Production framework for full panoramic scenes with photo-realistic augmented reality

Dalai Felinto*, Aldo René Zang† and Luiz Velho†
* Fisheries Centre, University of British Columbia, http://www.fisheries.ubc.ca
† Visgraf Laboratory, Instituto de Matemática Pura e Aplicada, http://www.visgraf.impa.br
Email: d.felinto@fisheries.ubc.ca, zang@impa.br, lvelho@impa.br

***Abstract*—The novelty of our workflow is the end-to-end solution to combine captured panoramas and computer generated elements. This pipeline supports productions specially aiming at spherical displays (e.g., fulldomes). 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.**

***Index Terms*—full panorama, photorealism, augmented reality, photo-realistic rendering, hdri, light-depth, ibl illumination, 3d modeling**

I. INTRODUCTION

The realm of digital photography opens the doors for the artist work with (pre/post)filters and other artifices where long gone are the days constrained by the physical nature of the film - ISO, the natural lighting, and so on. As a creative artist we can't stand to merely capture the environment that surround us. We urge the interfere with our digital carving toolset. There comes the computer, guided by the artist eyes to lead this revolution. The photo-video production starts and often ends in its digital form. Ultimately those are pixels we are producing. And as such, a RGB value is just as good whether it comes from a film sensor or a computer render. But in order to merge the synthetic and the captures elements, we need to have them in the same space. Since we can't place our rendered objects in the real world, we teleport the environment inside the virtual space of the computer. Thus the first part of our project is focused on better techniques for environment capturing, and reconstruction.

We also wants to work with what we believe 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 Géode, in Paris, France). In 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, gyroscopic based mobile panorama viewing apps?

All this is producing a boom in the demand for panorama productions. And with these, the opportunity to address new problems by the computer graphic industry.

Our project started motivated by these new airs. 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.

The first 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.

Our proposal for this problem is a framework for photo-realistic panorama rendering of synthetic objects inserted in a real environment. We chose to generate the depth of the relevant environment geometries to a complete simulation of shadows and reflections of the environment and the synthetic elements. We are calling this a *light-depth environment map*.

As part of the work we extended the open source software *ARLuxrender* [1] [2], which allows for physic based lighting rendering of virtual scenes. Additionaly an addon for the 3d open source software *Blender* was developed. All the tools required for the presented framework are open source. This reinforce the importance of producing authoring tools that are accessible for artists worldwide and also present solutions that can be implemented elsewhere, based of the case-study presented here. Additional information about this framework can be found in the *ARLuxrender* project web site [2].

II. RELATED WORK

The first attempts of doing environment reflection date from the early 80ies with the work of G. Miller [3] and M. Chou [4]. 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.

Debevec and Malik developed a technique of recovering high dynamic range radiance maps from photography taken with conventional camera, [5]. Many shots are taken with a variance of the exposure camera settings 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 iluminate synthetic objects with lighting captured from the real world, [6]. 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, [1]. In the same year (2011), Karsch presented a relevant work on rendering of synthetize objects in legacy photographies, using a different method for recovering the light sources of the scene, [7].

As for panoramic content creation important work was developed in the past for realtime content in domes by Paul Bourke [8]. Part of our work is inspired on his multiple ideas on full dome creation, and the work of Felinto in 2009, implementing Bourke's realtime method in Blender and proposing other applications for panorama content in architecture visualization, [9].

III. Our Framework

We can describe our workflow for the creation of augmented reality scenes with the following steps:

1) Environment capture and panorama assembling
   - Displacement of the camera in strategic spots
   - Capture of photos to fill a 360°x360° field of view
   - Stack the individual HDR images
   - Stitching of the panorama image
2) Calibration of the equirectangular panorama
3) Environment modeling
4) Movie making
   - Modeling of the new objects
   - Camera animation touch ups
5) Final rendering
   - Rendering of the augmented reality panorama
   - Rendering of video with restricted frustrum

IV. Environment Capture

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 [5]. In total we took 9 pictures per camera orientation with the total of 7 different orientations to obtain the total of a 360° x 360° 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.

We based our panorama capturing on the study presented by Kuliyev [10] which presents different techniques, not discussed in the current project. Additionally, there is equipment in the market specially target for the movie making industry, not discussed in Kuliyev's work. Expensive all-in-one solutions can be found in Hollywood for capturing of the environment light field and geometry (e.g., point clouding). For the purposes of this project we aimed at more affordable solutions within our reach and more accessible to a broader audience.

For future projects we consider the possibility of blending panorama video and panorama captured photography. Our workflow with some considerations is as follow:

A. Photography capture

A common solution for light field capturing - mirrorball pictures - was developed to be used as reflection map and lighting [5]. However, when it comes to background plates, our earlier tests showed that a mirrorball 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 photographies to assemble in the panorama.

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 realtime applications) 6 pictures generated with a 90° frustum camera oriented towards a cube are sufficient to recreate the environment with a good tradeoff between render size and pixel distortion [8].

