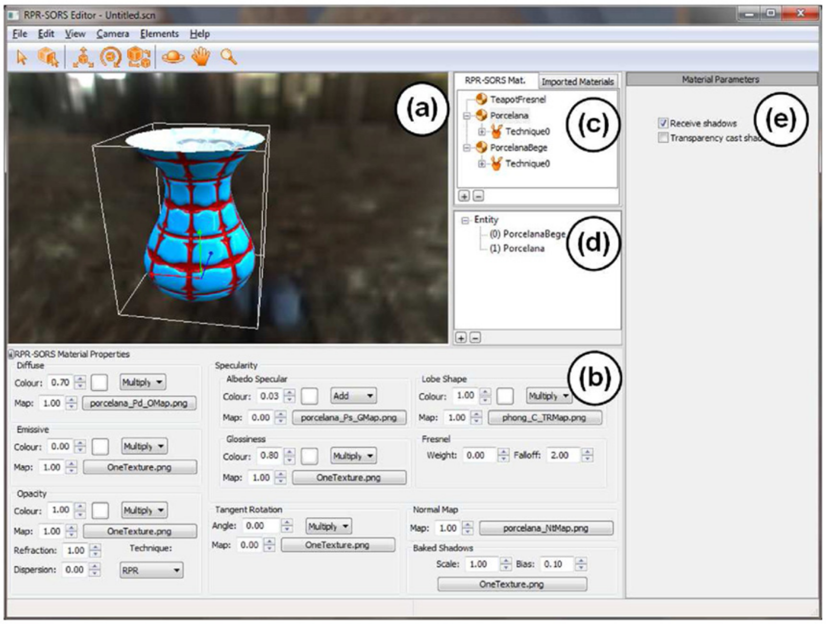
The material fine tuning is performed through the wide pane placed below the viewport (see Fig. 16b), where the SBRDF parameters are located. This panel is always visible to the user, since these are the most important parameters to be adjusted.

In addition to the parameters from the RPR-SORS API, the editor supports manipulation of the regular OGRE material parameters. These parameters are available on the panel placed on the right side of the interface (see Fig. 16e), and can change if the user is editing a material, technique, pass or texture unit (OGRE elements that define a material). 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.

Apart from the parameters for each material, the RPR-SORS Material Editor features the Scene Properties dialog, which provides tabs to configure AR mode, shadowing, and camera effects (see Fig. 17). When activated, the AR mode automatically detects the computer webcam and allocates the necessary resources. Then, the user can associate ARToolKitPlus markers for each object in the scene. In addition, it is also possible to enable the Differential Rendering technique and manage which objects will appear in the final scene (i.e., virtual objects) and which will function just as placeholder for real objects (i.e., phantom objects).

7. Results

The results presented in this section were obtained using a desktop PC with the following configuration: Windows® XP 32-bit Professional SP3, Intel® CoreTM i7 920 2.67 GHz processor, 3 GB of RAM and an NVIDIA GeForce® GTX 295 video card with 896 MB of memory. Application screen resolution was set to 1024×768 and four phantom light sources were created, each one using a 1024×1024 shadow map.



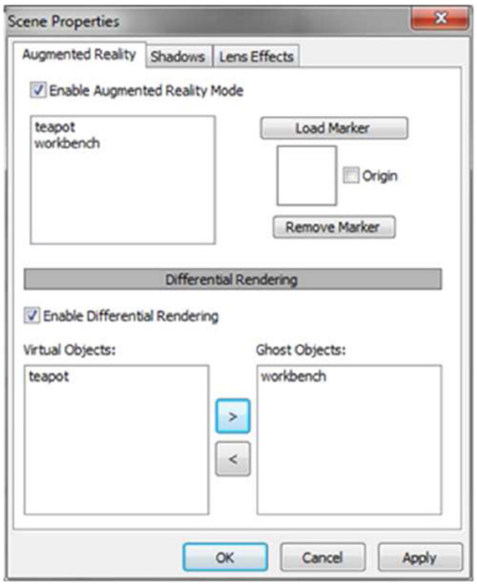
Receive shadows [✓]
Transparency cast sha

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
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

Fig. 16. RPR-SORS Material Editor interface: (a) viewport; (b) photorealistic material properties pane; (c) materials list; (d) objects list; and (e) OGRE material properties pane.



[✓] Enable Augmented Reality Mode
teapot
workbench
Origin [ ]
[✓] Enable Differential Rendering
Virtual Objects: teapot
Ghost Objects: workbench

Fig. 17. Scene Properties dialog. The AR mode tab contains AR markers configuration (top) and Differential Rendering (bottom).

The distant part of the real scene where the results were captured consists of: a yellowish wall, gray workstations and fluorescent lamps on the ceiling. The local scene consists of a gray table with a red box on top of it. Markers are attached to these last two objects in order to allow the registration of the respective phantom objects. These last objects were textured so that the indirect lighting effects could be handled. Fig. 18 shows the environment map that was pre-captured from the described real scene.

7.1. *Visual*

Fig. 19 presents a comparison between an image augmented without taking photorealism into account and another one obtained with the proposed toolkit.

The next figures were obtained using the proposed toolkit. Fig. 20 shows the effect of indirect lighting coming from the red box on a diffuse virtual teapot. The teapot tends to be slightly redder as it approaches the box.

Fig. 21 shows a virtual teapot with chrome material accordingly reflecting the real scene around it. Not only the red box, but the gray table, its marker, and the real light sources are also reflected.

Fig. 22 shows a metallic virtual table being inserted on the real scene. The virtual table has a material with brushed steel appearance. The circular patterns exhibited by this material are obtained by rotating the tangent vectors based on a monochromatic texture, as explained in Section 4.2.

Fig. 23 presents a transparent virtual teapot being inserted. The objects on the surroundings appear on the surface of the teapot. This figure also shows the transparency effect combined with the implemented normal mapping technique. As described in Section 4.2, the transparency effect is obtained by sampling the background image according to refracted vectors.

Fig. 24 shows a virtual teapot with increasing specularity at grazing angles. The material of this teapot has a diffuse and a specular component. It can be seen in the right image that the specular component tends to be more intense at the teapot's silhouette. As explained in Section 4.2, both specular albedo and glossiness coefficient are incremented at the regions near the silhouette.

Fig. 25 shows a virtual statue being gradually occluded by the red box. It can be seen in the lower right image that the reentrance in the middle of the box allows the base of the statue to still be visualized.

