Universiteit Utrecht

Realtime Environment Lighting Simulation for Augmented Reality Applications

Marco Antonio Salcedo Sanson

Utrecht University, Faculty of Science, Game and Media Technology Master's Program

Supervised by Amir Vaxman

Department of Information and Computing Sciences, Utrecht University

Dated: October 20, 2017

Contents

Chapter 1

Introduction

Augmented Reality is a technology with a lot of potential, but in its current state it's not very attractive to end-consumers. Having to point a device's camera for extended periods of time is uncomfortable and makes it hard to engage in using AR applications, but the way the virtual objects are rendered is completely unrealistic, which makes many AR applications unattractive to end users as well. The rendering of virtual objects on current AR applications has a number of shortcomings in the state of the art: marker jiggle, inaccurate depth cues, lack of occlusion from real life objects and incompatible lighting.
The method that we propose addresses the latter problem. The method is based on the idea that a 360° photograph of the environment can give us a lot of information about the lighting conditions, and that such 360° panoramic photographs are now simple to create locally on an average mobile device.
We present a method that automatically infers the lighting from a scene, and shades virtual objects accordingly. The environment-aware lighting is applied at runtime, and so it produces more realistic imagery regardless of the AR tracking technology used.
Unlike previous works, we focus on mobile devices, utilizing the ease of creating 360° imagery in practice. This decision comes by noting that AR is better suited for mobile devices than computers given their ease of mobility and the ready available sensors they have at the developer's disposal.
The major research question we investigate is: "Can the accuracy of lighting in AR applications be improved by using a mobile device to its full potential?". What this means is that thanks to the many ways a smartphone or tablet can interact with the user and the environment, we can know a lot about the surroundings. Such information is useful to save computation time.
The main contributions of our method are:

- An image-based method with which the lighting conditions of the entire environment are approximated.
- The seamless integration of virtual objects by the emulation of shading and shadows from acquired light sources.

Chapter 2

Related Work

2.1 Light detection methods

[Laskowski, 2007] came up with a method to infer which pixels in an image is a direct light source. It is a robust method that supports cases where there are no visible light sources and HDR images. Their method has a number of shortcomings, the response to outdoors photographs where there is only ambient light is not very accurate. Another limitation is that their algorithm has a trade-off with accuracy and converge time, as they explain that there is an optional step involving a Bresenham method which makes results more accurate but also presents a significant slow down. Our method not only calculates the position of light sources within the image, but also approximates their properties in 3D space.

2.2 Offline synthetic image composition

There have been several methods to insert virtual objects into static photographs automatically with highly realistic results. One such example is [Karsch et al., 2014]. These methods are not suitable for realtime applications, and they use a single view image to infer the lighting conditions. Our method is more robust in the sense that it uses a 360° view of the environment and thus uses information of the full environment and not just a section.

2.3 Realtime synthetic image composition

[Agusanto et al., 2003] proposed a method related to our methodology. Their method calculates the scene lighting using lightmap images. In order to generate such lightmaps they need a DSLR camera, software to generate HDR images

from multiple exposures, and a light probe in the form of a reflective sphere. Our method proposes using the 360° photograph on the device itself, which is accessible to an average consumer.

[Pessoa et al., 2011] created the Real-Time Photorealistic Rendering of Synthetic Objects into Real Scenes (RPR-SORS) toolkit as an extension of the ARToolkit Augmented Reality SDK. The method behind their toolkit is an updated version of the one in Agusanto et al. [2003] that uses cubemaps instead of HDR imagery to improve the rendering. This approach also has the drawback of requiring hand-crafted image maps that involve technical knowledge and dedicated software and hardware to generate.

Our method is also based on the work by [Kanbara and Yokoya, 2004]. Their method calculates direct lighting on an AR object using a 2D/3D marker. One part is a regular fiducial marker and the other is a small chrome sphere. The light reflections on the sphere are detected and the light direction is approximated using this information. The advantage of this method is that it doesn't require pre-generated image maps, but the need for a physical chrome sphere is still a shortcoming. Our method builds on this idea but uses a virtual analogy of the chrome sphere consisting of a 360° panoramic image mapped on a sphere; this has all the advantages of Kanbara's method without the disadvantage of needing an actual reflective sphere.

Chapter 3

Theoretical Framework

In this chapter we will introduce a few technical concepts in order to make our proposed method and what we aim to accomplish more clear.

3.1 Augmented Reality

Augmented Reality is defined as a software system that features a blend of real and virtual objects; the user can interact with it in real time and displays a 2D representation of a 3D world for both the real and virtual sides. This is the most accepted definition originally presented by [Azuma, 1997].
In order to combine real and virtual objects both worlds must be aligned, in the sense that they must share a common origin for both position and rotation, so that they can coexist in a realistic way within the same image. If the virtual objects present in the real scene are not aligned, the consistency of the whole composition is compromised.
The tracking of real world features as a guideline for alignment in the virtual world is one of the challenges in the field of AR. [Zhou et al., 2008] extensibly researched the state-of-the-art technologies used for tracking. Many techniques have been used, such as sensor-based tracking, that uses dedicated hardware to feed the software application the position and rotation information it needs. Our method uses vision-based tracking, in which the position and orientation origin of the virtual world are determined by images recognized using a computer vision algorithm. These images are named *fiducial markers* and there are different kinds of them:

- **Binary markers**: Artificial black and white images that form patterns. Any combination of at least 8×8 black and white pixels that don't form a repeating pattern can work as a binary marker.
- **Natural image markers**: The vision algorithm extracts features as points, lines, shapes and/or textures in any image that has identifiable

elements. This means that the image must have contrasting details spread across the whole image and not form repeating patterns.