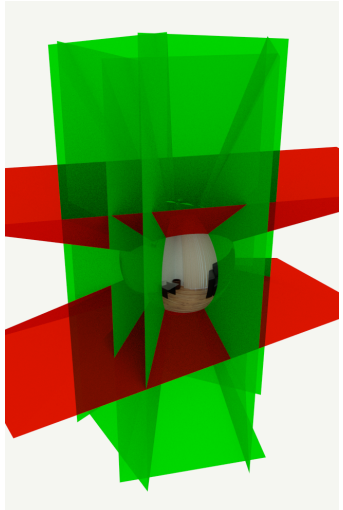
We chose a fisheye lens (Nikon 10.5 mm) to increase the field of view per picture and minimize the number of required shots. We used 7 pictures in total, including the north and south pole (see figure 1).

B. Nadir - south pole

The south pole is known for its penguins and panorama stitching problems. Many artists consider the capture of the *nadir* optional, given that the tripod occludes part of the environment in that direction. In fact this can be solved by the strategical placement of synthetic elements during the environment modeling reconstruction (see VI-C). To produce a complete capture of the environment we took the nadir picture without the tripod - hand-helding the camera. This can introduce anomalies and imprecisions in the panorama stitching.

To minimize this problem, we masked out the *nadir* photo to contain only the missing pixels from the other shots. To give a better estimative on what to mask we stitched a panorama without the south pole, and created a virtual fisheye lens to

(a) Manfrotto Pano Head, Nikon DX2S camera and Nikon 10.5mm fisheye lens.



(b) For Nikon 10.5mm: 5 photos around z-axis, 1 zenith photo and 1 nadir photo.

Fig. 1. (a): Equipment used to capture the full environment. (b): Space partition used for assembly the panorama.

emulate the DX2S + 10.5mm lens. The fisheye lens distortion can be calculated with the equisolid fisheye equation (1):

$$FOV_{equisolid} = 4 \cdot arcsin \cdot (\frac{framesize}{focallength \cdot 4}) \qquad (1)$$

In the figure 2 you can see the comparison between a fisheye image taken from the camera and a fisheye render captured from the spherical panorama with the virtual lens described above. This implementation was done in *Cycles*, a full raytracer renderer of *Blender*.

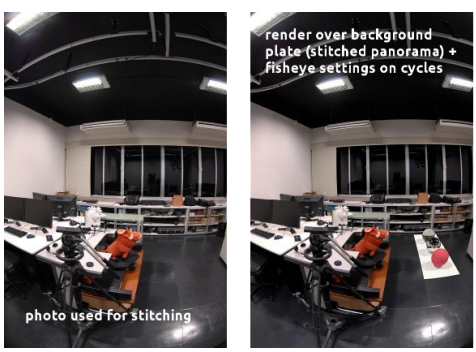


Fig. 2. Left: the real photgraph of the scene, Right: image rendered with cycles using the spherical panorama with virtual lens.

V. Panorama Calibration

An equirectangular panorama is a discretization of a sphere in the planar space of the image. As such, there is an implicit but often misleading orientation of the representative space. The top and bottom part of the images represent the poles of the sphere. The discretized sphere, however is not necessarily aligned with the real world directions. In other words, the north pole of the sphere/image may not be pointing to the world's zenith.

A second issue is the scale of the parametric space. The space represented in the equirectangular panorama is normalized around the camera point of view. In order to reconstruct the tridimensional space, one needs to anchor one point in the represented space where the distance is known. For pictures not generated under controlled conditions, it can be hard to estimate the correct scale. This is not a problem as far as reconstruction of the environment goes. Nonetheless a high discrepancy in the scale of the scene should be considered for physic simulations, real world based lighting and artificial elements introduced with their real world scale in the scene.

The decision on introducing a calibration step was also to add more freedom to the capturing and obtaining of the panorama maps. The system can handle images with wavy horizons just as good as with pole oriented/aligned images.

Extra advantages of a calibration system:

1) Allow to move important sampling regions off the poles.
2) Less concern on tripod alignment for picture capturing.
3) It works with panoramas obtained from the internet.
4) Optimal aligned axis for the world reconstruction.

A. Horizon Alignment

In order to determine the horizontal alignment of a panorama we choose to locate the horizon through user input on known elements. We start of by opening the panorama in its image space (mapped in a ratio of 2:1) and have the user to select 4 points that represent the corners of a rectangular shape in the world space that is placed on the floor (see figure 3).

