

RENDERING SYNTHETIC OBJECTS INTO FULL PANORAMIC SCENES USING LIGHT-DEPTH MAPS

Aldo René Zang¹, Dalai Felinto² and Luiz Velho¹

¹*Visgraf Laboratory, Institute of Pure and Applied Mathematics (IMPA), <http://www.visgraf.impa.br>*

²*Fisheries Centre, University of British Columbia, <http://www.fisheries.ubc.ca>
zang@impa.br, d.felinto@fisheries.ubc.ca, lvelho@impa.br*

Keywords: photorealism, augmented reality, photorealistic rendering, HDRI, light-depth map, ibl illumination, 3d modeling, full panorama

Abstract: This photo realistic rendering solution address the insertion of computer generated elements in a captured panorama environment. This pipeline supports productions specially aiming at spherical displays (e.g., full-domes). Full panoramas have been used in computer graphics for years, yet their common usage lays on environment lighting and reflection maps for conventional displays. With a keen eye in what may be the next trend in the filmmaking industry, we address the particularities of those productions, proposing a new representation of the space by storing the depth together with the light maps, in a full panoramic light-depth map. Another novelty in our rendering pipeline is the one-pass solution to solve the blending of real and synthetic objects simultaneously without the need of post processing effects.

1 INTRODUCTION

In the recent years, we have seen an increase demand for immersive panorama productions. We believe this is a future for cinema innovation.

Panorama movies and experiments are almost as old as the cinema industry. Yet, this medium hasn't been explored extensively by film makers. One of the reasons been the lack of rooms to bring in the audience for their shows. Another is the time it took for the technology to catch up with the needs of the panorama market. Panorama films are designed to be enjoyed in screens with large field of views (usually 180°). And for years, around the world, there were only a few places designed to receive those projections (e.g. La Gode, in Paris, France). Since the past years, a lot of old planetariums are upgrading their old star-projectors to full digital projection systems. Additionally, new panorama capture devices are becoming more accessible everyday (e.g., Lady Bug, a camera that captures a full panorama in motion). And what to say of new consumer devices and applications such as Google Street View, gyrosopic based mobile panorama viewing apps?

We wanted to validate a workflow to work from panorama capturing, work the insertion of digital elements to build a narrative, and bring it back to the panorama space. There is no tool in the market right

now ready to account for the complete framework.

In order to address that, we presented in a previous work an end-to-end framework to combine panorama capture and rendered elements. The complete pipeline involves all the aspects of the environment capture, and the needed steps to work with a custom light-path algorithm that can handle this information (Felinto et al., 2012).

The original problem we faced was that full panoramas are a directional map. They are commonly used in the computer graphics industry for environment lighting and limited reflection maps. And they work fine if you are to use them without having to account for a full coherent space. However if a full panorama is the output format, we need more than the previous works can provide.

In that work (Felinto et al., 2012) we presented the concept of *light-depth environment maps* - a special environment light field map with a depth channel used to compute the position of light samples in real world - and explored mechanisms to build them. We are storing the environment information as a equirectangular panorama that contains the HDR lighting information and the reconstructed depth of the scene. This provides the necessary information to simulate accurate shadows and reflections of the original environment and its interactions with synthetic objects.

The rendering solution to deal with a *light-depth*

environment map, however, is non trivial. We here propose a solution for panoramic photo-realistic rendering of synthetic objects inserted in a real environment using a single-pass path tracing algorithm. We generate an approximation of the relevant environment geometries to get a complete simulation of shadows and reflections of the environment and the synthetic elements.

Additionally, the rendering is not constrained to do only panorama renders. If the production output is aimed at conventional displays, a restricted camera frustum can be used inside the environment model. That allows for camera panning, zooming and even camera traveling, proven the environment is properly reconstructed.

We developed a modified version of the open source software *Luxrender*, *ARLuxrender* (Zang and Velho, 2011b) and (Zang and Velho, 2011a), which allows for physic based lighting rendering of virtual scenes. Additionally an add-on for the 3D open source software *Blender* is available for the scene modeling and production steps, better explained at (Felinto et al., 2012). Additional information about this rendering algorithm and the production framework can be found in the *ARLuxrender* project web site (Zang and Velho, 2011a).

2 RELATED WORK

If we consider a particular point in the world, the light field at this point can be described as the set of all colors and light intensities reaching the point from all directions. A relatively simple way to capture this light field is by taking a picture of a mirror ball centered in this point in space - the mirror ball will reflect all the environment lights that would reach that point towards the camera. There are other techniques for capturing omni directional images including the use of fisheye lens, a mosaic of views, and special panoramic cameras. But even though there are multiple methods and tools to capture the environment lighting, this wasn't the case in our recent history.

The simplest example of image based illumination, dates from the beginning of the 80ies only. The technique is known as environment mapping or reflection mapping. In this technique, a photography of a mirror ball is taken and directly transferred to the synthetic object as a texture. The application of the technique using photographs of a real scene was independently tested by Gene Miller (Miller and Hoffmann, 1984) and Mike Chou (Williams, 1983). Right after, the technique started to get adopted by the movie industry with the work of Randal Kleiser

in the movie *Flight of the Navigator* in 1986, and the robot *T1000* from the movie *Terminator 2* directed by James Cameron in 1991. This technique proved successful for environment reflections, but limited when it comes to lighting - the capture image had a low dynamic range (LDR) insufficient to represent all the light range present in the original environment.

Debevec and Malik developed a technique of recovering high dynamic range radiance maps from photography taken with conventional camera, (Debevec and Malik, 1997). Many shots are taken with a variance of the camera exposure between them. After the camera response is calculated, the low dynamic range images are stacked into a single high dynamic range image (or *HDRI*) that represents the real radiance of the scene. We use this technique to assemble our environment map.