Fig. 26 presents a more complex virtual object being inserted into the scene. In this figure, a virtual mockup of a building can be seen. Camera effects, as described in Section 4.5, can be noted in the two upper images and the lower left one. While the Glare effect appears only on some windows of the building, the Bloom effect appears on these same windows and also on the sign of the building (the effects on the windows are due to their specular material that reflects the real fluorescent lamps and the effect on the sign is due to the emissive properties of its material). Precomputed self-shadows (stored in a texture) were also utilized in order to obtain this result.

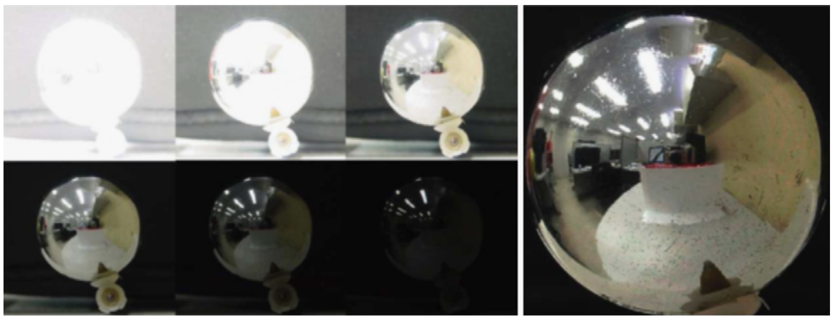


Fig. 18. On the left, chrome sphere shots with different exposure times. On the right, the resultant HDR spherical environment map.

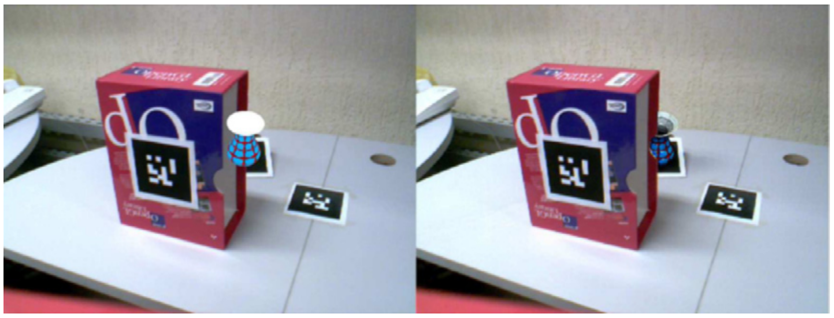


Fig. 19. Real scene augmented with a virtual vase. On the left, the image augmented without taking photorealism into account. Observe how the vase appears highlighted from the other real objects and without being properly occluded. On the right, the image obtained with the proposed toolkit seems more natural.

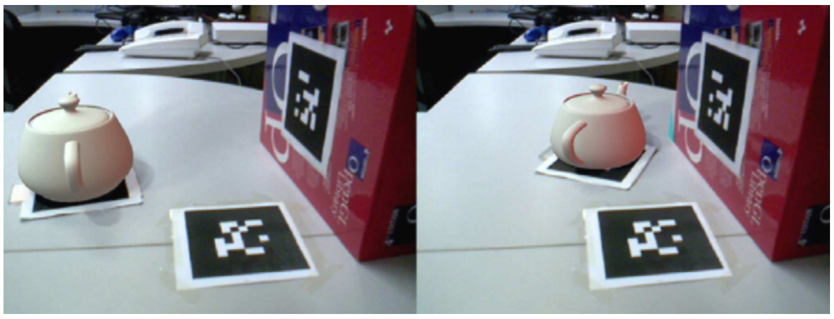


Fig. 20. Effect of indirect lighting coming from the red box on a virtual teapot with diffuse white material. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

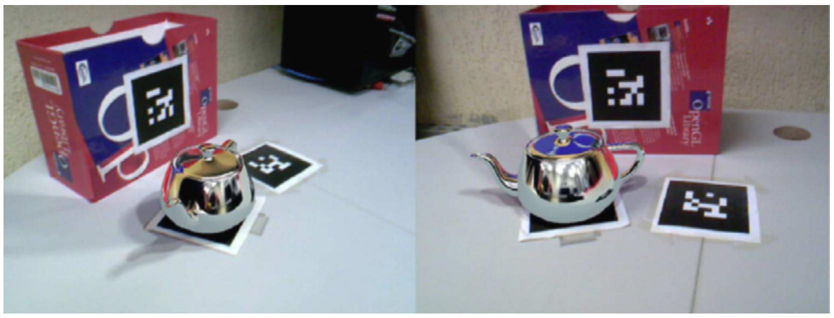


Fig. 21. Virtual teapot with chrome material accordingly reflecting the real scene around it. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

7.2. Performance

Depending on the number of virtual objects and which techniques are enabled, the presented pipeline achieves interactive rates. Fig. 27 exhibits the FPS obtained for each technique (in logarithmic scale) when the scene contains different number of virtual objects (VOs). These results were captured using the Hebe Goddess statue (this model is shown in Fig. 25 and has 31 939 vertices) as virtual object and the local scene has only one phantom object, the gray table. Since the IBL (illumination) and SBRDF (reflectance) techniques produce complimentary results, as well as the DR (composition) and the PCC (shadowing) techniques, they were evaluated together. Observe that when new objects are inserted, the IBL and SBRDF costs rapidly increase, while the remaining techniques are less sensitive to it.

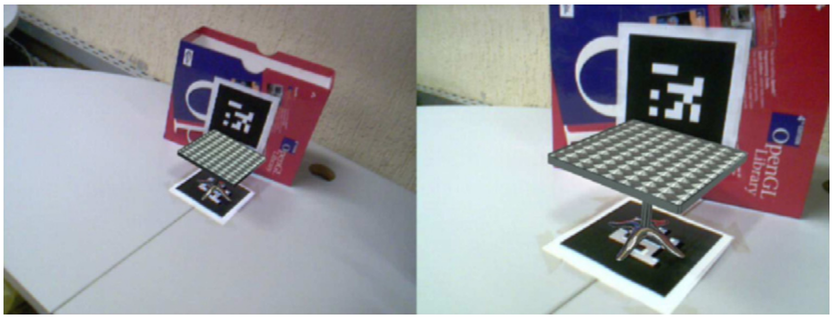


Fig. 22. Metallic virtual table exhibiting an anisotropic material with circular pattern.