The selected points define vectors whose cross products determine the **x** and **y** axis of the horizontal plane. The cross product of **x** and **y** determines the **z** axis (connecting the south and the north poles). That should be enough to determine the global orientation of the image, but the manual input introduces human error to the calculation of the **x** and **y** axis. Instead of using the inputed data directly, we recalculate the **y** axis as the cross product of the **z** and the **x** axis.

$$\vec{v_x} = (p_0 \times p_1) \times (p_3 \times p_2)$$
$$\vec{v_y} = (p_1 \times p_2) \times (p_0 \times p_3)$$
$$\vec{v_z} = \vec{v_x} \times \vec{v_y}$$
$$\vec{v_y} = \vec{v_z} \times \vec{v_x}$$

To assure the result is satisfactory we reproject the horizon line and the axis in the image to provide visual feedback for user fine tuning of the calibration rectangle.

The rectangle chosen for the calibration define the world axis and the floor plane alignment. This helps the stages of reconstruction of the existent world and the 3d modeling of new elements.

B. World Scaling

Once the orientation of the map is calculated we can project the selected rectangle into the 3d world floor. We need,

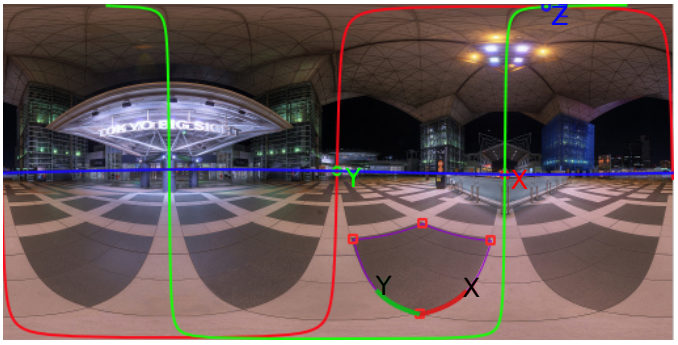


Fig. 3. Calibration system. The four points on floor determines the orientations axis for the environment. The blue, red and green lines determines the xy (horizon), xz and yz slice planes.

however, to gather more data - as the orientation alone is not sufficient. In fact any arbitrary positive non-zero value for the camera height will produce a different (scaled) reconstruction of the original geometry in the 3d world. For example, if the supposed original camera position is estimated bigger than the position the tripod had when the pictures were taken, the reconstructed rectangle will be bigger than its original counterpart.

We have a dual system with the rectangle dimensions and the camera height. Thus we leave to the user to decide which one is the most accurate data available - the camera height or the rectangle dimensions. The rectangle dimensions are not used in the further reconstruction operations though. Instead we always calculate the camera height for the given input parameter (width or height) and use it to calculate the other rectangle side as well (height or width, respectively).

C. Further Considerations

There are other possible calibration systems that were considered for future implementations. In cases where no rectangle is easily recognized, the orientation plane can be defined by independent converging axis. In architecture environments it is common to have easy to recognize features (window edges, ceiling-wall bound) to use as guides for the camera calibration.

The adopted solution is a plane-centric workflow. The calibration rectangle does not need to be on the ground. It can just as well be part of a wall or the ceiling. In our production set we had more elements to use as reference keys on the floor, thus the preference on the implementation. We intend, for future projects to explore the flexibility of this system though. Nonetheless this is also the reason why the environment is reconstructed from the ground up, as described in the VI section.

VI. Environment Modeling

There are different reasons for the environment to be modeled. With the full reconstruction of captured space in the panorama we can move the camera freely around and have reflexive objects to bounce the light with the correct perspectives. Nevertheless, the equirectangular image is not a direct 3d map of the environment. The panorama is one of other possible parameterizations of the sphere. It lacks information to reproject the space beyond the two dimensions of the sphere surface. If we can estimate the depth of the image pixels we can reconstruct the original represented space. Therefore we can render more complex light interactions such as glossy reflections and accurate shadow orientation.

The environment meshes serve multiple purposes in our work: (a) In the rendering stage, the environment is used to compute the light position in world space for the reflection rays (see VII-F); (b) The scene depth needs to be calculated and stored with the HDR needed by the render integrator to pre-computate the light scale factor of the support surfaces and resolve visibility tests (see VII-A, VII-E and VII-D); (c) Part of the modeled environment serves as support surfaces for the shadow and reflection rendering of the synthetic elements in the assembled panorama (see VII-B); (d) Modeled elements can be transformed while keeping their reference to how they map to the original environment to produce effects such as environment deformation and conformation to the synthetic elements (see VII-G and VI-D); (e) Finally, in the artistic production it's important to have structural elements for the physic animation/simulations and for visual occlusion of the synthetic elements.

The scene is reconstructed in parts. Due to the implications of the implemented calibration system, the floor is the most known region. Therefore we start modeling the object projections on the floor plane.

A. Floor reconstruction

A system with the floor region defined in the image space and extra structuring points helps the building of basic geometric elements. Two points can be used to define the corners of a square. Three points can delimitate the perimeter of a circle or define a side and the height of a rectangle.