- **3D markers**:A small object is analyzed and a point cloud is extracted. During the tracking phase, point clouds are generated from the camera input and matched to the reference cloud. The main advantage is that the viewing angle becomes irrelevant to recognize the marker.
- **Mixed**:Other approaches involving both sensors and vision algorithms have also been used.

While our method does not aim to alleviate the technical challenge of tracking other than existing means, it is necessary to explain the concept for clarity.

3.2 Sensors

State-of-the-art smartphones and tablets in the market have built-in sensors to perform different kinds of measurements that are helpful for application developers, such as motion, orientation and other environmental conditions. The ones that are relevant for this method are the following:
Accelerometer: Measures proper acceleration relative to gravity. Its application in mobile development is to measure motion changes and to the device orientation relative to the surface of the Earth. It usually consists of 3 orthogonal axes, and therefore can measure acceleration on one, two or three axes. The output is given in m/s^2.
Gyroscope: A sensor that is capable of measuring the rate of rotation around a particular axis. It serves the same purpose as the accelerometer, but mobile devices usually have both for more robust measurements. The output is given in rad/s. In our method these will be used to acquire the 360° panoramic photograph, making sure that the device is within the same pitch while capturing all the images that will be stitched together to make the panorama.
Magnetometer: It serves the purpose of a compass by measuring the orientation of the device with respect to the Earth magnetic poles. For application development, it is useful to get the heading of the device with respect to the geographic north in a clockwise direction. This heading is an angle expressed in degrees.
GPS: Measures the raw position of the device in three-dimensional Cartesian coordinates where the origin is the center of the Earth. It is used to determined where on Earth the device is located (Country, city, neighborhood, etc.). It needs to have an uninterrupted line of sight, with no electromagnetic interference, to at least 4 out of the 24 satellites in orbit that are used for GPS.

3.3 Standard Illuminant

In order to describe the color of a light source, it is important to have detailed knowledge of the type of illuminant used. The [on Illumination, 1998] (CIE)

defined a number of spectral power distributions, referred to as CIE standard illuminants, to provide reference spectra for colorimetric issues. The illuminants are denoted by a letter or a letter-number combination. Their spectral power distributions (SPD) are normalized to a value of 100 at a wavelength of 560 nm. Illuminants series A through D exist, all of them specializing on a different kind of light source. For our method, the relevant one is series D, it describes a type of average light source, typically associated to daylight global illumination.
D65 corresponds roughly to the average midday light in western and norther Europe, comprising both direct sunlight and the light diffused by a clear sky in indoors situations. Because of that, it is also called a *daylight illuminant*. It has a correlated color temperature of approximately $6500K$. As any standard illuminant is represented as a table of averaged spectrophotometric data, any light source which statistically has the same relative spectral power distribution (SPD) can be considered a D65 light source. It is important to note that D65 is purely theoretical, there are no actual artificial D65 light sources. The D65 is the white point that we used to convert chromatic information into luminance for the light source detection, as we will detail further in the methodology section.

3.4 Panoramic photography

Panoramic photography is a technique for capturing images with horizontally, and sometimes also vertically, elongated fields of view. While there is no formal definition of the minimum field of view required for a photograph to be considered panoramic, in this work we focus on complete horizontal 360° captures.
There are different types of panoramic images. The one called wide-format photograph consists of a series of photos taken from the same height, that are then stitched together to appear as a single wide image. The image can subsequently be projected on to a cylinder. This kind of panoramic photograph captures 360° of the horizontal field of view, while the vertical field of view is dependant entirely on the lens used to capture the individual photographs. Figure 3.1 shows an example of a cylindrical panoramic image.

Figure 3.1: A cylindrical panorama of Trafalgar Square.

The type of panoramic image used in our method is called equirectangular panoramic image, or spherical panoramic image, due to the fact that the resulting image can be projected on to a sphere. The projection maps meridians to vertical straight lines of constant spacing, and circles of latitude to horizontal straight lines of constant spacing. This kind of projection introduces the types of distortion often seen in spherical projections, such as Mercator, where the poles of the sphere appear bigger than they really are. The process used by Google to create equirectangular panoramic images on mobile devices involves taking several pictures of the environment and, using the device's sensors, project them onto a sphere in their position relative to the geographic north.
Image features are identified for each individual image, and then those features are identified on neighboring images and aligned so that they overlap. The area of overlap is then blended, resulting in a seamless, yet distorted representation of all the images as one. The projection is then applied as follows:

$$\theta = \frac{x}{cos(\phi_l)} + \theta_0 \tag{3.1}$$

$$\phi = y + \phi_l \tag{3.2}$$

Where:
θ and ϕ are the longitude and latitude of the location to project, respectively.
ϕ_l are the standard parallels (tropics) where image distortion is zero.
θ_0 is the central meridian of the map.
x and y are the horizontal and vertical coordinates of the projected location on the map, respectively.