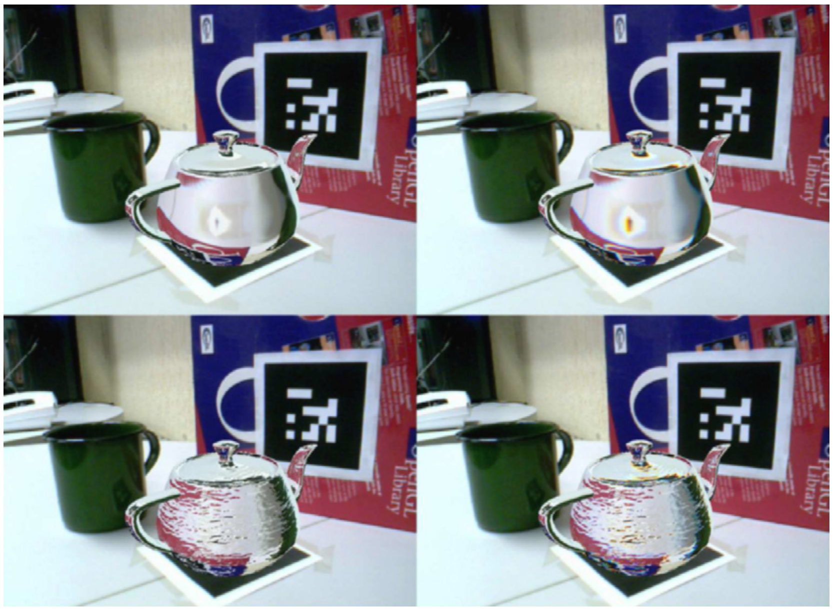


Fig. 23. Transparent virtual teapot refracting the real scene. In the lower images, a normal map was utilized in order to modify the original normal vectors, producing a rough appearance. On the right column, the images show the light dispersion effect. These results were captured with the shadowing technique disabled.

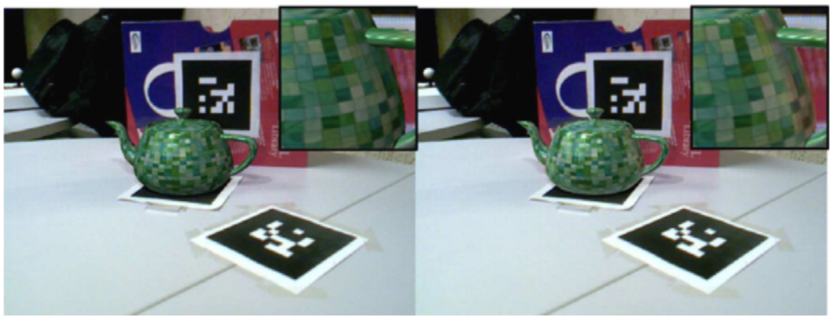


Fig. 24. In the left image, virtual teapot without increasing specularity at grazing angles. In the right image, the effect is enabled. Observe how the teapot's silhouette is shiner on the right image.

Fig. 28 exhibits the time spent in each pipeline stage when different number of virtual objects are inserted. The measured stages are: (1) environment maps acquisition from the center of each virtual object (six rendering passes are performed per map, one for each cube face); (2) conversion of the acquired environment maps to the latitude–longitude format (generating the specular environment maps); (3) glossy2 environment maps generation through a convolution operator (spatial domain filtering); (4) Spherical Harmonics transform of the maps; (5) filtering and inverse transform in order to generate the glossy1 environment maps; (6) filtering and inverse transform in order to generate the diffuse environment maps; (7) rendering of the first image of the Differential Rendering technique (the image exhibiting only the local scene); (8) rendering of the second image of the Differential Rendering technique (the image exhibiting both local and virtual scenes); (9) scene composition (it involves the mask generation, the computation of the difference between the two images, and the superimposition of the virtual objects into the real background); (10) Bloom effect generation; (11) Glare effect generation; and (12) exposure control. It is worth mentioning that these times were measured considering only the draw calls of each stage.

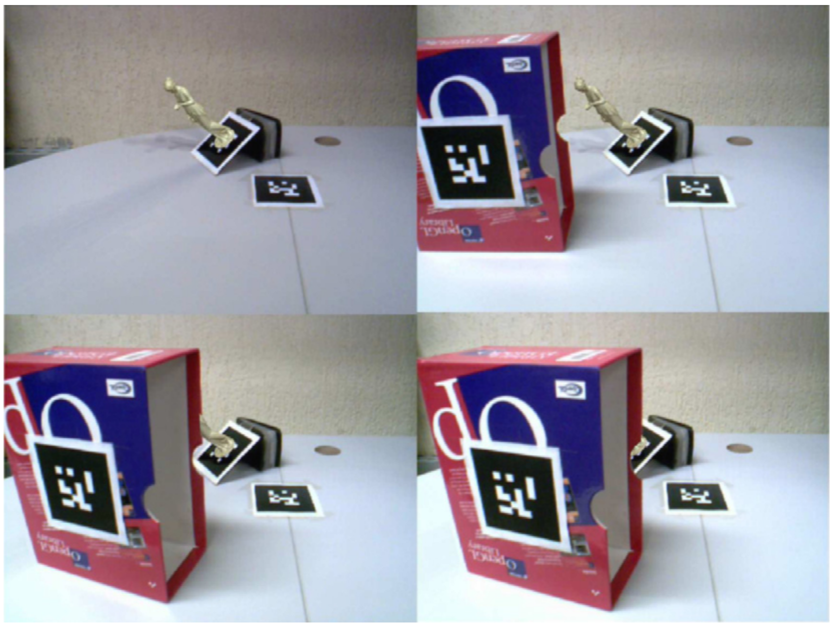


Fig. 25. From left to right, top to down, the virtual statue being gradually occluded by the red box. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