1) Square: The making of a square by its corners is a problem of vectorial math in the reconstructed 3d space. It's convenient for the 3d artist to have a canonical square defined in the local space, while determining the square size, location and rotation in the world space. Thus we use the two selected points of the image to determine the length of the square diagonal which will be used to assign the square world scale.

$$\vec{v_{diagonal}} = p_1 - p_0$$
$$p_{center} = p_0 + \frac{\vec{v_{diagonal}}}{2}$$
$$\alpha = \vec{v_{diagonal}} \cdot \vec{v_{(1,1,0)}}$$

2) Circle: The canonical formula of the circle is defined by its center and its radius. Only in few cases you will have the center visible in the panorama though. Instead we implemented a circle defined by three points of the circumference. Even if the circle is partially occluded this method can be successfully

applied.

$$
\begin{aligned}
c_0 &= \frac{p_0 + p_1}{2}, c_1 = \frac{p_1 + p_2}{2} \\
\vec{v_0} &= (p_0.y - p_1.y, p_1.x - p_0.x, 0) \\
\vec{v_1} &= (p_1.y - p_2.y, p_2.x - p_1.x, 0) \\
p_{center} &= c_0, \vec{v_0} \cap c_1, \vec{v_1} \\
radius &= ||p_0 - \vec{p}_{center}||
\end{aligned}
$$

3) Rectangle: Objects that have a parallelogram geometry (for example table feet, chests, boxes) can be rebuild with a rectangular base projected on the floor. Unless their material is transparent, or only structural (for example, wires), they will have at least one of their corners occluded by its own tridimensional body. In this case three points will have to be used to reconstruct the rectangular base. In the pure mathematical sense, three points can define a rectangle in different ways. For instance, if they define three corners the fourth vertex can be inferred from the angles defined between the existent vertices. However, it is hard to rely on user input to precisely reconstruct the right angle intrinsic of the rectangle shape. Thus we have the user defining one of the sides of the rectangle through the two first points and setting the height with the third point. This way, even if the point doesn't produce a perfect right angle when projected on the floor with the defined rectangle side we can use it to calculate the distance between the opposite sides of the rectangle and ensure a perfect reconstruction of the shape.

$$
\begin{aligned}
\vec{v_{side}} &= p_1 - p_0 \\
width &= ||\vec{v_{side}}|| \\
height &= DistancePointLine(p_2, p_0, \vec{v_{side}}) \\
c_0 &= \frac{p_0 + p_1}{2} \\
\vec{v_0} &= (p_0.y - p_1.y, p_1.x - p_0.x, 0) \\
center &= c_0 + \frac{\vec{v_0} * height}{2} \\
\alpha &= \vec{v_{side}} \cdot \vec{v_{(1,0,0)}} \\
scale &= (width, height, 1)
\end{aligned}
$$

The final shape is built in world space with the scale, orientation and position defined by the formula above. In the local space we preserve a square geometry of unitary dimensions to help the artist to adjust the real size of the geometry by directly setting its scale.

4) Polygon: Any other simplex can be traced to outline the floor boundary or the projection of other scene elements on the floor. The points selected in the panorama image are projected to the 3d world using the camera height the same way we do for the other geometry shapes.

B. Background mapping

To rebuild elements that are not contained on the ground we did a way to edit the meshes while looking at their projection in the panorama image. Traditionally, this is done using individual pictures taken of fractions of the set and used as background plates [1]. The same mapping used during render for the background plate needs to be replicated in the 3d viewport of the modeling software. In the end the environment image is mapped spherically as background element, allowing it to be explored with a virtual camera with regular frustum (fov $< 180°$) common in any 3d software.

The implementation prioritized a non-intrusive approach in the software to ensure it can be replicated regardless of the suite chosen by the studio/artist. After every 3dview rendering loop we capture the color and the depth buffer and run a GLSL screen shader with the inverse of the projection modelview matrix, the color and depth buffers and the panorama image as uniforms. There are two reasons to pass the matrix as uniform: (a) we used the classic GLSL screen shader implementation [11] which re-set the projection and modelview matrixes in order to draw a rectangle in the whole canvas so the shader program can run as a fragment shader on it. The matrixes are then rescued before the view is setup, and the inverse matrix is calculated in CPU; (b) We need to account for the orientation of the world calculated when the panorama is calibrated (see V-A).

$$M_{MVP}^{-1} = ((M_{ModelView} \cdot M_{Environment}) \cdot M_{Projection})^{-1}$$