In 1998, Debevec presented a method to illuminate synthetic objects with lighting captured from the real world, (Debevec, 1998). This technique became known as *IBL* - Image Based Lighting. The core idea behind *IBL* is to use the global illumination from the captured image to simulate the lighting of synthetic objects seamlessly merged within the photography.

Zang expanded Debevec's work in synthetic object rendering developing an one-pass rendering framework, dispensing the needs of post-processing composition, (Zang and Velho, 2011b). In the same year (2011), Karsch presented a relevant work on rendering of synthetic objects in legacy photographs, using a different method for recovering the light sources of the scene, (K. Karsch and Hoiem, 2011).

As for panoramic content creation important work was developed in the past for real-time content in domes by Paul Bourke (Bourke and Felinto, 2010). Part of our work is inspired on his multiple ideas on full dome creation, and the work of Felinto in 2009, implementing Bourke's real-time method in *Blender* and proposing other applications for panorama content in architecture visualization, (Felinto, 2010).

3 PRODUCTION FRAMEWORK

The rendering is the final part of our production framework. All the steps of this process, rendering included, were validated using accessible tools and equipment. The complete description of this framework can be found in (Felinto et al., 2012). The final results you see in this paper were all generated with the following workflow.

Production Pipeline:

1. Environment capture and panorama assembling
2. Calibration of the equirectangular panorama
3. Environment modeling
4. Creation of the light-depth environment map
5. Modeling of the new objects
6. Rendering of the augmented reality panorama or restricted frustum

One of the most important steps on our workflow is to obtain a proper representation of the environment. This is used for the lighting reconstruction of the scene, the calibration of the original camera device, the background plate for our renders and for the scene depth reconstruction.

We settled for a industry de facto image format known as equirectangular panorama *HDR* images. This is a 2:1 image that encompass the captured light field.

In order to obtain the light field of the environment, we need to resort to capture devices. Photography cameras can be calibrated to work as a light sensor with the right techniques and post processing software. For the ambit of this project we chose to work with semi-professional photographic equipment, considering this a good trade-off between consumer cameras and full professional devices. We used Debevec's method to produce *HDR* maps from bracketed pictures (Debevec and Malik, 1997). In total we took 9 pictures per camera orientation with the total of 7 different orientations to obtain the total of a $360^\circ \times 360^\circ$ field of view from the camera point of view. The *HDR* stacks were then stitched together to produce an equirectangular panorama. We used a Nikon DX2S with a Nikon DX10.5mm fisheye lens, and a Manfrotto Pano head. For the *HDR* assembling we used *Luminance HDR* and for the stitching we used *Hugin*, both open source projects freely available in the Internet.

3.1 Environment Capture

A common solution for light field capturing - mirror ball pictures - was developed to be used as reflection map and lighting (Debevec and Malik, 1997). However, when it comes to background plates, our earlier tests showed that a mirror ball picture lack in quality, resolution and field of view. Therefore, in order to maximize the resolution of the captured light field we opted to take multiple photographs to assemble in the panorama. This is the method explored in our

production framework as presented in (Felinto et al., 2012).

To use multiple pictures to represent the environment is a common technique in photography and computational visioning. For rendered images, when a full ray tracer system is not available (specially aggravating for real-time applications) 6 pictures generated with a 90° frustum camera oriented towards a cube are sufficient to recreate the environment with a good trade-off between render size and pixel distortion (Bourke and Felinto, 2010).

We chose a fisheye lens (Nikon 10.5 mm) to increase the field of view per picture and minimize the number of required shots. We used 7 pictures in total, including the north and south pole (see figure 1(b)).

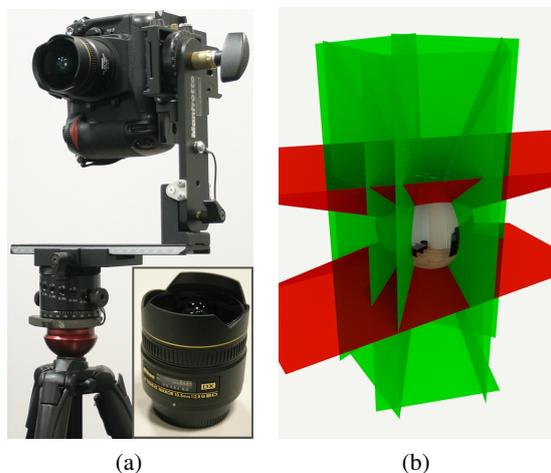


Figure 1: (a): Equipment used to capture the full environment: Manfrotto Pano Head, Nikon DX2S camera and Nikon 10.5mm fisheye lens. (b): Space partition used to assemble the panorama, 5 photos around z-axis, 1 zenith photo and 1 nadir photo.

4 LIGHT-DEPTH ENVIRONMENT MAP

In this work we explore a new type of light field representation proposed in (Felinto et al., 2012), the *light-depth environment map*. This map contains both radiance and the spatial displacement (i.e., depth) of the environment light. The traditional approach for an environment map is to take it as a set of infinite-distant or directional lights. In this new approach the map gives information about the geometry of the environment, so we can consider it as a set of point lights instead of directional lights. This second approach results in a more powerful tool for rendering purposes, because most common environment maps



Figure 2: The upper picture shows the radiance channel of the environment and the lower one the depth channel used for reconstruct the light positions.

have their lights originally in a finite distance from the camera, such as an indoor environment map. With the depth we can reconstruct their original location and afford more complex and accurate lighting and reflection calculations. This enhanced map is no longer a map of directional lights but a conglomeration of lights points.