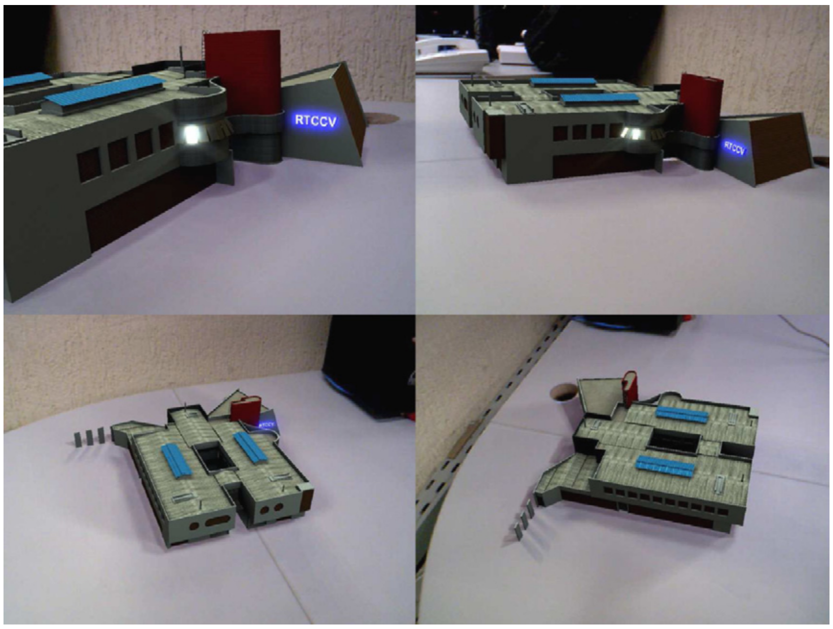


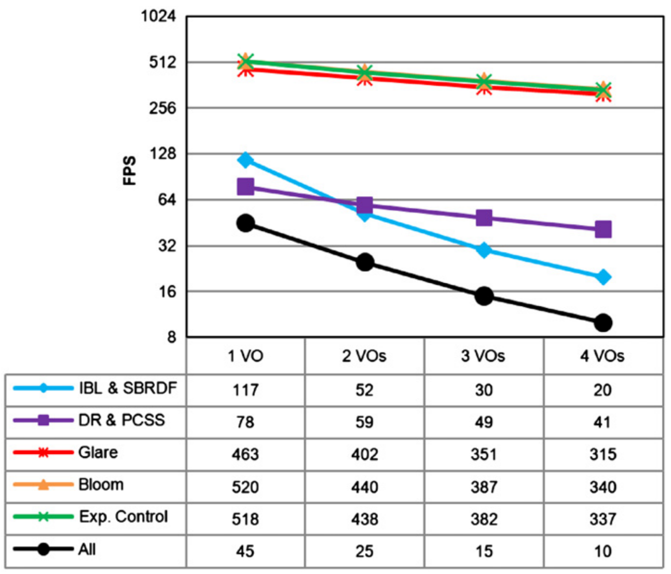
Fig. 26. Virtual mockup of a building being inserted into the real scene. Camera effects can be seen in the two upper images and the lower left one.

Observing the results in Fig. 28 is easy to understand why the IBL and SBRDF techniques cause high performance drop when new virtual objects are inserted. Notice that stage (1) cost rapidly increases. It is mainly due to the fact that every virtual object needs to be rendered during each environment map acquisition. Optimizations like LOD (level of detail) and material fallbacks could minimize this issue. The costs of the remaining IBL stages (2)–(6) increase linearly with virtual objects insertion. However, stage (4) is naturally expensive due to the elevated amount of samples needed to perform a Spherical Harmonics transform of degree 15. Stage (8) (where the SBRDF technique is performed) is also expensive and has a linear cost.

Stage (7) kept a constant cost since the number of phantom objects was not changed. A tiny overhead can be observed in stage (9) when new virtual objects are inserted. It happens since the mask creation step of this stage handles virtual objects. However, differently from stage (8), just the depth information of these objects is used in this case. That is the reason why stage (9) consumes significant less time than stage (8). The camera effects are independent from the number of virtual objects (see stages (10)–(12) in Fig. 28), since they are generated in screen space. The performance drop for these effects observed in Fig. 27 is due to the inherent rendering cost of the inserted objects.

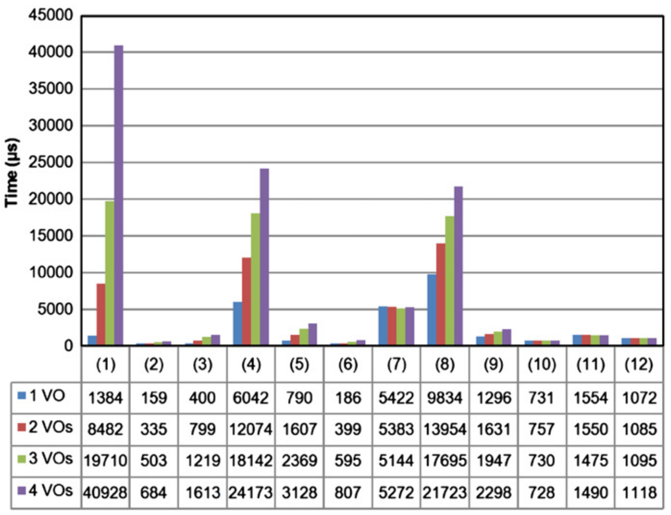
7.3. Experimental application

The first step of the development of the experimental application was creating some prototypes. One of them was a show room application for car manufacturers (Fig. 29a). This idea evolved to an architecture application where mockups could be visualized for advertising (Fig. 29b). In order to test the tracking system flexibility, this second prototype used a keypoint based markerless solution [63]. The application scenario consists of a newspaper ad for house sales. When this ad is shown to the application, the building mockup is rendered. The markerless solution was appropriate in this case because the ad itself could be used as a marker, avoiding intrusive elements.



	1 VO	2 VOs	3 VOs	4 VOs
IBL & SBRDF	117	52	30	20
DR & PCSS	78	59	49	41
Glare	463	402	351	315
Bloom	520	440	387	340
Exp. Control	518	438	382	337
All	45	25	15	10

Fig. 27. FPS obtained for each technique (in logarithmic scale) when different number of virtual objects are inserted. The last line exhibits the FPS when all techniques are enabled.



	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
1 VO	1384	159	400	6042	790	186	5422	9834	1296	731	1554	1072
2 VOs	8482	335	799	12074	1607	399	5383	13954	1631	757	1550	1085
3 VOs	19710	503	1219	18142	2369	595	5144	17695	1947	730	1475	1095
4 VOs	40928	684	1613	24173	3128	807	5272	21723	2298	728	1490	1118

Fig. 28. Time spent (µs) by each pipeline stage for different number of virtual objects.

These prototypes contributed to the development of the experimental application which is a Computer-Aided Design (CAD) AR application, named Scene Designer. It was conceived as a possible retail solution for architects that want to virtually decorate apartments in order to show to clients.