The shader performs a transformation from the canvas space to the panorama image space and uses the depth buffer to determine where to draw the panorama. If the alpha channel is present in the viewport color buffer, the depth buffer is not needed. The background texture coordinate is calculated with a routine $equirectangular(normalize(world))$ where *world* is obtained with a GLSL implementation of **glUnproject** using the *Model View Projection* matrix uniform to convert the view space coordinates into the world space.

```
#version 120
uniform sampler2D color_buffer;
uniform sampler2D depth_buffer;
uniform sampler2D texture_buffer;

uniform mat4 projectionmodelviewinverse;

#define PI  3.14159265

vec3 glUnprojectGL(vec2 coords)
{
  float u = coords.s * 2.0 - 1.0;
  float v = coords.t * 2.0 - 1.0;

  vec4 view = vec4(u, v, 1.0, 1.0);
  vec4 world = projectionmodelviewinverse * vec4(view.x, ↩
       view.y, -view.z, 1.0);

  return vec3(world[0] * world[3], world[1] *  world[3], ↩
       world[2] * world[3]);
}

vec2 equirectangular(vec3 vert)
{
  float theta = asin(vert.z);
  float phi = atan(vert.x, vert.y);

  float u = 0.5 * (phi / PI) + 0.25;
  float v = 0.5 + theta/PI;

  return vec2(u,v);
}
```

```
void main(void)
{
  vec2 coords = gl_TexCoord[0].st;
  vec4 foreground = texture2D(color_buffer, coords);
  vec3 world = glUnprojectGL(coords);
  vec4 background = texture2D(texture_buffer, ↩
       equirectangular(normalize(world)));

  float depth = texture2D(depth_buffer, coords).s;
  if (depth > 0.99995) foreground = background;

  gl_FragColor = foreground;
}
```

The background is consistent even for different frustum lens and camera orientations. This technique frees the artist to create entirely in the 3d space without the troubles of the image/panorama space. For a perfect mapping it's important to use an image with no mipmaps or to use the GLSL routine to specify which mipmap level to access ($textureLod$).

C. Removal of environment elements

Modeling can be quite time consuming and a very daunting task. Some elements present in the environment may be undesired in the final composition. One example is the table in the middle of the scene showed along this paper. We inserted a synthetic carpet in the scene in a way that it completely overlays the table in the panorama image space.

In the figures 8, 9 and 10 you can see the carpet rendered in details. Also notice that we do not need the table to be modelled for the depth map (see figure 5. In the end if the object doesn't exist in the depth or the light parts of the map, is as if it was never there.

The same technique can be used to handle missing nadir capture (see IV-B).

D. Environment coordinate projection

We are using a method to transform the environment by deforming the support mesh created using the image as reference. Once the artist is satisfied with the accuracy of the mesh of the object to be transformed, she can store the panorama image space coordinates (UV) of each vertex in the mesh itself. From that point on, any changes in the vertices position can be performed as if affecting the original environment elements. For example, we can simulate a heavy ball (synthetic element) kicking on a couch (environment element) and animate the deformation of the couch pillows to accommodate the weight of the ball. In the figure 4 you can see the couch modeled from the background image and the mesh deformation over time. The renderer is capable of using this information to always use the light information from the stored coordinates instead of the actual position of the vertices (see VII-G). This works similar to traditional UV unwrapping and texture mapping. In fact this can be used for simple camera mapping (using this as UV and passing a LDR version of the panorama as texture) in cases where the renderer can't be ported to support the augmented reality features implemented in the *ENVPath* integrator. This can also be used to duplicate the scene elements in new places. A painting can move from one wall to another, the floor tiling can be used to hide out a carpet present in the environment, and so on.

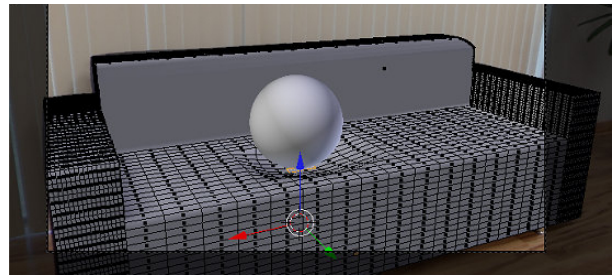


Fig. 4. Deformations of real objects using the environment coordinate texture. The deformed geometry is texturized using the texture of the original environment mesh before the deformation.

For the *ENVPath* integrator the UV alone is not enough. The vertices in the image space are represented by a direction, but we need to be able to store the original depth of the point as well. Therefore we store in a custom data layer not the UV/direction, but a 3-float vector with the original position of each vertex. We take advantage that both Blender internal file format and the renderer native mesh format can handle custom data. The renderer supports ply as mesh format, so we extended it with *property float wx*, *property float wy* and *property float wz*. The data needs to be stored in the world space (in oppose to the local space) to allow for the mesh to suffer transformations at the object level, not only at the mesh.

E. Modeling more than meets the eye

Even with a static camera we may need to know more information from the scene than what was captured originally in the panorama. For example, if a reflexive sphere is placed inside an open box you expect to see the reflection of the interior of the box in the sphere even if it was not visible from the camera point of view (and consequently is not visible in the pictures taken). Another case is to perform camera travelling in the 3d world. A new camera position can potentially reveal surfaces that were occluded before, and for them there is no information present in the panorama.