The kind of expected result can be seen on figure 3.2

Figure 3.2: An example of an indoors equirectangular panorama.

3.5 Environment mapping

Environment mapping is an image-based technique to approximate the appearance of the overall light conditions of an environment. This is accomplished by means of a precomputed texture image mapped on a geometric surface as a far-away environment surrounding. This technique was originally proposed by [Blinn and Newell, 1976] using spheres. Nowadays there are other alternatives, such as cube, paraboloid, pyramid or cylinder maps. The principle for each surface is the same, but the way to map a planar image onto the surface varies per surface.

In order to generate an environment map it is necessary that the panorama is made into a High Dynamic Range image. This is because a Low Dynamic Range image fails to capture the information necessary to simulate correct color balance, shadows, and highlights of the lighting environment; ultimately producing both inaccurate and less visually pleasing results. This has been illustrated by [Debevec, 2008].

A relatively easy and effective way to make an image into an HDR version is a technique called Tone Mapping, in which versions with different exposure values of the same image are blended together to include the full range of highlights and shadows of the overexposed and underexposed versions in a single image. It's important to disclaim that the Tone Mapping process does not yield an actual HDR image, but a standard 24 bit image in the 0...255 range, with highlight and shadow valued clipped. Nevertheless, the advantage of this process is that the environment will be described in a richer way, capturing the bright areas and the shadows better than the standard exposure image.

Chapter 4

Methodology

The input to our problem is a fiducial marker F that provides the object position in the real world and a 3D mesh M that will be rendered in Augmented Reality. The method will calculate the properties of a set of light sources L_i, specifically position, rotation, intensity and color. In order to analyze the luminance ($L()$), a 360° panoramic image is required, generated as a pre-processing step as explained in chapter 3. The complete process is depicted in the flowchart in Figure 4.1, and each step is consequently described in more detail according to the mentioned sections.

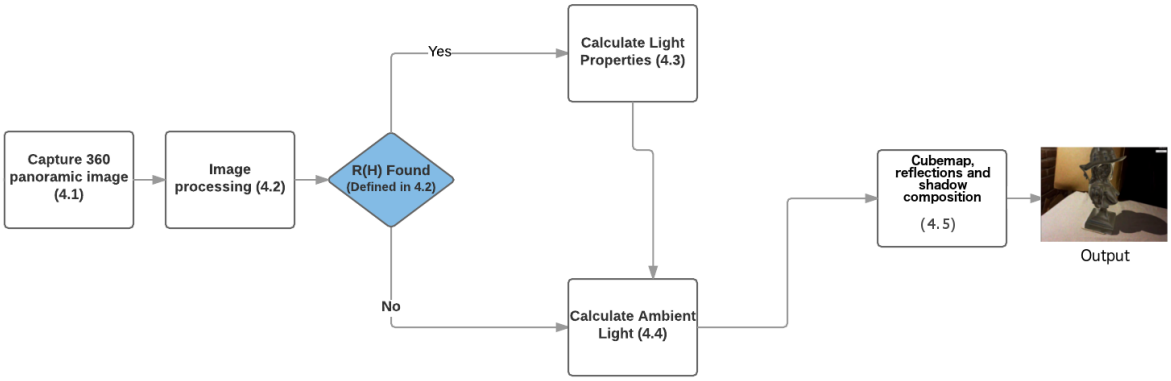

Figure 4.1: Our method in a nutshell.

4.1 Capture 360° panoramic image

We use panoramic images as a pre-process and therefore it's not within the scope of the method to define a new way of capturing 360° images. The application we

used to procure a spherical panoramic image within the device itself is Google Street View. This application guides the user through the process of generating the 360° panorama. In total 44 photos are necessary and the application shows an orange dot on the screen on the point where the user has to point the camera next. This functionality was originally conceived to capture outdoors scenes, but if the user stands roughly in the middle of a room it works to capture indoors scenes as well.
We ask the user to provide the origin of the virtual world by rotating the virtual reflective sphere so that the view of the room aligns with that of the section of the real room that the camera is facing. This also simplifies calculations of light orientations later on.