A *light-depth environment map* can be constructed from a *HDR environment map* adding the depth channel, as shown in figure 2. The depth channel can be obtained by a special render from the reconstructed environment meshes or also by scanning of the environment or other techniques. The production framework proposed in (Felinto et al., 2012) allow us to construct the *light-depth environment map* from the captured *HDR light map* using the modeled reconstruction of the scene.

For the following mathematic notations, a pixel sample from the *light-depth environment map* is denoted by $\mathbb{M}(\omega_i, z_i)$, where ω_i is the direction of the light sample in the map and the scalar value z_i denotes the distance of the light sample from the light space origin. The position of the light sample in light space is given by the point $z_i\omega_i$. We use the simplified notation $\mathbb{M}(\omega_i)$ to represent the radiance of the sample located along the ω_i direction in the *light-depth map*.

5 RENDERING PIPELINE FOR AUGMENTED SCENES

Our rendering system is based on the ray tracing algorithm. Ray tracing is based on following the path of a ray of light through a scene as it interacts with and bounces off objects in an environment. A ray tracer system must simulate at least the following objects and phenomena:

- **camera:** shows how and where the scene is viewed from. A camera generates rays from the viewing point into the scene.
- **shapes/primitives:** classifies the different types of objects that can be encountered in the scene by their properties (see section 5.1).
- **ray-objects intersections:** we must be able to compute precisely where a given ray pierces a geometric object and determine some additional properties such as the surface normal or its material.
- **light distribution:** A ray tracer must model the distribution of light throughout the scene, including their locations and the way in which they distribute their energy. In section 4 we presented a new type of light distribution that our system uses to improve the shadows and reflections accounts as will be discuss in section 6.
- **visibility:** to know whether a given light deposits energy at a point on a surface, we must know whether there is an uninterrupted path from the point to the light source (see section 5.2).
- **surface scattering:** each object has a description of its appearance, describing how light interacts with its surface, as well as the nature of the scattered light. The properties of the light that is scattered directly toward the camera is also important (see section 5.3).
- **recursive ray tracing:** because light can arrive at a surface after bouncing off or passing through several other surfaces, it is usually necessary to trace additional rays originating at the surface to fully capture this effect. We will discuss this aspect in sections 7 and 7.1.

To render the synthetic objects into a real scene in a photorealistic way, our system implements some of these task differently. We will discuss the aspects that differ from the traditional approach to lead the introduction of the rendering algorithm for augmented scenes.

5.1 Primitives: Synthetic, Support and Environment Surfaces

Our rendering system needs a special classification of the scene primitives, where each category defines different light scattering contributions and visibility tests.

- **Synthetic primitives:** the objects that are new to the scene. They don't exist in the original environment. Their light scattering calculation does not differ from a traditional rendering algorithm.
- **Support primitives:** surfaces present in the original environment that needs to receive shadows and reflections from the synthetic primitives. Their light scattering calculation is not trivial, because it needs to converge to the original lighting.
- **Environment primitives:** all the surfaces of the original environment that need to be taken into account for the reflections and shadows computations for the other primitive types. They don't require any light scattering calculation, because their color is computed directly from the *light-depth environment map*.

The level of detail of the environment reconstruction will depend on how you need the final render to be. For example, in a scene with no objects with glossy reflections, the environment mesh can be simplified to define only the main features that contribute with the lighting of the scene (e.g., windows and ceiling lamps) and a bounding box. Every ray starting at the light origin in world space must intersect with some primitive. We need to make sure the light field has the depth of all the rays. Thus it is important to model the environment mesh around all the scene without leaving holes/gaps.

5.2 Visibility

Each light contributes illumination to the point being shaded only if the path from the point to the light's position is unobstructed. In a ray tracer we trace a ray whose origin is at the surface point and whose direction points toward the light. If there is no blocking object between the light and the surface, the light contribution is included. These special rays are called *shadow rays*.

In augmented scenes, where we have *support* and *synthetic* objects, the traditional visibility test cannot

solve all types of interaction between the scene objects. Consider a point p on a *support* surface. When we are computing the light contribution for this point we do not consider obstructions given by other *support* object because all interaction between *support* objects are computed in the captured environment map. We only consider shadows generated by the *synthetic* objects and shadows casted over *synthetic* objects. To do this, we modified the visibility test to include the new special cases. We use a boolean variable *ignore_support* to turn on/off *support* objects during the visibility test. Table 1 shows the possible results for our visibility test for the possible values of the *ignore_support* variable and intersection types.

Environment primitives are ignored by the visibility test because they are associated with the *light-depth map* and used only for get correct computations of shadows and reflections (sections 6.1 and 6.2).

Table 1: Visibility test for augmented reality scenes, $VisTest(p, \omega, ignore_support)$.

First intersection Object type	<i>ignore_support</i>	
	true	false
<i>Synthetic</i>	false	false
<i>Support</i>	true	false
<i>Environment</i>	true	true

5.3 Surface Scattering: Estimating the Direct Lighting Integral

The scattering equation (1) says that exitant radiance $L_o(p, \omega_o)$ from a point p on a surface in direction ω_o is the sum of emitted radiance from the surface at the point plus the integral of incoming radiance over the sphere times the *BPDF* (Bidirectional Scattering Distribution Function) for each direction and a cosine term.

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\mathbb{S}^2} f(p, \omega_o, \omega_i) L_i(p, \omega_i) |\cos \theta_i| d\omega_i. \quad (1)$$

As our scene has only the light-depth environment map as direct light source (L_d) and do not contain other emissive objects (i.e $L_e(p, \omega_o) = 0$) the equation (1) becomes

$$L_o(p, \omega_o) = \int_{\mathbb{S}^2} f(p, \omega_o, \omega_i) L_d(p, \omega_i) |\cos \theta_i| d\omega_i. \quad (2)$$

To compute this estimate, we need to choose one or more directions ω_j and apply the Monte Carlo estimator:

$$L_o(p, \omega_o) \approx \frac{1}{N} \sum_{j=1}^N \frac{f(p, \omega_o, \omega_j) L_d(p, \omega_j) |\cos \theta_j|}{pdf(\omega_j)} \quad (3)$$

where $pdf(\omega_j)$ is the probability of sampling the ω_j direction.