There are three solutions we considered for this problem. If the occluded element is not relevant for the narrative it can simply be ignored completely as if it was never presented in the real world (for example, an object lost underneath the sofa, invisible from the camera position); (2) In other cases the artist need to map a texture to the modeled element and create a material as she would in a normal rendering pipeline. (3) A mesh with the environment coordinate stored can be used to fill some gaps (see VI-D).

VII. Rendering process

The main problem in the traditional method for rendering 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 position where the

synthetic objects are to be inserted into the real scene, as did Devebec in [6]. If we need to introduce several objects in different positions of the scene, we will have problems with the positions of the shadows and reflections from objects in the final render (see figure 9). In addition to the above problems, if the map resolution is poor, we also have to get the background of the scene separately through photographs or video, and this makes the panoramic rendering more difficult.

The proposal framework allows us to model and synthesize a full panoramic 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 panoramic scene to be rendered.

The *ENVPath* integrator used in this production framework is a modified path tracing algorithm, similar to that described in [1], but with changes in implementation due to new features explored here. We will introduce now some key ideas of the rendering process because they will help the reader to understand the decisions taken in the production process.

A. Light-depth environment maps

In this paper we introduce a new type of space representation, 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 a 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 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 5. The channel depth map for a depth-of illumination light can be obtained by rendering from the shaping or also by scanning of the environment or other techniques.

For the following mathematic notations, a pixel sample from the light-depth environment map is denoted by $Map(\omega_i, z_i)$, where ω_i is the direction of the sample light in the map and the scalar value z_i denotes the distance from the light sample to the light space origin. The position for the light sample in the light space is given by the point $z_i\omega_i$.

B. Primitives: synthetic, support and environment surfaces

The rendering integrator used for this work is based in a unique classification of the different scene primitive types. Each primitive category defines different light scattering contributions.

Fig. 5. The upper picture shows the radiance channel of the environment and the lower one the depth channel used for reconstruct the light positions.

- *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 path tracer 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 take into account for the reflection and shadow computation for the other primitive types. They don't require any light scattering calculation, because their color is calculated directly from the light-depth environment map.

The level of detail of the environment reconstruction (see VI) 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. Since our primary goal is to render a full panorama image, 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.

C. Support surface BSDFs

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

$$L_o(p, \omega_o) = Map(\omega_p/|\omega_p|, |\omega_p|), \quad \omega_p = WTL(p),^1 \quad (2)$$

where the $L_o(p, \omega_o)$ term comes from the light scattering equation

$$L_o(p, \omega_o) = \int_{\mathbb{S}^2} f(p, \omega_o, \omega_i) L_d(p, \omega_i) |\cos\theta_i| \mathrm{d}\omega_i. \quad (3)$$

In order to get the equality from equation (2) is used a pre-computed scale factor

$$ES(p, n_p) = \frac{\int_{S^2} Lum(Map(\omega_s, z_s))\mathrm{d}\omega_s}{\int_{S^2} Lum(Map(\omega_j, z_j)) \left\langle \frac{\bar{\omega}_j}{|\bar{\omega}_j|}, n_p \right\rangle \mathrm{d}\omega_j}, \quad (4)$$

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

The $ES(p, n_p)$ scale factor represent the percent of light contribution coming from the map to the point p and is used in the Monte Carlo estimator of equation (3). Thus we have

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

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

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

D. Computing BSDFs scale factor for support surfaces

As our environment map has a depth channel, we can compute the integral from equation (4) considering geometric positions of the light samples instead of only use his directions as well as was made in [1].

To compute the integral (4) the map is discretized in a limited set of point lights whose radiance is equivalent to the total radiance of the whole map. Instead of summing over all pixels of the map, we only sum on the small group of point lights to get a quick estimate of the ES scale factor. The discretization of the map is made using the median cut method described by Debevec in [12]. Equation (4) calculated on the discretized representation of the light map by the set $L_d(i)$ of K point lights is given by

$$ES(p, n) = \frac{\sum_{i=1}^{K} L_d(i)}{\sum_{i=1}^{K} L_d(i) \langle \omega_i, n \rangle}, \quad with \quad \langle \omega_i, n \rangle > 0, \quad (5)$$

where

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

E. The visibility test for direct lighting

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 environment map to compute the direct light copntribution for point p. So, the render need to make 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 $LTW(\omega_i)$ of 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 for all object in the scene the shadows direction becomes in the same orientation because only are used directions for the light contributions accounts.

The *ENVPath* integrator works a little differently. Exploiting the light-depth environment map properties the visibility test uses the light sample position in real world and not only his direction.

The *ENVPath* 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 the world space to obtain $l_i = LTW(z_i \cdot \omega_i)$. Thus the calculation of visibility is made for the ray$r(p, l_i - p)$, whit origin p and direction $l_i - p$ as in figure 6.