4.2 Light source filtering

We factor out the luminance from the RGB representation of the panorama. This is achieved by the following equation:

$$\forall\ P_{ij}; L(P_{ij}) = 0,2126 \cdot R_{ij} + 0,7152 \cdot G_{ij} + 0,0722 \cdot B_{ij}, \tag{4.1}$$

where L is the luminance obtained at D65 white point and P is the pixel in the i, j position of the image and R, G, B are the red, green and blue components of the pixel, respectively.
The contrast ratio has to be adjusted, so that the regions with high luminance are more clearly separated.

$$g(i, j) = \alpha \cdot L(i, j) + \beta \tag{4.2}$$

Where $g(i, j)$ is the adjusted image, $L(i, j)$ is the original grayscale image and α and β are the brightness and contrast constants respectively, determined by parameter tuning. We carried out the parameter tuning in a trial-and-error basis, by using initial extreme values and run the program, and varying the values in order to achieve the best result. The values that worked in the implementation were $\alpha = 220$ and $\beta = 255$. In order to prevent outlier pixels and noise from causing false positives, we normalize $g(i, j)$ as follows:

$$N(g_{ij}) = \frac{2 \cdot g_{ij}}{min(L) + max(L)}, \tag{4.3}$$

Where $min(L)$ and $max(L)$ are the overall minimum and maximum luminance values in the image.
The result of these steps is a black and white image with the rough shape of the light source, we call these shapes *regions of high luminance*. If there are no regions of high luminance in the image at all it means that no relevant light sources were found. A region of high luminance in the image is formally defined as follows:

$$R(H) = \{p_{00}, p_{01}, ..., p_{mn}\}, \tag{4.4}$$

where p_{ij} is the pixel in the i,j position of the image, so that

$$L(p_{ij}) \leq 0.9 \cdot max(L(p_{ij}))$$

The region must also be connected side-by-side, so the pixels must be adjacent. Regions of high luminance are for the most part characterized by much noise and artifacts, caused by clear objects, reflections of light sources on polished surfaces, and even light sources that are far away, but don't contribute an important amount of light to the area of interest. Therefore, it's necessary to also add further filtering to the pipeline.
The amount of light that a lighting source contributes to a given scene is directly proportional to the emission area of that source. When analyzing a photograph this area translates to the pixel size of the light source in the image. So we consider this to be a good measure for filtering. Figuring what constitutes an acceptable region size to determine if we have to filter a the light source is a challenging problem for which we couldn't find an existing solution. The approach we took is to define a percentage of the width and height of the overall image as a threshold in search for the best solution. The size filter is then:

$$width(R(H)) \cdot height(R(H)) \geq k \cdot W \cdot H \tag{4.5}$$

Where k is the threshold, in the implementation the value that yielded the best results was $k = 0.004$; W and H are the total image width and height.
Each identified region of high luminance that passes all the filers is a light source used at runtime. We capped the maximum amount of light to 8 in practice. We decided to use a maximum of 8 for a few different reasons. With more than 8 light sources it becomes challenging to keep the scene from being too lit, even normalizing the intensities. Since we're calculating shadows for each light source, we found that doing it for more than 8 light sources made this process a bottleneck that made the whole application performance drop.

4.3 Calculate light properties

We need to calculate position, orientation, intensity, and color for each light, remembering that we have the original full color image available.

1. **Orientation**: This process is carried out with an array P_i of points containing the pixel coordinates (x, y) of the regions of high luminance; the normalized luminance analysis image $N(g_{ij})$ mapped on a sphere S and a camera C facing the sphere from four different points of view (in order to cover a full revolution). The orientation of the pth light source $O(L_p)$ is calculated as follows:

Algorithm 1 Light source orientation calculation

for `i` $\in [1, 4]$ **do**
  for `Each pixel in camera view` **do**
    `Cast ray to` S
    if Color at hit point H is white AND $H = P_p$ **then**
      `Calculate` $O(L_p)$ `where` N_h `is the normal of` S `at` H, `and` C_r `is the camera ray direction.`

$$O(L_p) = -2 \cdot (N_h \cdot C_r)N_h + C_r, \tag{4.6}$$

      `Normalize` $O(L_p)$
    end if
  end for
  `Rotate camera 90° clockwise`
end for

This process is depicted in Figure 4.2 for further clarification. The figure shows the vectors N_h and C_r, the C orbiting S and the four distinct points of view with which C reconstructs the entire 360° environment.

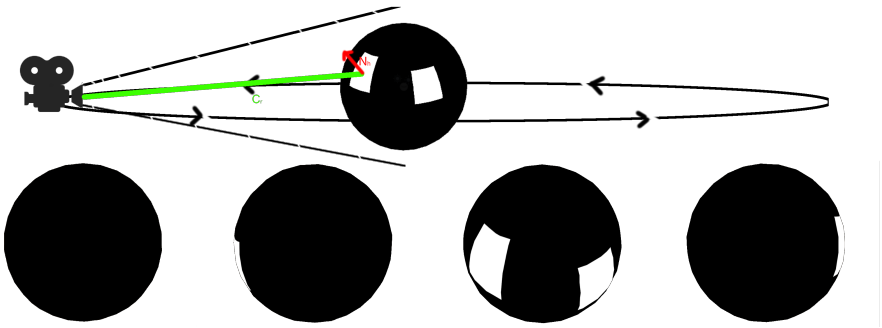

Figure 4.2: Illustration of how the light orientations are calculated based on the virtual reflective sphere.