The very first stage of the Scene Designer development involved building a real model of an apartment with stiff paper (Fig. 30b) based on a blueprint (Fig. 30a). This replica was modeled intentionally without some walls for better visualization of the objects within it. Afterwards, a virtual model was created mimicking the real one in order to be used as phantom object (Fig. 30c).

The Scene Designer was developed using the RPR-SORS API. Since the photorealistic techniques were provided by the API, it was not necessary to spend time re-implementing them. The 3D furniture models were created in 3DS Max and then exported to the RPR-SORS Material Editor, where materials were edited. Without the proposed tool, the time necessary to achieve the same results could be longer or the appearance could not be satisfactory. The modeled 3D furniture library contains TV racks, tables, center tables, shelves, sofas, chairs, luminaries, paintings, and carpets.

The user can add pieces of furniture one by one and transform them. Simple material modifications, such as diffuse, specular, and emissive color change, are possible as well. In addition, if the object is animated (e.g., a rotating fan) the Scene Designer supports toggling the animation on and off, as well as modifying the state of the animation through the timeline. A decorated apartment is shown in the Scene Designer GUI in Fig. 31.

This experimental application is a sample of what the developers can achieve using the RPR-SORS toolkit. The obtained FPS rate was kept within interactive rates for up to 7 objects, for all the effects simultaneously enabled. The application starts with no object at approximately 180 FPS and falls down to 5 FPS when seven objects are present in the scene, following the tendency depicted in Fig. 27. Regardless of the low FPS rate, the application still coherently manages user inputs, for object manipulation.

8. User evaluation

A user evaluation was conducted in order to identify RPR-SORS benefits and limitations. For the pipeline evaluation, users were asked to implement a simple AR application using the RPR-SORS API. Whereas for the material editor, users should create and apply materials in a previously modeled object.

Users task can be summarized in: initialize and parameterize RPR-SORS API resources; and create objects to populate the scene. Since the API does not have a formal documentation, a step-by-step explanation for what should be done was provided and there was always a conductor helping the user. Five users evaluated the RPR-SORS API. The chosen users did not have any previous knowledge about the RPR-SORS although they are minimally experienced with

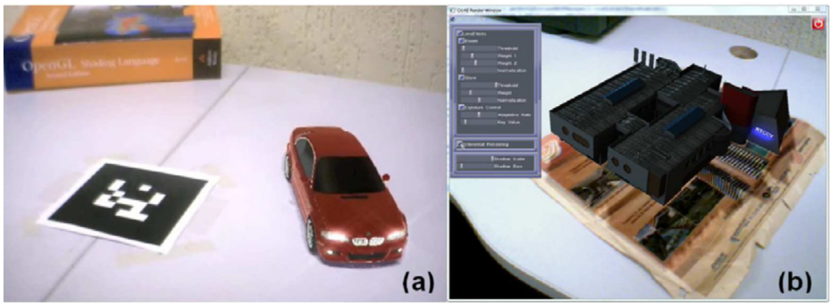


Fig. 29. On the left, show room application exhibiting a car. On the right, markerless architecture application exhibiting a building.

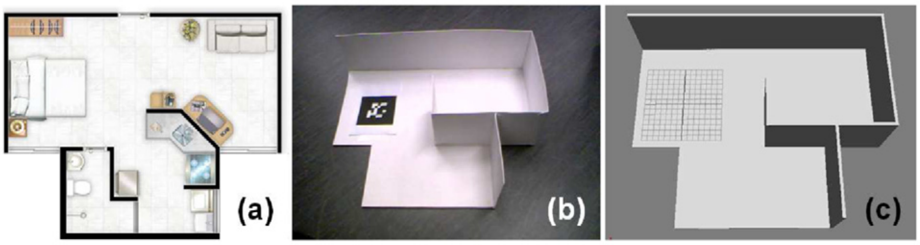


Fig. 30. Apartment model for Scene Designer. (a) Original blueprint used as inspiration, (b) real model built with stiff paper, and (c) the 3D model that is used as phantom object.

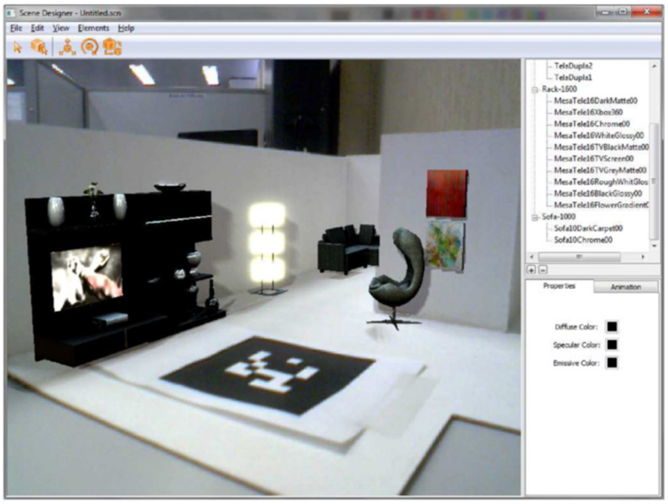


Fig. 31. Scene Designer GUI showing a decorated apartment.

AR. The goal was to create an application similar to the final result in Fig. 13. Users started from a sample application which already created a window and exhibited the image captured by a webcam (OGREAR/ARToolKitPlus was used to handle webcam and makers tracking).

Users spent on an average 40 min to finish the application implementation. After that, they were asked to answer some questions. In general, users said after writing a few lines of code, the following lines were much more easier to write. This result suggests the RPR-SORS API has a smooth learning curve. Users also mentioned about how much time they saved by using the RPR-SORS API. Some of them said that if they had to rewrite the features provided by the API, they would not have achieved the same result in a timely fashion. Others mentioned that the task would be impracticable. Therefore, for the evaluated kind of application it is worth using the RPR-SORS API. Users also mentioned that they did not know similar toolkits. Most AR toolkits care only about tracking and registering, rendering is usually neglected. Finally, users were asked about limitations. One of them complained about tracking/registering failures, which is not actually RPR-SORS API fault. Another problem observed was the lack of an anti-aliasing processing, making virtual objects appear with jagged edges. Since background image is naturally smooth, this problem can be even more apparent in AR applications. Therefore, an anti-aliasing technique would be a good improvement.