The computation of $L_o(p, \omega_o)$ depends on the primitive type of the point p . Thus, for each type of primitive, we have a particular way to compute the direct lighting contribution

- **Synthetic primitives:** $L_o(p, \omega_o)$ is computed using the traditional Monte Carlo estimator (3);
- **Environment primitives:** direct lighting contribution is obtained directly from the *light-depth map* as

$$L_o(p, \omega_o) = \mathbb{M}(\omega_p / |\omega_p|, |\omega_p|), \quad (4)$$

where $\omega_p = WTL(p)$.¹

- **Support primitives:** $L_o(p, \omega_o)$ is computed by the estimator

$$\frac{1}{N} \sum_{j=1}^N \frac{\mathbb{M}(\frac{\omega_p}{|\omega_p|}, |\omega_p|) \cdot ES(p, n_p) \cdot \left\langle \frac{\bar{\omega}_j}{|\bar{\omega}_j|}, n_p \right\rangle}{p(\omega_j)}, \quad (5)$$

where $ES(p, n_p)$ denotes a scale factor term used instead the *BPDF* term $f(p, \omega_o, \omega_j)$. We dedicate the next section to show the derivation of the $ES(p, n_p)$ scale factor and the Monte Carlo estimator for support primitives.

5.3.1 Direct Lighting for Support Primitives

Support surfaces need to be rendered to include shadows and reflections from the synthetic objects. The rendered value of a support point p that doesn't have any contribution from the synthetic objects must converge to the radiance value stored in the *light-depth map* for its position. Thus we have that

$$L_o(p, \omega_o) = \mathbb{M}(\omega_p / |\omega_p|, |\omega_p|), \quad (6)$$

where $\omega_p = WTL(p)$ ¹, and the $L_o(p, \omega_o)$ term comes from the light scattering equation (2).

In order to get the equality from equation (6) we use a precomputed scale factor

$$ES(p, n_p) = \frac{\int_{S^2} Lum(\mathbb{M}(\omega_s, z_s)) d\omega_s}{\int_{S^2} Lum(\mathbb{M}(\omega_j, z_j)) \left\langle \frac{\bar{\omega}_j}{|\bar{\omega}_j|}, n_p \right\rangle d\omega_j}, \quad (7)$$

¹ $WTL(p)$ transform p from world space to light space.

where $\bar{\omega}_j = LTW(z_j \omega_j) - p$.²

The $ES(p, n_p)$ scale factor represents the percentage of light contribution coming from the map to the point p and is used in the Monte Carlo estimator of equation (2). Thus we have the Monte Carlo estimator for support primitives

$$L_o(p, \omega_o) = \frac{1}{N} \sum_{j=1}^N \frac{\mathbb{M}(\frac{\omega_p}{|\omega_p|}, |\omega_p|) \cdot ES(p, n_p) \cdot \left\langle \frac{\bar{\omega}_j}{|\bar{\omega}_j|}, n_p \right\rangle}{p(\omega_j)},$$

presented in the equation (5) in section 5.3

The $ES(v, n_v)$ value is estimated for all vertices v on *support meshes*. Thus, during rendering, we calculate the scale factor for a point $p = (a_1, a_2, a_3)$ in barycentric coordinates of the triangle primitive $T(v_1, v_2, v_3)$ as

$$ES(p, n_p) = a_1 \cdot ES(v_1, n_1) + a_2 \cdot ES(v_2, n_2) + a_3 \cdot ES(v_3, n_3).$$

5.3.2 Computing the BSDFs Scale Factor for Support Surfaces

The two integrals in (7) can be calculated by computing the contribution of each light point, summing over the pixels of the entire *light-depth map*. This process is computationally expensive due to the size of the map (about 8k or higher). To simplify the accounts, the map is discretized in a limited set of point lights whose radiance is equivalent to the total radiance of the whole map. So, instead of summing over all pixels of the map, we compute the sum on the small group of point lights to get a quick estimate of the illumination.

The discretization of the *light-depth map* is made using the median cut algorithm described in (Debevec, 2005). Computing the equation (7) over the discretized representation of the *light-depth map* by a set $L_d(i)$ of K point lights we have

$$ES(p, n) = \frac{\sum_{i=1}^K L_d(i)}{\sum_{i=1}^K L_d(i) \langle \omega_i, n \rangle}, \quad \text{with } \langle \omega_i, n \rangle > 0, \quad (8)$$

where

$$\omega_i = \frac{LTW(L_d(i)) - p}{|LTW(L_d(i)) - p|}.$$

² $LTW(z_j \omega_j)$ transform $z_j \omega_j$ from light to space world space.

6 RENDERING BASED IN LIGHT-DEPTH MAPS

The main deficiency in traditional rendering algorithms based in environment maps is that all light scattering calculations are performed using the environment map as a set of directional lights. This approach has the drawback that the environment map must be captured in the same position where the synthetic objects are inserted into the real scene, as did Debevec in his work (Debevec, 1998). If we need to introduce several objects in different positions of the scene, usually the positions of the shadows and reflections are wrong in the final render, as shown in figure 8, because the spatial location of the light is not considered in the directional approach.