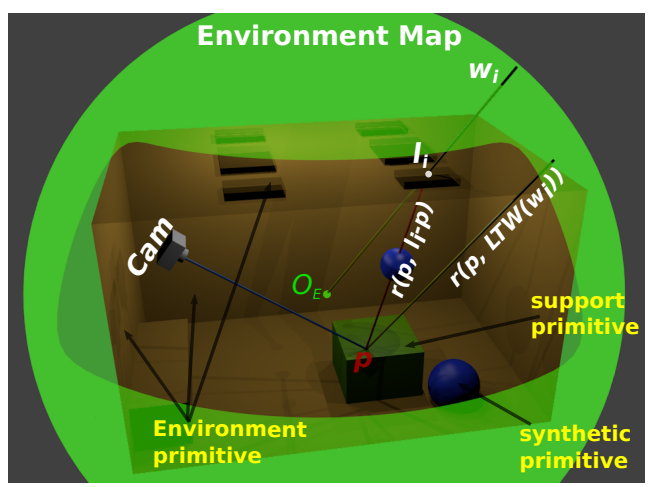


Fig. 6. The visibility test used by the *ENVPath* integrator. The ray $r(p, LTW(\omega_i))$ used by the directional approach is unoccluded and thus light l_i contributes for the scattering account of point p. The ray $r(p, l_i - p)$ used for by our light-positional approach is occluded by a synthetic object and so gets the correct world-based situation.

[1] $WTL(p)$ transform (p) from World space to Light space.
[2] $LTWTL(z_j\omega_j)$ transform $(z_j\omega_j)$ from Light to space World space.

F. Environment reflections over synthetic surfaces

Given a point p on a surface, the integrator takes a direction ω_i by sampling the surface *BRDF* to add reflection contributions to point p. To do this, is computed the intersection with the scene 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 over a synthetic or support surface his contribution must to be added to reflection account. Otherwise, if the intersection was with an environment mesh, we transform q from world space to light space by computing $q_L = WTL(q)$. position q for their position on the map lighting by computing the transformation $q_L = WTL(q)$. Finally the contribution given by the direction q_L in the environment map is added to reflection account. The figure 7 shows this proposal.

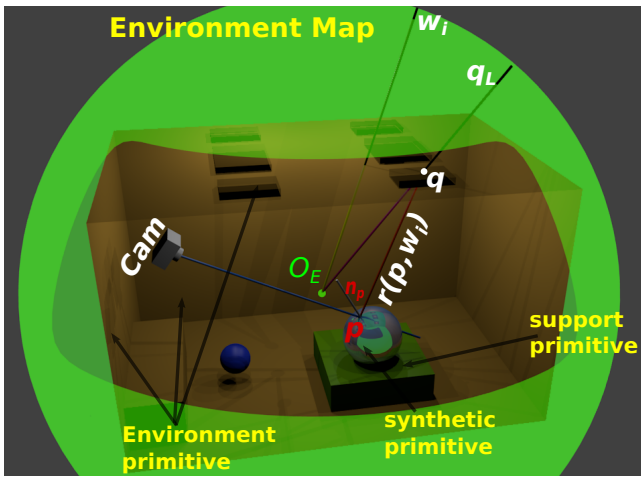


Fig. 7. Reflection ray path account used by the *ENVPath* integrator. As shown, the ray $r(p, \omega_i)$ intersects the scene at point q, over the environment mesh. The radiance of q is stored at q_L direction in the environment map. Our aproach computes the correct reflection q_L, instead compute the directional ray ω_i that gives a wrong reflection mapping.

G. Environment texture coordinates

Environment texture coordinates are used to get some rendering effect such as apply deformation on support objects when synthetic objects interact with them. Other application its copy texture from one environment region to other. This can be a powerfull tool for texturing support meshes on regions where do you don't know the color directly by the environment because they is occluded by other support object.

VIII. Results

The rendering time of this method is equivalent to the rendering of a normal scene with the same mesh complexity using a physically based light rendering algorithm. There was no visible lost 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. And the rendering solution converges to the final results without 'adjustments' loops been required like the original Debevec technique [6].

As explained during the paper, the core aspect of this process is to develop the rendering of full panorama images. More specifically the integration of a captured environment and synthetic objects. In the figure 8 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 mirror ball reflection calculation can be seen in more details in the figure 9. 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).

The proper calibration of the scene and the correct shadows helps the natural feeling of belonging for the synthetic elements in the scene. In the figure 10 you can see the spheres and the carpet inserted in the scene. The details are shown in a non panoramic frustum to showcase the correct perspective when seen in a conventional display.

Finally, we explored camera travelling for a few of our shots. In the figure 11 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 modelled. For slight camera shifts this is not even a problem.

Fig. 8. The top image is a full panoramic render of the augmented reality scene. The bottom images show the same scene rendered with fisheye lens.