The experiment involving the RPR-SORS Material Editor consisted of the following task: replicate a object that was previously created using the same editor, having as reference only its visual appearance. The object used was a TV rack, containing 10 different materials. Not every possible material was covered in this experiment, due to time constraints (e.g., tangent varying materials were not present in this model). Four volunteers that had no previous contact with the tool were selected to perform the tests (some of them were also involved in the RPR-SORS API experiment). These subjects were chosen considering their knowledge on both CG concepts and 3D modeling tools. The tests were successfully completed by three of them, while one person failed to reproduce the same appearance, alleging that the interface was too different from the tools he was used to. After the task completion, the subjects were interviewed in order to retrieve their opinions about the experience.

The results obtained by the experiment indicate that the Material Editor tool provides a reasonable improvement to the task of creating photorealistic virtual objects. All the users spent about 30–40 minutes to achieve the desired results. Commonly, this task would be performed by editing text scripts containing the material properties, and the possibility to see in real-time the final appearance was reported as a gain in productivity. It was observed that the first materials that the users had to set up spent more time, since they were learning the new tool, and the last materials were concluded much faster. Some improvements were suggested by the users, such as template material library and a more intuitive interface. Thus, these tests also indicate that an usability study is required to improve the user experience in a final version.

9. Conclusion and future work

This work presented the state of the art in the field of photorealistic rendering for AR. It also described an authoring toolkit created by the authors that aims to facilitate the development of photorealistic AR applications through the use of an API and a material editor tool. The toolkit comprises a flexible pipeline proposed by the authors that allows enabling/disabling effects in a transparent way. In addition, it also encompasses punctual extensions developed by the authors regarding the IBL and SBRDF techniques.

Based on the experiments performed, it was noted that the RPR-SORS toolkit was able to fulfill its goals, providing a photorealistic solution that is very complete when compared to the existing ones in the literature. Satisfactory results were obtained regarding visual aspects as well as performance. A user evaluation was also conducted showing that the presented toolkit is effective. It was observed that the applications are more sensitive to the amount of virtual objects due to the cost of computing the four environment maps for each object. The creation of an extremely complex scene would decrease the frame rate and compromise user interaction. This restriction is not so relevant to the proposed material editor, since it was observed that its most common use is to refine the appearance of a single object and to insert few additional objects in order to study the interaction between them.

As future work, a technique for building the environment map of the real world in real-time [64] could be adapted to interactively generate HDR maps. New rendering techniques are also planned to be incorporated to the RPR-SORS pipeline, such as subsurface scattering (SSS). Reflections and indirect illumination effects could also be captured on the local scene (i.e., the real objects), by using the same approach adopted to generate these effects on the virtual scene (i.e., capturing and filtering environment maps for each phantom object).

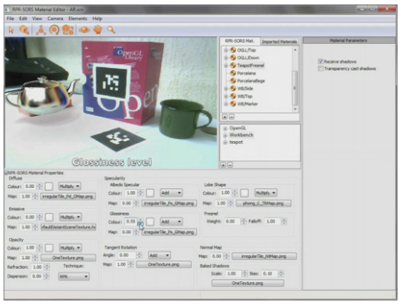
The resources exported by the Material Editor could be optimized. The material and shader files exported by the editor currently contain all the possible effects that can be enabled in the object. However, some objects can be essentially simpler than others (e.g., completely specular objects do not require diffuse textures in their material), and thus should avoid allocating unnecessary resources. Furthermore, a completely specular object not even requires the diffuse, glossy1, and glossy2 maps to be generated.

The overall performance of the API and the Material Editor could also be improved by not rendering all the illuminant environment maps in every frame. For example, a given virtual object would be parameterized to generate environment maps only after two or three frames since the last generation. This would cause a delay in the map update procedure for each object of the scene, but if this delay is kept at a small number of frames, the user should not be able to easily notice resulting inconsistencies. Since the environmental map generation is the bottleneck of the RPR-SORS pipeline, this idea could notably increase performance. This dependence on the number of objects in the scene also suggests that ray tracing can be a good option instead of rasterization.

Further usability tests should be done in order to refine the user interface of the RPR-SORS Material Editor. Although the current interface is functional and allows the manipulation of every important design parameter, formal usability studies are required to make the user interface simpler and more intuitive, without harming the capability of editing so many parameters accurately. Detailed case studies are also planned to be held in order to evaluate the effectiveness of the toolkit for aiding the development of photorealistic AR systems.

Appendix A. Supplementary material

The following is the supplementary data to this article: Video S1.



Video S1

A video clip is available online. Supplementary material related to this article can be found online at doi:10.1016/j.cag.2011.12.003.