Our proposal is based on modeling and synthesize the scene using a single large resolution environment map as input. This map is used for rendering the background, apply textures and model the environment geometry to keep photo-realistic light scattering effects in the final augmented scene to be rendered. In a parallel work (Felinto et al., 2012) we propose a production framework for render full panoramic scenes with augmented reality. This framework plus the rendering solution we are proposing make possible to add synthetic objects into a real scene using full panoramic cameras or conventional perspective cameras for rendering the output image. We will discuss now rendering aspects that differ from the traditional environment approach in the light-depth based rendering.

6.1 Computing Shadows

For every camera ray that intersects with the scene at a point p on a surface, the integrator takes a light sample ω_i by importance from the *depth-light map* to compute the direct light contribution for point p . Next, the renderer performs a visibility test to determine if the sampled light is visible or not from the point p .

In the traditional rendering scheme, that uses environment maps as directional lights, the visibility test is computed for the ray $r(p, LTW(\omega_i))$, with origin in p and direction $LTW(\omega_i)$ with ω_i in world space. This approach introduces several errors when the object is far away from the point where the light map was captured. Note that all the shadows will have the same orientation, given that only the direction of the light sources is considered.

Our rendering algorithm works differently. Thanks to the light-depth environment map properties, the visibility test uses the light sample world position and not only its direction.

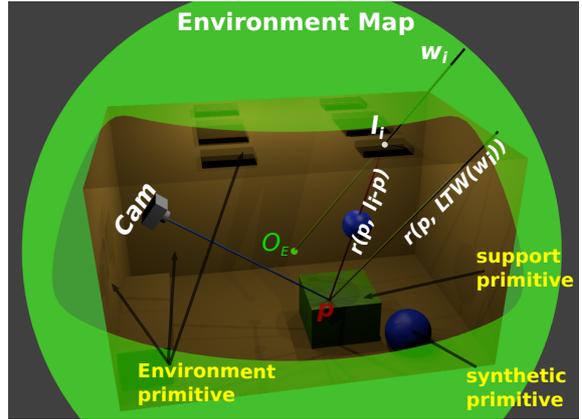


Figure 3: The visibility test used by the *ENVPath* integrator. The ray $r(p, LTW(\omega_i))$ used by the directional approach is not occluded so the light l_i contributes for the scattering account of point p . The ray $r(p, l_i - p)$ used by our light-positional approach is occluded by a synthetic object and gets the correct world-based estimation.

The new visibility test use the direction ω_i of the light sample and multiply it by their depth value z_i obtaining the point $z_i \cdot \omega_i$ in the coordinate system of the light. The point $z_i \cdot \omega_i$ is transformed to world space to obtain $l_i = LTW(z_i \cdot \omega_i)$. Thus the calculation of visibility is made for the ray $r(p, l_i - p)$, with origin p and direction $l_i - p$ as shown in figure 3.

6.2 Computing Reflections

Given a point p on a surface, the integrator takes a direction ω_i by sampling the surface *BRDF* (Bidirec-

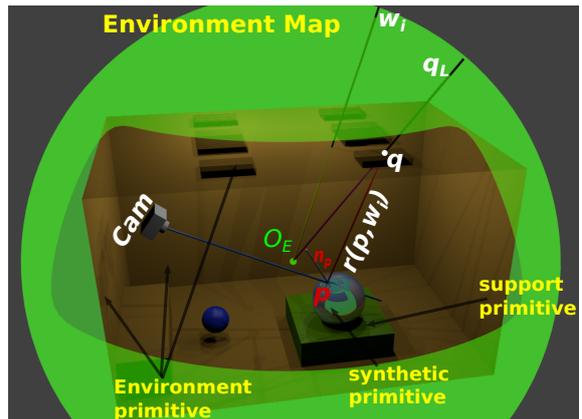


Figure 4: Reflection ray path account used by the *ENVPath* integrator. As shown, the ray $r(p, \omega_i)$ intersects the scene at point q , over the environment mesh. The radiance of q is stored at q_L direction in the environment map. Our approach computes the correct reflection q_L , instead of computing the ray ω_i that would give a wrong reflection mapping.

tional Reflectance Distribution Function) to add reflection contributions to the point p . To do this, we compute the scene intersection of the ray $r(p, \omega_i)$ with origin p and direction ω_i . Note that the intersection exists because the environment was completely modeled around the world. If the intersection point q is on a synthetic or support surface its contribution must be added to reflection account. Otherwise, if the intersection is with an environment mesh, we transform q from world space to light space by computing $q_L = WTL(q)$. Finally the contribution given by the direction q_L in the environment map is added to the reflection account. The figure 4 illustrates this proposal.

6.3 Environment Texture Coordinates

Environment texture coordinates was implemented to support rendering effect such as the deformation in the original environment by the interaction with synthetic objects. Another application is to copy a texture from one environment region to another. This can be a powerful tool for texturing support meshes on regions where do you don't know the color directly by the environment because they are occluded by other support object. In figure 5 we show the power of this tool to apply deformations on support objects when they interact with synthetic objects.



Figure 5: The two spheres on the couch were added to the scene. The couch is real and has been stored on the light-depth map. We deformed the *environment texture coordinates* to simulate the interaction between the spheres and the couch.

7 LIGHT TRANSPORT THEORY AND PATH TRACING ALGORITHM

To understand the modified path tracing algorithm that we are proposing for augmented scenes it is necessary understand first the *light transport equation* -

LTE (9), (Kajiya, 1986), that describes the phenomena of global interaction between the light sources and surfaces of the scene.