IX. Conclusion

We have presented a general framework for adding new objects to full panoramic scenes based on illumination captured from real world and reconstructed depth. To show

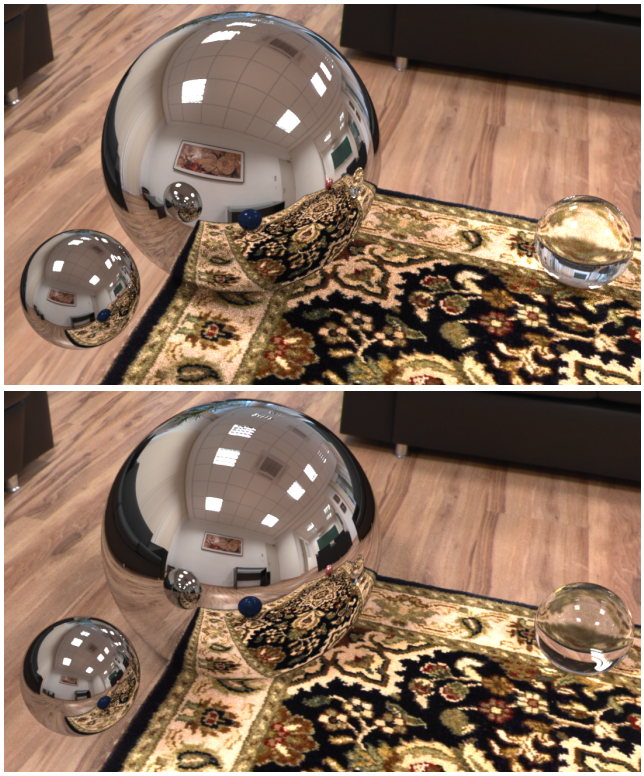


Fig. 9. Comparison between reflections given by directional (upper) and light-depth (lower) maps approaches.

Fig. 10. Shadows using light-depth maps. The red ball has different shadows directions than the blue ball.

the feasibility of such a production workflow is essential to produce realistic immersive scenes for panoramic displays.

A key point of our method, until now unexplored for this means, is the use of light position in the environment map instead 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.

Another thing to explore is the use more than one single environment capture to allow more complex camera traveling

(a) Camera positioned at position where the environment was captured.

(b) The camera position is displaced from the environment origin.

Fig. 11. Detail of the camera travelling effect. A door partially visible in (a) is occluded by the wall in (b).

in the scene. The use of multiple environment captures helps too to have minor occluded regions of the real environment. This is important to avoid abuse on using environment texture coordinates. They are better fit for applying soft deformations on the environment instead of texture copy to complete unknown environment elements.

Finally, we were quite please with the results of this framework for producing content for conventional displays. Camera panning and travelling works regardless of the camera frustum and field of view. For non-panoramic frustums camera zooming can also be used to deliver the essential camera toolset for traditional filmmaking.

References

[1] A. R. Zang and L. Velho, *Um framework para renderizações foto-realistas de cenas com realidade aumentada*. XXXVII Conferencia Latinoamericana de Informática (CLEI), 2011, Quito.

[2] A. R. Zang and L. Velho, *ARLuxrender project*, Visgraf, 2011, http://www.impa.br/∼zang/arlux .

[3] G. S. Miller and C. R. Hoffmanm, *Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments*, Curse Notes for Advanced Computer Graphics Animation, SIGGRAPH, 1984.

[4] L. Williams, *Pyramidal parametrics*, SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques, ACM, 1983, vol 17(3), pp. 1-11, ISBN:0-89791-109-1.

[5] P. E. Debevec and J. Malik, *Recovering high dynamic range radiance maps from photographs*, SIGGRAPH, 1997, pp. 369-378.

[6] P. E. Debevec, *Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-based Graphics with Global Illumination and High Dynamic Range Photography*, SIGGRAPH, 1998, pp. 189-198.

[7] K. Karsch, V. Hedau, D. Forsyth and D. Hoiem, *Rendering synthetic objects into legacy photographs*, Proceedings of the 2011 SIGGRAPH Asia Conference. ACM, New York, NY, USA, 2011, pp. 157:1–157:12

[8] P. D. Bourke and D. Q. Felinto, *Blender and Immersive Gaming in a Hemispherical Dome*, GSTF International Journal on Computing (JoC). ISSN: 2010-2283, Vol. 1, No. 1, 2010.

[9] D. Q. Felinto, *Domos Imersivos em Arquitetura*, Bachelor thesis at the Escola de Arquitetura e Urbanismo, Universidade Federal Fluminense, Niterói, 2010.

[10] P. Kuliyev, *High Dynamic Range Imaging for Computer Imagery applications - a comparison of acquisition techniques*, Bachelor thesis at the Department of Imaging Sciences and Media Technology, University of Applied Sciences, Cologne, 2009.

[11] R. J. Rost and B. Licea-Kane, *OpenGL Shading Language*, The Khronos Group, 3rd ed., 2010.

[12] P. E. Debevec, *A median cut algorithm for light probe sampling*, SIGGRAPH Poster, 2005.