References

[1] Sony Computer Entertainment. Eyepet TM, 2011. Available from: 〈http://www.eyepet.com〉.
[2] Kakuta T, Oishi T, Ikeuchi K. Virtual kawaradera: fast shadow texture for augmented reality. In: Proceedings of international society on virtual systems and multimedia; 2008. p. 141–50. doi:10.1.1.80.8246.
[3] Gibson S, Cook J, Howard T, Hubbold R. Rapid shadow generation in real-world lighting environments. In: Proceedings of the 14th Eurographics workshop on rendering (EGRW '03). Aire-la-Ville, Switzerland: Eurographics Association; 2003. p. 219–29. doi:10.1109/882404.882436. ISBN 3-905673-03-7.
[4] Lensch HPA, Goesele M, Chuang YY, Hawkins T, Marschner S, Matusik W, et al. Realistic materials in computer graphics. In: ACM SIGGRAPH 2005 courses (SIGGRAPH '05). New York, NY, USA: ACM; 2005. doi:10.1145/1198555.1198601.
[5] Pessoa SA, Apolinário EL, Moura GdS, Lima JPSM, Teichrieb V, Kelner J. Illumination techniques for photorealistic rendering in augmented reality. In: Proceedings of the X symposium on virtual and augmented reality (SVR '08); 2008. p. 223–32.
[6] Pessoa SA, Moura GdS, Lima JPSM, Teichrieb V, Kelner J. A global illumination and BRDF solution applied to photorealistic augmented reality. In: Proceedings of the 2009 IEEE virtual reality conference. Washington, DC, USA: IEEE Computer Society: 2009. p. 243–4. doi:10.1109/VR.2009.4811036. ISBN 978-1-4244-3943-0.
[7] Pessoa S, Moura G, Lima J, Teichrieb V, Kelner J. Photorealistic rendering for augmented reality: a global illumination and BRDF solution. In: Virtual reality conference (VR). IEEE; 2010. p. 3–10. doi:10.1109/VR.2010.5444836.
[8] Moura GdS, Pessoa SA, Lima JPSM, Teichrieb V, Kelner J. RPR-SORS: an authoring toolkit for photorealistic AR. In: Proceedings of the XIII symposium on virtual and augmented reality (SVR '11); 2011. p. 178–87. doi:10.1109/SVR.2011.14.
[9] Fournier A, Gunawan AS, Romanzin C. Common illumination between real and computer generated scenes. Technical Report, Vancouver, BC, Canada; 1992.
[10] Drettakis G, Robert L, Bugnoux S. Interactive common illumination for computer augmented reality. In: 8th Eurographics workshop on rendering, Saint Etienne, France; 1997. p. 45–56.
[11] Loscos C, Frasson MC, Drettakis G, Walter B, Granier X, Poulin P. Interactive virtual relighting and remodeling of real scenes. In: Lischinski D, Larson G, editors. Rendering techniques '99 (Proceedings of the 10th Eurographics workshop on rendering), vol. 10. New York, USA: Springer-Verlag, Wien; 1999. p. 235–46.
[12] Sato I, Sato Y, Ikeuchi K. Acquiring a radiance distribution to superimpose virtual objects onto a real scene. IEEE Trans Vis Comput Graph 1999;5(1):1–12. doi:10.1109/2945.764865.
[13] Loscos C, Drettakis G, Robert L. Interactive virtual relighting of real scenes. IEEE Trans Vis Comput Graph 2000;6(4):289–305. doi:10.1109/2945.895874.
[14] Debevec PE, Malik J. Recovering high dynamic range radiance maps from photographs. In: Proceedings of the 24th annual conference on computer graphics and interactive techniques (SIGGRAPH '97). ACM Press/Addison-Wesley Publishing Co.; 1997. 369–78. doi:10.1145/258734.258884. ISBN 0-89791-896-7.
[15] Debevec P. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In: Proceedings of the 25th annual conference on computer graphics and interactive techniques (SIGGRAPH '98). New York, NY, USA: ACM; 1998. p. 189–98. doi:10.1145/280814.280864. ISBN 0-89791-999-8.
[16] Breen DE, Whitaker RT, Rose E, Tuceryan M. Interactive occlusion and automatic object placement for augmented reality. Comput Graph Forum 1996;15(3):11–22. doi:10.1111/1467-8659.1530011.
[17] Wloka MM, Anderson BG. Resolving occlusion in augmented reality. In: Proceedings of the 1995 symposium on interactive 3D graphics (I3D '95). New York, NY, USA: ACM; 1995. p. 5–12. doi:10.1145/199404.199405. ISBN 0-89791-736-7.
[18] Stauder J. Augmented reality with automatic illumination control incorporating ellipsoidal models. IEEE Trans Multimedia 1999;1(2):136–43. doi:10.1109/6046.766735.
[19] Kanbara M, Yokoya N. Real-time estimation of light source environment for photorealistic augmented reality. In: Proceedings of the 17th international conference on pattern recognition (ICPR '04), vol. 2; 2004. p. 911–4. doi:10.1109/ICPR.2004.1334407.
[20] Grosch T, Müller S, Kresse W. Goniometric light reconstruction for augmented reality image synthesis. In: Graphiktag 2003; 2003. p. 3–13.
[21] Wang Y, Samaras D. Estimation of multiple directional light sources for synthesis of augmented reality images. Graph Models 2003;65:185–205. doi:10.1016/S1524-0703(03)00043-2.
[22] Gibson S, Murta A, Interactive rendering with real-world illumination. In: Proceedings of the eurographics workshop on rendering techniques 2000. London, UK: Springer-Verlag; 2000. p. 365–76. ISBN 3-211-83535-0.
[23] Grosch T, Eble T, Mueller S. Consistent interactive augmentation of live camera images with correct near-field illumination. In: Proceedings of the 2007 ACM symposium on virtual reality software and technology (VRST '07). New York, NY, USA: ACM; 2007. p. 125–32. doi:10.1145/1315184.1315207. ISBN 978-1-59593-863-3.
[24] Agusanto K, Li L, Chuangui Z, Sing NW, Photorealistic rendering for augmented reality using environment illumination. In: Proceedings of the 2nd IEEE/ACM international symposium on mixed and augmented reality (ISMAR '03). Washington, DC, USA: IEEE Computer Society; 2003. p. 208. ISBN 0-7695-2006-5.
[25] Heymann S, Smolic A, Müller K, Froehlich B. Illumination reconstruction from real-time video for interactive augmented reality. In: 6th international workshop on image analysis for multimedia interactive services (WIAMIS), 2005.
[26] Hughes C, Konttinen J, Pattanaik S. The future of mixed reality: issues in illumination and shadows. In: Proceedings of the interservice/industry training, simulation & education conference (I/ITSEC); 2004.
[27] Nishina Y, Okumura B, Kanbara M, Yokoya N. Photometric registration by adaptive high dynamic range image generation for augmented reality. In: 7th

IEEE/ACM international symposium on mixed and augmented reality (ISMAR '08); 2008. p. 53–6. doi:10.1109/ISMAR.2008.4637323.

[28] Jensen T, Andersen MS, Madsen CB. Real-time image based lighting for outdoor augmented reality under dynamically changing illumination conditions. In: Proceedings of the international conference graphics theory and applications; 2006. p. 364–71.

[29] Feng Y. Estimation of light source environment for illumination consistency of augmented reality. Congress on image and signal processing 2008;3: 771–5. doi:10.1109/CISP.2008.87.

[30] Madsen CB, Nielsen M. Towards probe-less augmented reality. In: Proceedings of the international conference on graphics theory and applications (GRAPP); 2008. p. 255–61.

[31] Haller M, Drab S, Hartmann W, 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 (VRST '03). New York, NY, USA: ACM; 2003. p. 56–65. doi:10.1145/1008653.1008665. ISBN 1-58113-569-6.

[32] Madsen CB. Using real shadows to create virtual ones. In: Proceedings of the 13th Scandinavian conference image analysis, 2003. p. 820–7.

[33] Jacobs K, Nahmias JD, Angus C, Reche A, Loscos C, Steed A. Automatic generation of consistent shadows for augmented reality. In: Proceedings of graphics interface 2005 (GI '05). School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada: Canadian Human–Computer Communications Society; 2005. p. 113–20. ISBN 1-56881-265-5.

[34] Supan P, Stuppacher I, Haller M. Image based shadowing in real-time augmented reality. Int J Virtual Reality 2006;5(3):1–7.