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\mathbb{S}^2} f(p, \omega_o, \omega_i) L(t(p, \omega_i), -\omega_i) |\cos \theta_i| d\omega_i. \quad (9)$$

Using a special form of the *LTE*, a sum over light carrying paths of different lengths, we can describe the incident radiance at a point p_0 from another point p_1 , where p_1 is the first point on a surface along the ray from p_0 in direction $p_1 - p_0$:

$$L(p_1 \rightarrow p_0) = L_e(p_1 \rightarrow p_0) + \int_A L_e(p_2 \rightarrow p_1) f(p_2 \rightarrow p_1 \rightarrow p_0) G(p_2 \leftrightarrow p_1) dA(p_2) + \int_A \int_A L_e(p_3 \rightarrow p_2) f(p_3 \rightarrow p_2 \rightarrow p_1) G(p_3 \leftrightarrow p_2) f(p_2 \rightarrow p_1 \rightarrow p_0) G(p_2 \leftrightarrow p_1) dA(p_3) dA(p_2) + \dots$$

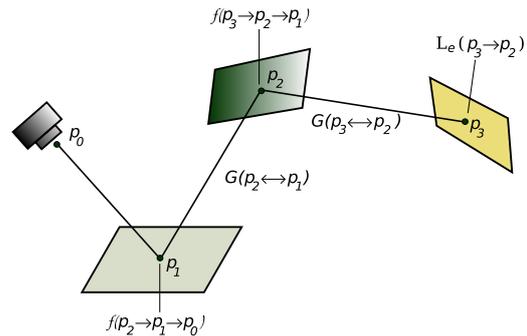


Figure 6: The integral over all points p_2 and p_3 on surfaces in the scene given by the light transport equation gives the total contribution of two bounce paths to radiance leaving p_1 in the direction of p_0 . The integrand components are: the emitted radiance from the light, L_e ; the geometric terms between vertices, G ; and scattering from the *BSDFs*, f .

Each term on the right of this equation represents a path of increasing length. For example, the third term is illustrated in figure 6. This path has four vertices, connected by three segments. The total contribution of all such paths of length four is given by this term. The infinite sum can be written compactly as

$$L(p_1 \rightarrow p_0) = \sum_{i=1}^{\infty} P(\bar{p}_i). \quad (10)$$

$P(\bar{p}_i)$ gives the amount of radiance scattered over path \bar{p}_i with $i + 1$ vertices, $\bar{p}_i = (p_0, p_1, \dots, p_i)$ where p_0 is on the film plane and p_i is on the light source. So

$$P(\bar{p}_i) = \underbrace{\int_A \cdots \int_A}_{i-1} L_e(p_i \rightarrow p_{i-1}) T(\bar{p}_i) dA(p_2) \cdots dA(p_i),$$

where

$$T(\bar{p}_i) = \prod_{j=1}^{i-1} f(p_{j+1} \rightarrow p_j \rightarrow p_{j-1}) G(p_{j+1} \leftrightarrow p_j)$$

describes the fraction of radiance from the light source that arrives at the camera after all of the scattering at vertices between them.

Given equation (10) and a particular length i , all that we need to do to compute a Monte Carlo estimate of the radiance arriving at p_0 due to paths of lengths n is to sample a set of vertices with an appropriate sampling density in the scene to generate a path and then evaluate an estimate of $P(\bar{p}_i)$ using those vertices. More details about the *light transport equation* can be found in (Pharr and Humphreys, 2010).

7.1 ENVPath Integrator: Path Tracing for Light-Depth Environment Maps

We use an implementation of the path tracing algorithm that solves the light equation constructing the path incrementally, starting from the vertex at the camera p_0 . At each vertex p_k , the *BSDF* is sampled to generate a new direction; the next vertex p_{k+1} is found by tracing a ray from p_k in the sampled direction and finding the closest intersection.

For each vertex p_k with $k = 1, \dots, i-1$ we compute the radiance scattering at point p_k along the $p_{k-1} - p_k$ direction. The scattering computation for p_k on a synthetic, support or environment surface is performed using respectively the equations (3), (4) and (5) from section 5.3.

A path $\bar{p}_i = (p_0, \dots, p_i)$ can be classified according the nature of its vertices as

- **Real path:** the vertices p_k , $k = 1, \dots, i-1$ are on support surfaces.
- **Synthetic path:** the vertices p_k , $k = 1, \dots, i-1$ are on synthetic objects surfaces.
- **Mix path:** some vertices are on synthetic objects and other on support surfaces.

There is no need to calculate any of the real path for the light. They are already present in the original photography. However, since part of the local scene was modeled (and potentially modified), we need to re-render it for these elements. The calculated radiance needs to match the captured value from the environment. We do this by aggregating all the real path

radiance in a vertex p_1 as direct illumination, discarding the need of considering the neighbor vertices light contribution.

The synthetic and mixed light paths are calculated in a similar way, taking the individual path vertex light contributions for every vertex of the light path. The difference between them is in the Monte Carlo estimate applied in each case. In the figure (7) you can see the different light path types and the calculation that happens on the corresponding path vertices.

Because we are using an incremental path construction technique, it's impossible to know the kind of path we have before we are done with it. Thus the classification above is only theoretical.

For implementation purposes during the incremental path construction we only need to know if the partial path (p_0, \dots, p_k) contains some synthetic vertices or not. This allows us to decide the Monte Carlo estimator to use. To do this, we use a boolean variable pt (*path type*) that has a *false* value if the partial path doesn't have synthetic vertices and *true* otherwise.