2. **Position**: With the orientation calculated in the previous step we know the angles of the light source with respect to the marker F. What we don't know is how far away along that vector the light actually is. In order to approximate the distance from a camera to an object in a photograph we can use triangle similarity, the device's known camera parameters and a known object size. The ratio of the size of the object on the camera sensor and the size of the object in real life is the same as the ratio between the

focal length of the lens and distance to the object. We use the average of a light source's height to approximate the desired distance:

$$D = \frac{f \cdot h(O_r) \cdot h(L)}{h(O_p) \cdot h(s)}, \tag{4.7}$$

where f is the camera aperture size, $h(O_r)$ and $h(O_p)$ are the real object height in millimeters (the value we used is 200 mm) and the image object height in pixels, respectively; $h(L)$ is the height of the full image and $h(s)$ is the camera sensor height.

3. **Color**: Storing both versions of the panorama, one in full color and another one after processing, allows us to have both the color and the luminance information. Once a light source is detected, the equivalent area in the color image is averaged to determine the color of the light source. We obtain the linear average of each individual color channel and use the combined results.

4. **Intensity**: There are two factors that influence the intensity of a light as perceived by a camera, the light size and the color temperature. The light's color temperature in Kelvin is calculated using the approximation proposed by [McCamy, 1992], as follows:

$$T(C_p) = -949.86315 + 6253.80338\,\frac{-n}{0.92159} + 28.70599\,\frac{-n}{0.20039} + 0.00004\,\frac{-n}{0.07125} \tag{4.8}$$

$$n = \frac{0.23881 \cdot R + 0.25449 \cdot G - 0.58291 \cdot B}{0.11109 \cdot R - 0.85406 \cdot G + 0.52289 \cdot B} \tag{4.9}$$

Where R, G, B are the red, green and blue components of the light color. The size analogy is given by the integral of the region of high luminance with respect to the full image. The light intensity is finally expressed by:

$$I(L_p) = T(C_p) \cdot \int_R L(i)\, dx \tag{4.10}$$

Where $L(i)$ is the output image of the luminance analysis and R is the p^{th} region of high luminance. $I(L_p)$ is the scalar value that denotes the intensity of the light source at runtime.

4.4 Calculate ambient light

Since a panoramic image of the environment is already available, using it to implement environment mapping would be an adequate use of resources. However asking the user to capture the environment more than once in order to carry out the Tone Mapping would have a bad impact on user friendliness, and also it's highly unlikely that the produced image would have the exact same framing every time. Instead we produce the different exposure values for the Tone

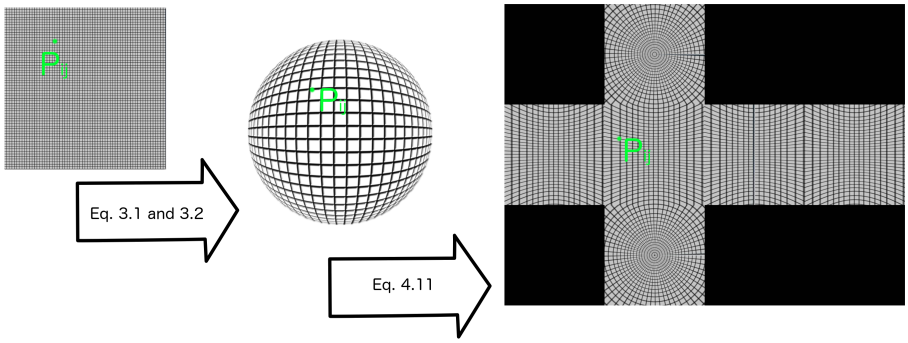

Figure 4.3: The cubemap creation process.

Mapping by altering the brightness and contrast values of the base image using equation 4.2 once again. After we produce the HDR image using the Debevec and Malik [1997] weighting algorithm.

4.5 Real-time Phase

We use the original spherical panoramic image to create a cubemap of the real environment for ambient lighting, to calculate the ambient contribution of the diffuse shaders and to use as reflections for the specular shaders.

1. **Cubemap:** We take the polar coordinates of a unit sphere $(1, \theta, \phi)$. The image coordinates are divided into four regions by latitude $-\pi/4 < \theta < \pi/4, \pi/4 < \theta < 3\pi/4, 3\pi/4 < \theta < 5\pi/4, 5\pi/4 < \theta < 7\pi/4$. These represent either one of the four faces of the cube, top or bottom. The projected coordinates are given by:

$$P_c = (1, tan(\phi), \frac{cot(\theta)}{cos(\phi)}), \tag{4.11}$$

   The process is further explained in Figure 4.3.

   The projected points for each face of the cube are composed into an image and each image stitched together to create a cross cube map.

2. Ambient contribution: The environment lighting contribution is done with a shader. The ambient contribution is reduced to a single value in the range of 0 and 1 and to the diffuse color component. The result is a dimmer color when the ambient light is low. The ambient contribution value A_c

for each region of high luminance in the luminance image $R(L(P_{ij}))$ is given by:

$$\forall R(L(P_{ij})); A_c = \frac{\sum_{i=1,j=i} R(L(P_{ij}))}{width(R(L(P_{ij}))) \cdot height(R(L(P_{ij})))} \tag{4.12}$$

3. **Reflections:** The cubemap from step 1 is also used to for reflections. A vector is cast from every vertex of the object along its normal and intersected with the cubemap generated from the panoramic image. The color is mixed with the albedo according to the specularity defined for the material to simulate reflection, this information is calculated in advance and used at runtime.