[35] Madsen CB, Laursen RE. A scalable GPU-based approach to shading and shadowing for photorealistic real-time augmented reality. In: Proceedings of computer graphics theory and applications; 2007. p. 252–61.

[36] Frahm JM, Koeser K, Grest D, Koch R. Markerless augmented reality with light source estimation for direct illumination. In: Proceedings of the 2nd IEE European conference on visual media production (CVMP '05); 2005. p. 211–20.

[37] Bimber O, Grundhöfer A, Wetzstein G, Knödel S, Consistent illumination within optical see-through augmented environments. In: Proceedings of the 2nd IEEE/ACM international symposium on mixed and augmented reality (ISMAR '03). Washington, DC, USA: IEEE Computer Society; 2003. p. 198. ISBN 0-7695-2006-5.

[38] Bradley D, Roth G. Augmenting non-rigid objects with realistic lighting. Technical Report, NRC Institute for Information Technology; 2004.

[39] Bradley D, Roth G, Bose. Augmented reality on cloth with realistic illumination. Mach Vision Appl 2009;20(2):85–92. doi:10.1007/s00138-007-0108-9.

[40] Grosch T. Panoar: interactive augmentation of omni-directional images with consistent lighting. In: Proceedings of MIRAGE; 2005. p. 25–34.

[41] Okumura B, Kanbara M, Yokoya N. Augmented reality based on estimation of defocusing and motion blurring from captured images. In: Proceedings of the 5th IEEE and ACM international symposium on mixed and augmented reality (ISMAR '06). Washington, DC, USA: IEEE Computer Society; 2006. p. 219–25. doi:10.1109/ISMAR.2006.297817. ISBN 1-4244-0650-1.

[42] Scheer F, Abert O, Muller S. Towards using realistic ray tracing in augmented reality applications with natural lighting. In: Proceedings of the 4th workshop virtual and augmented reality of the GI-group (VR/AR); 2007.

[43] Microsoft. The directx software development kit, 2011. Available from 〈http://msdn.microsoft.com/en-us/library/ee416398(v=VS.85).aspx〉.

[44] Ramamoorthi R, Hanrahan P. An efficient representation for irradiance environment maps. In: Proceedings of the 28th annual conference on computer graphics and interactive techniques (SIGGRAPH '01). New York, NY, USA: ACM; 2001. p. 497–500. doi:10.1145/383259.383317. ISBN 1-58113-374-X.

[45] King G. Real-time computation of dynamic irradiance environment maps. In: GPU Gems 2. Addison-Wesley; 2005. p. 167–76.

[46] Wan L, Wong TT, Leung CS. Isocube: exploiting the cubemap hardware. IEEE Trans Vis Comput Graph 2007;13(4):720–31. doi:10.1109/TVCG.2007.1020.

[47] University of Southern California. HDR shop, 2011. Available from 〈http://gl.ict.usc.edu/HDRShop〉.

[48] Hensley J, Scheuermann T, Coombe G, Singh M, Lastra A. Fast summed-area table generation and its applications. Comput Graph Forum 2005;24(3):547–55. doi:10.1111/j.1467-8659.2005.00880.x.

[49] Mikkelsen M. Simulation of wrinkled surfaces revisited. Master thesis, Department of Computer Science at the University of Copenhagen; 2008.

[50] Lafortune EPF, Foo SC, Torrance KE, Greenberg DP. Non-linear approximation of reflectance functions. In: Proceedings of the 24th annual conference on computer graphics and interactive techniques (SIGGRAPH '97). New York, NY, USA: ACM Press, Addison-Wesley Publishing Co.; 1997. p. 117–26. doi:10.1145/258734.258801. ISBN 0-89791-896-7.

[51] McAllister DK, Lastra A, Heidrich W. Efficient rendering of spatial bidirectional reflectance distribution functions. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on graphics hardware (HWWS '02). Aire-la-Ville, Switzerland: Eurographics Association; 2002. p. 79–88. ISBN 1-58113-580-7.

[52] Foo SC. A gonioreflectometer for measuring the bidirectional reflectance of material for use in illumination computation. Master thesis. Cornell University, Ithaca, NY, USA; 1997.

[53] Fernando R, Kilgard MJ. Environment mapping techniques. In: The Cg tutorial. Addison-Wesley; 2003. p. 162–88.

[54] Debevec, P. A median cut algorithm for light probe sampling. In: ACM SIGGRAPH 2008 classes (SIGGRAPH '08). New York, NY, USA: ACM; 2008. pp. 1–3. doi:10.1145/1401132.1401176.

[55] Fernando R, Percentage-closer soft shadows. In: ACM SIGGRAPH 2005 sketches. SIGGRAPH '05. New York, NY, USA: ACM; 2005. p. 35. doi:10.1145/1187112.1187153.

[56] Wyman C, Hansen C. Penumbra maps: approximate soft shadows in real-time. In: Proceedings of the 14th eurographics workshop on rendering (EGRW '03). Aire-la-Ville, Switzerland: Eurographics Association; 2003. p. 202–7. ISBN 3-905673-03-7.

[57] Kawase M. RTHDRIBL—real-time high dynamic range image-based lighting; 2011. Available from 〈http://www.daionet.gr.jp/~masa/rthdribl〉.

[58] Reinhard E, Stark M, Shirley P, Ferwerda J. Photographic tone reproduction for digital images. ACM Trans Graph 2002;21(3):267–76. doi:10.1145/566654.566575.

[59] OGRE, OGRE—open source 3d graphics engine; 2011. Available from 〈http://www.ogre3d.org〉.

[60] Farias T, Lima JPSM, Teichrieb V, Kelner J. OGREAR: construction of augmented reality applications using high-level libraries. Technical Report, Federal University of Pernambuco, Recife, Pernambuco; 2007.

[61] Laboratory CD. Handheld AR project; 2011. Available from 〈http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php〉.

[62] wxWidgets Team, wxWidgets cross-platform GUI library; 2011. Available from 〈http://www.wxwidgets.org〉.

[63] do Monte Lima JPS, Simoes FPM, Figueiredo LS, Kelner J. Model based markerless 3D tracking applied to augmented reality. SBC J 3D Interact Syst 2010;1:2–15.

[64] DiVerdi S, Wither J, Höllerer T. All around the map: online spherical panorama construction. Comput Graph 2009;33(1):73–84. doi:10.1016/j.cag.2008.11.002.