Algorithm 1 ENVPath Integrator

```

1:  $L := 0$  ▷ Radiance contribution
2:  $r_0 := ray(p_0, -\omega_0)$  ▷ Camera ray
3:  $pt := false$  ▷ Path type
4:  $T := 1$  ▷ Path throughput factor
5: for  $i = 0$  to  $MaxLength$  do
6:    $p_{i+1} = SceneIntersect(r_i)$ 
7:    $T* = ThroughputAttenuation(p_i, p_{i+1})$ 
8:   if  $type(p_{i+1}) = ENVIRONMENT$  then
9:     if ( $pt = true$ ) OR ( $i = 0$ ) then
10:       $L+ = \mathbb{M}(\frac{\omega_{p_{i+1}}}{|\omega_{p_{i+1}}|}, |\omega_{p_{i+1}}|)$  ▷ eq. (4)
11:      break ▷ Terminate the path
12:   if  $type(p_{i+1}) \neq SUPPORT$  then
13:      $pt := true$ 
14:   if ( $i+1$ ) =  $MaxLength$  then
15:     break ▷ Terminate the path
16:    $L+ = DirectLighting(p_{i+1}, pt, i)$ 
▷ Sample BSDF to get new path direction
17:    $r_{i+1} := SampleNewDirection(p_{i+1})$ 
18:    $T* = f(p_{i+1}, -dir(r_i), dir(r_{i+1}))$ 
19:   ▷ Continues the path construction or not ?
20:   if  $Prob() < END\_PROBABILITY$  then
21:     break ▷ Terminate the path
22:   else
23:      $T / = Prob()$  ▷ Increase path contribution
24:   if  $type(p_{i+1}) = SUPPORT$  then
25:      $T* = SupportAttenuation(p_i, p_{i+1})$ 
26: return  $L$ 

```

Algorithm 2 Direct Lighting Estimator

```

1: Procedure DirectLighting( $p, \omega_o, PT, RD$ )
2:  $Ld := 0$ 
3:  $mis := UseMultipleImportanceSampling(p)$ 
4:  $\omega_i := SampleLightDir(\mathbb{M})$ 
5:  $\omega_p := LightToWorld(\omega_i)$ 
6: if type( $p$ ) = SUPPORT then
7:   if ( $PT = true$ ) OR ( $RD = 0$ ) then
8:     if ( $mis = false$ ) OR ( $RD = 0$ ) then
9:       if VisTest( $p, \omega_i, true$ ) = true then
10:         $Ld+ = \frac{ES(p, n_p) \mathbb{M}(\omega_p)(\omega_p, n_p)}{pdf(\omega_i)}$ 
11:      else
12:        if VisTest( $p, \omega_i, false$ ) = true then
13:           $Ld+ = \frac{ES(p, n_p) \mathbb{M}(\omega_p)(\omega_p, n_p)}{pdf(\omega_i)}$ 
14:           $Ld+ = \frac{f(p, \omega_o, \omega_p) \mathbb{M}(\omega_p)(\omega_p, n_p)}{pdf(\omega_i)}$ 
          ▷ Trace a 2nd shadow ray by sampling the
          BSDF
15:        if  $mis = true$  then
16:           $\omega_i := SampleBSDFDir(p)$ 
17:           $r := ray(p, \omega_i)$ 
18:           $q := SceneIntersect(r)$ 
19:          if type( $q$ ) = ENVIRONMENT AND
           $DR > 0$  then
20:             $Ld+ = \mathbb{M}(\omega_p)$ 
21:        else ▷ type( $p$ ) = SYNTHETIC
22:          if VisTest( $p, \omega_i, false$ ) = true then
23:             $Ld+ = \frac{f(p, \omega_o, \omega_i) \mathbb{M}(\omega_i)(\omega_i, n_p)}{pdf(\omega_i)}$ 
            ▷ Trace a 2nd shadow ray by sampling the
            BSDF
24:          if  $mis = true$  then
25:             $\omega_i := SampleBSDFDir(p)$ 
26:             $\omega_p := LightToWorld(\omega_i)$ 
27:             $r := ray(p, \omega_i)$ 
28:             $q := SceneIntersect(r)$ 
29:            if type( $q$ ) = ENVIRONMENT then
30:               $Ld+ = \mathbb{M}(\omega_p)$ 
31: return  $Ld$ 
32: end procedure

```

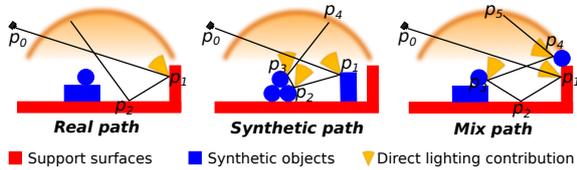


Figure 7: Path type classification. The yellow cones represent the direct lighting contribution from the vertices p_i that contributes with the illumination accounts.

8 RESULTS

The rendering performance of our approach is equivalent to the rendering of a normal scene with



Figure 8: Comparison between reflections given by directional (upper) and light-depth (lower) maps approaches.

the same mesh complexity using a physically based light rendering algorithm. There was no visible loss in that regard. Part of the merit of this, is that this is an one-pass solution. There is no need for multiple composition passes. The rendering solution converges to the final results without "adjustments" loops been required like the original Debevec's approach (Debevec, 1998).

The mirror ball reflection calculation can be seen in more details in the figure 8. This is a comparison between the traditional direction map method and our light-depth solution. The spheres and the carpet are synthetic and render the same with both methods. The presence of the original environment meshes makes the reflection to be a continuous between the synthetic (e.g., carpet) and the environment (e.g., wood floor).

As explained during the paper, the core aspect of our process is to develop the rendering of full panorama images. More specifically the integration of a captured environment and synthetic objects. In the figure 9 you can see synthetic spheres integrated with the original panorama. The lighting and shadows on the floor are calculated with the light-depth map.

The proper calibration of the scene and the correct shadows helps the natural feeling of belonging for the synthetic elements in the scene. In figures 11 and 12 you can see the spheres and the carpet inserted in the scene. The details are shown in a non panoramic frus-



Figure 9: Rendering of a full panoramic scene with synthetic objects. All the balls and the doll in the scene are synthetics. The carpet on the floor was synthesized to cover a table that is in the original environment capture.