4. Shadows: During early experiments it became clear that one of the bigger differences between real and virtual object were a product of the shadows as well as the light. A problem when it comes to casting shadows for virtual objects in AR is the fact that there has to be an object underneath M to cast the shadow on, but this object has to be as unobtrusive as possible. We worked around this issue by using a transparent material on a plane that receives shadows. M is projected on the plane from each light source and rendered flat in black color.

Chapter 5

Implementation

In this chapter, implementation details of our method. We devised our method so that it could be used entirely on a mobile device. To that end, the implementation uses different technologies. We used the Google Street View app for panoramic image capture, it's available for both iOS and Android and it's a ready-made solution to use panoramic images as a tool for the experiments. The base of the implementation is done in Unity, but with a native code library written in C++. Our library uses OpenCV for the image processing tasks. OpenCV is also available as a library for Unity, but we decided to create our own custom library because we identified limitations in the OpenCV for Unity library. It is a paid library, it doesn't have all the functionality that its C++ counterpart does, and it is slower than a native code library. All of these technologies are platform independent, even as some adjustments do have to be made in order to target either iOS or Android. For instance, to be able to use the panoramic images created with Street View, we had to create a native code plugin to read images from the file system of the device. This plugin needs separate implementations for Android and iOS. The ARToolKit library is used for the Augmented Reality tasks of tracking markers and providing the virtual world's point of reference. The device used for this particular implementation is an iPad Air tablet running iOS 11. Having said that, adjusting the application to support Android devices as well would be a simple task.
The OpenCV API has all the necessary functions implemented to apply the mathematical functions proposed in the method section so this part of the implementation was a direct translation into code. We used the tools available in Unity for the sake of achieving the desired graphic quality. Reflections are implemented using Reflection Probes for example. Shadows are baked using Unity, but we created a custom shader for the "shadow catcher" plane. We also applied a full screen Film Grain effect to the composted scene. We dit it in order to make the clean virtual camera match the noisy device's camera and so homologate the look of both the real and virtual camera.

Chapter 6

Experiments

The aim of the method is to produce plausible lighting for Augmented Reality applications, and thus increase the sense of realism of the composed graphics. The experiments are designed to evaluate the similarity of a real object and a virtual representation of the same object in the same controlled lighting conditions.

6.1 Setup

We chose a particular object with a rich variety of materials, in this case it was an Xbox 360 controller with a custom paint job. We modified a 3D model of the same Xbox 360 controller to match the custom paint job and the materials were replicated as closely as posible to the real object using Unity's built-in shaders. In the end 4 main shaders were used, a highly reflective plastic for the borders and some of the buttons, a more matte plastic for the main body, a completely specular chrome for the Xbox button and a semi-transparent and glossy plastic for the colored buttons. The choice of this motif was based both on the ease to find a reliable 3D counterpart for a real object and on the already wide variety of materials present in the object.
In order to provide a ground truth for a reliable size by side comparison a screen capture of the application running while both the real and virtual object are in the frame, in similar positions and orientations and affected by controlled lighting that is also simulated using the method for the virtual counterpart. Another experiment is a direct substitution: place the marker on the table, then place the real controller on the marker in a similar position and orientation. This provides a clear comparison, everything in the scene remains the same, and both the real and virtual objects can be appreciated in the same setting.
As for benchmarking how this method stands in comparison to other similar ones, we replicate the conditions of the experiments presented in the results section for the methods by Kanbara and Yokoya [2004], Karsch et al. [2014] and Pessoa et al. [2011] in terms of similar setting and virtual objects used.

The resulting images are compared to the ones from their respective methods. In order to replicate these settings additional 3D models are needed, namely a teapot and the dragon and Buddha from the Stanford 3D Scanning Repository. The actual way in which the experiment is conducted is defined in the following scenario:

- **Setting:** A room with consistent and invariable lighting is used and some kind of flat, matte surface to lay the objects on. An Xbox 360 controller with the characteristics described previously and a marker to track the virtual object.
- **Requirements:** A mobile device running the developed demo application, a marker for virtual objects. A real object and its corresponding virtual counterpart, modelled as close as possible. Common 3D models from the Stanford 3D Scanning Repository.
- **Goals:** Obtaining a set of images that will enable readers and experimenters alike to make a fair comparison of the method application, side by side with a real object counterpart.
- **Actions:** Place the controller on the surface, and capture and image. Then remove the real controller and substitute it with the marker in a way that the virtual controller appears in the most similar position and orientation possible; capture an image in the end as well. Replicate the scenarios in Kanbara and Yokoya [2004], Karsch et al. [2014] and Pessoa et al. [2011] using teapots, Buddahs and dragons and capture images of each.
- **Benchmark:** The image result yielded by the method, as well as the time per frame will be captured and used for comparison's sake and benchmarking head to head with the results from similar methods.

It's important to also mention the need of replicating the previous work's settings and the reason why. In some cases the product is not a runtime application, and thus the only applicable comparison is image-based. The authors of other runtime methods were approached to evaluate the possibility of using their binaries to produce images and they either declined or gave no answer. And so the only possible comparison is image to image and taking their word for the performance indications in the form of frames per second.
All of the experiments were recorded on video by connecting the tablet device to a laptop, and are available to watch. The images shown here are select still images from said videos and the framerates discussed per experiment are averages from the on-screen counter seen on each video. It's also important to say that the recorded framerates are about 10% lower than when the application is not being recorded.
.

6.2 Experiment 1: Real object substitution

This experiment produced satisfactory results, the materials and the position and orientation on the virtual object are not exactly the same, and so the highlights shining off the surface are not in the same relative places. But the shadow shows that the light is placed in a very acceptable approximation of the real light. The application is running at 24 FPS on this screen capture and it's a good measure of the average of performance.

Figure 6.1: Real custom painted Xbox 360 controller

Figure 6.2: Virtual counterpart of the same object

6.3 Experiment 2: Karsch's scenario

This scenario was hard to replicate, due to the unusual framing and placing of the objects. The method we are comparing to is not a runtime method, but even so there are conclusions that can be drawn. In the image produced by the Karsch et al. [2014] application it's hard to say if the lighting in the real and virtual components of the image really corresponds. The light sources, except for one, are not present in the frame and there are no similar real objects to the virtual ones that would serve as ground truth. Our results show enough evidence that the lighting in both the real and virtual worlds is similar, with cues such as the presence of real objects and their shadow directions.
The framerate of our application for this experiment is 15 FPS on average. It is still acceptable considering that there are 6 virtual dense models.

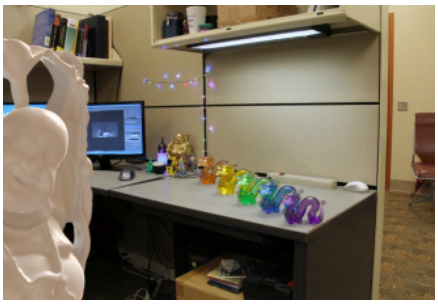

Figure 6.3: Karsch's method results

Figure 6.4: This methods's results

6.4 Experiment 3: Kanbara and Pessoa's scenarios

These two scenarios will be grouped together due to the fact that the steps to replicate the experiment are the same. They both presented ceramic material teapots with a lighting setting that they did not specify.

Figure 6.5: Pessoa's method results

Figure 6.6: Kanbara's method results

Both results look well blended into the environment, however in Pessoa's case it also must be said that the lighting setting is not disclosed and is not obvious from just looking at the results. For the aforementioned method there's also a considerable amount of pre-production required for the method to actually work, and the performance has a rather bad scaling, ranging from 180 FPS with no objects to 5 FPS with seven objects. The method in this work doesn't have those problems, the only pre-step required is to take a spherical panoramic image with the same device and no technical knowledge necessary; and performance scales better going from 24 FPS with a single object to 15 with six of them.
In the case of Kanbara's results they look really good, the downsides to their

method that the one here resolves are the need for a physical 3D marker to probe lighting and the performance, they report 20 FPS on a computer for the experiment they published. It's admittedly a more than 10 year old computer at the time of this writing, but our method is achieving slightly better framerates on a mobile device.

Figure 6.7: Pessoa's comparison results

Figure 6.8: Kanbara's comparison results

6.5 Gallery

Here are a few more screen captures from the application that show the potential of the method for the reader to draw their own conclusions.


FPS 23.2


FPS 23.0


FPS 24.1


FPS 20.5

Chapter 7

Discussion

In this chapter the results, discussed and a conclusion is formulated. The shortcomings of the method and the implementation are also mentioned and taken into consideration for future work.

7.1 Discussion

We can appreciate a very close resemblance among the virtual and real objects, however it's still not a perfect fit. Some of the factors cannot be worked around trivially, for example one of the differences is that there are small imperfections present in the real object that are not there in the virtual one, in order to account for them, a 3D scan of the specific real object would be necessary. 3D scans are often extremely dense meshes that would pose a problem to maintain interactive framerates.
Despite the film grain filter, it's still evident that the virtual camera has better resolution than the real one. The fact that in the virtual world everything looks "too perfect" also breaks the illusion of realism. However, in a real application of the method the user would ideally use AR to view something that is not there in the real world, instead of a side by side comparison to a real object. So by not having a clear ground truth of how such an object would look through the camera feed this problem is mitigated.
Our method is placing lights using a reasonably accurate orientation, but at an admittedly inaccurate distance. After seeing the results we can also conclude that the orientation is far more important than the position, we do see differences in the length of the projected shadow, which is a determined by the distance to the light source, nevertheless this discrepancy is less noticeable than when the angle of the shadow is different from the real reference.
When virtual shadows overlap a real object that is in front of it the result is not visually pleasing or realistic. In order to prevent this and make the shadow cast on non-planar neighboring objects, or at least have the objects occlude it, some knowledge of the environment in 3D would be necessary, and that would