Figure 10: The images show the scene from figure 9 rendered with fisheye lens.



Figure 12: Shadows and reflections computed using the *light-depth environment map*. It's possible to see how the highlights and shadows vary with the position of the balls in the scene.

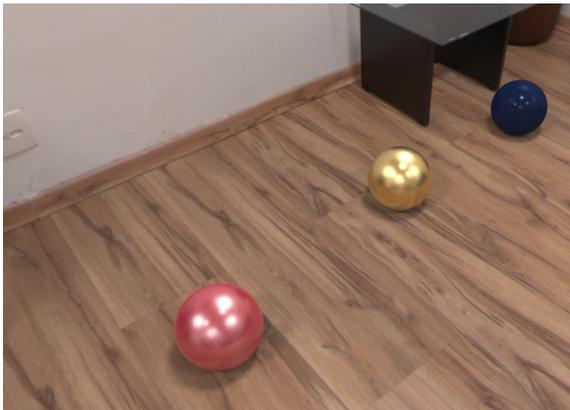


Figure 11: Shadows computed using the *light-depth environment map*. The orientation of the shadows varies respecting the position of the balls in the scene. Look the difference between shadows of the red and the blue balls.



(a)

(b)

Figure 13: Detail of the camera traveling effect. (a): the camera is positioned where the environment was captured. (b): The camera position is displaced from the environment origin. A wall partially visible in (a) is occluded in (b).

tum to showcase the correct perspective when seen in a conventional display.

Finally, we explored camera traveling for a few of our shots. In the figure 13 you can see part of the scene rendered from two different camera positions. The result is satisfactory as long as the support environment match is properly modeled. For slight camera shifts this is not even a problem.

9 CONCLUSION

We have presented a novel approach for rendering synthetic objects into full panoramic scenes based on illumination captured from real world and reconstructed depth. To show the feasibility of such, a production workflow is essential to produce realistic immersive scenes for panoramic displays.

A key element of this work is the introduction of light-depth environment map for rendering of synthetic elements in an augmented scene. This solution, until now unexplored for this means, uses the lights positions in the environment map instead of the directional approach to get the correct shadows and reflections effects for all synthetic objects along the scene.

Among the possible improvements, we are interested on studying techniques to recover the light positions for assembling the light-depth environment map and semi-automatic environment mesh construction for the cases where we can capture a point cloud of the environment geometry.

An extension of this work is the combination of a low resolution light-depth environment map, for example captured with a mirror ball, and a background plate captured with a conventional camera. That is aimed at conventional non-panorama augmented scenes that require more accurate accurate lighting, shadow and reflection calculations.

Another path to explore is the use of more than one single environment capture to allow more complex camera traveling in the scene. The use of multiple environment captures also helps to reduce the occluded regions (i.e., blind spots, noticeable in reflections). This is important to avoid the abuse of copying environment texture to complete unknown environment elements.

Finally, we are accomplished by the results and the contributions we add to the panorama and augmented rendering universes. Even though the movie industry has well established commercial pipelines, they can be of hard access for production with more restricted budgets, or simply closed behind doors due to NDAs and patented technologies. This work, on the other hand, praises on its replicability.

A working full functional implementation is already available in our site and in the *Luxrender* repository. And additionally to the renderer, we made available the plugin developed to help with the necessary modeling for this framework - built on top of the 3D suite, Blender. We hope that by encouraging the use of free and accessible software we help both the artistic and the scientific communities to benefit from adapting and incorporating this work in their pipelines.

REFERENCES

- Bourke, P. D. and Felinto, D. Q. (2010). Blender and immersive gaming in a hemispherical dome. In *GSTF International Journal on Computing (JoC)*, Vol. 1, No. 1, ISSN: 2010-2283.
- Debevec, P. E. (1998). Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. pp. 189-198. SIGGRAPH.
- Debevec, P. E. (2005). A median cut algorithm for light probe sampling. SIGGRAPH Poster.
- Debevec, P. E. and Malik, J. (1997). Recovering high dynamic range radiance maps from photographs. pp. 369-378. SIGGRAPH.
- Felinto, D. Q. (2010). Domos imersivos em arquitetura. In *Bachelor thesis at the Escola de Arquitetura e Urbanismo*, Niterói. Universidade Federal Fluminense.
- Felinto, D. Q., Zang, A. R., and Velho, L. (2012). Production framework for full panoramic scenes with photo-realistic augmented reality. In *XXXVIII Latin American Conference of Informatics (CLEI)*, Medellin.
- K. Karsch, V. Hedau, D. F. and Hoiem, D. (2011). Rendering synthetic objects into legacy photographs. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, pp. 157:1-157:12. ACM, New York, NY, USA.
- Kajiya, J. T. (1986). The rendering equation. pp. 143-150. SIGGRAPH.
- Miller, G. S. and Hoffmann, C. R. (1984). Illumination and reflection maps: Simulated objects in simulated and real environments. In *Curse Notes for Advanced Computer Graphics Animation*. SIGGRAPH.
- Pharr, M. and Humphreys, G. (2010). *Physically Based Rendering: From theory to Implementation*. Morgan Kaufmann Publishers, Burlington, USA, 2nd edition.
- Williams, L. (1983). Pyramidal parametrics. In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, vol 17(3), pp. 1-11, ISBN:0-89791-109-1. ACM-SIGGRAPH.
- Zang, A. R. and Velho, L. (2011a). Arluxrender project. <http://www.impa.br/~zang/arlux>. Visgraf.
- Zang, A. R. and Velho, L. (2011b). Um framework para renderizações foto-realistas de cenas com realidade aumentada. In *XXXVII Latin American Conference of Informatics (CLEI)*, Quito.