require hardware that is not present on average consumer smartphones.
There are a few known cases where false positives are found by the light detection algorithm. When a light source is directed towards a glossy surface, such as a varnished desk for example, the incidence of the light on such a surface is detected as a light source as well. A similar thing happens sometimes for intense white objects that are not light sources. Although in the first scenario it can be argued that the light bouncing off a glossy surface could indeed be considered a light source.
The system is closed once the runtime phase starts, therefore it lacks adaptation to light changes, due to the nature of the method, having a pre-calculation phase and a runtime phase, it wouldn't be possible to change the light setup in the same way as it was originally captured.
The system limits the amount of lights it simulates to 8 for performance reasons, but even at 8 light sources the performance of the application is low. Between 15 and 24 FPS, it can still be considered interactive, but it's not ideal, specially when taking into account that the application is only doing rendering, a real application of the method, such as a game, would require other layers of complexity that would need processing time.

7.2 Conclusion

All in all, despite the shortcomings of the method the objective was achieved. The main goals of the method are to approximate the lighting conditions of the environment via a panoramic image and to use this knowledge of the environment to improve the realism of the rendering. Both objectives are met, in the understanding that improvement is not an absolute measure, meaning that as long as the method makes the composted scene more realistic the objective is fulfilled, regardless of being far from perfection still.
There's room for improvement, both on the technical side, on the user experience side and in the results. This will be further developed in the next section.

7.3 Future work

During the development of this method there were several changes to the Augmented Reality landscape through the introduction of new SDK's. ARKit [Apple, 2007] and Tango/ARCore [Google, 2007] are more robust solutions than the ARToolKit we used in this method. While both ARKit and ARCore have an ambient light estimation feature, and ARCore supposedly even has direct illumination estimation, the overlap with what was proposed here is not complete. In a future iteration of the system it would be interesting to see the method working with these new SDK's. Google Tango would be even more interesting, due to the fact that it has a depth sensing camera, so it would even be possible to roughly reconstruct the scene for accurate shadow casting and determine the

distance to light sources precisely.
Other than that, a vast improvement would be support for changing light conditions. Most real-world applications for such AR graphics would require interaction outdoors for more than a few minutes, so the lighting conditions are bound to change in the middle of a session.
Just for the sake of maintaining a more uniform user experience it would be a nice addition to implement something similar to the Google Street View spherical panorama capture module within the app. This is something completely off-topic for the method and something not trivial to implement, but if it was there it would be a nice addition nonetheless.

Bibliography

Agusanto, K., Li, L., Chuangui, Z., and Sing, N. W. (2003). Photorealistic rendering for augmented reality using environment illumination. *Proceedings of the Second IEEE and ACM International Symposium on Mixed and Augmented Reality*.

Apple (2007). Introducing arkit. https://developer.apple.com/arkit/.

Azuma, R. T. (1997). A survey of augmented reality. *Presence*, 6(4).

Blinn, J. F. and Newell, M. E. (1976). Texture and reflection in computer generated images. *Commun. ACM*, 19(10):542–547.

Chuang, A., Yang, T.-L., and Chee, Y.-W. (2010). 360 degree panorama using an android phone. http://www.cs.cornell.edu/courses/cs4670/2010fa/projects/final/results/group_of_acc269_ty244_yc563/cs4670_final.html.

Debevec, P. (2008). Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, pages 32:1–32:10, New York, NY, USA. ACM.

Debevec, P. E. and Malik, J. (1997). Recovering high dynamic range radiance maps from photographs. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 369–378, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Google (2007). Tango developer overview. https://developers.google.com/tango/developer-overview.

Kanbara, M. and Yokoya, N. (2004). Real-time estimation of light source environment for photorealistic augmented reality. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 2, pages 911–914 Vol.2.

Karsch, K., K., S., N., H. S. C., H., J., R., F., M., S., and D, F. (2014). Automatic scene inference for 3d object compositing. *ACM Transactions on Graphics*.

Laskowski, M. (2007). Detection of light sources in digital photographs. *In 11th Central European Seminar on Computer Graphics*.

McCamy, C. S. (1992). Correlated color temperature as an explicit function of chromaticity coordinates. *Color Research and Application*, 17(2):142–144.

on Illumination, I. C. (1998). Cie standard illuminants for colorimetry. http://www.cie.co.at/publ/abst/s005.html.

Pessoa, S. A., de S. Moura, G., do M. Lima, J. P. S., Teichrieb, V., and Kelner, J. (2011). Rpr-sors: Real-time photorealistic rendering of synthetic objects into real scenes. *Elsevier*.

Xing, G., Zhou, X., Peng, Q., Liu, Y., and Qin, X. (2013). Lighting simulation of augmented outdoor scene based on a legacy photograph. *Computer Graphics Forum*, 32(7).

Zhou, F., Been-Lirn, H., and Billinghurst, M. (2008). Trends in augmented reality tracking, interaction and display: A review of ten years of ismar. *IEEE International Symposium on Mixed and Augmented Reality*, 15(18).