Generating Light Estimation for Mixed-reality Devices

through

Collaborative Visual Sensing

by

Siddhant Prakash

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved October 2018 by the
Graduate Supervisory Committee:

Robert LiKamWa, Co-Chair
Yezhou Yang, Co-Chair
Dianne Hansford

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

Mixed reality mobile platforms co-locate virtual objects with physical spaces, creating immersive user experiences. To create visual harmony between virtual and physical spaces, the virtual scene must be accurately illuminated with realistic physical lighting. To this end, we design *GLEAM*, a system that Generates Light Estimation Across Mixed-reality devices to continually sense realistic lighting of a physical scene in all directions. We designed GLEAM to optionally operate across multiple mobile mixed-reality devices to leverage collaborative multi-viewpoint sensing for improved estimation. We also studied policies that prioritize resolution, coverage, or update interval of the illumination estimation depending on the situational needs of the virtual scene and physical environment.

To evaluate the runtime performance and perceptual efficacy of our system, we implemented GLEAM on the Unity 3D Game Engine. We deployed our implementation on Android and iOS devices. On these implementations, GLEAM can prioritize dynamic estimation with update intervals as low as 15 ms or prioritize high spatial quality with update intervals of 200 ms. Our user studies across 99 participants and 26 scene comparisons reported a preference towards GLEAM over other lighting techniques in 66.67% of the presented augmented scenes and indifference in 12.57% of the scenes. Our controlled lighting user study on 18 participants revealed a general preference for policies that strike a balance between resolution and update rate.

To my parents and sister

ACKNOWLEDGMENTS

I would first like to thank my thesis advisor Dr. Robert LiKamWa of the School of Electrical, Computer and Energy Engineering and the School of Arts, Media and Engineering at Arizona State University. Prof. LiKamWa's constant support and presence during these last two years helped me get past every hurdle I faced or had a question about my research, writing, or presentation. He consistently guided me in the right direction making sure I take ownership of this work and checking me whenever he thought it was necessary.

I would like to thank Dr. Yezhou Yang and Dr. Dianne Hansford of the School of Computing, Informatics and Decision Systems Engineering at Arizona State University who were always there to give their valuable insights on this thesis. They have been present throughout my master's journey and this thesis could not have shaped as successfully as it did without their continuous encouragement.

I would also like to acknowledge Dr. Pavan Turaga and Dr. Suren Jayasuria as the external guide of this thesis, and I am gratefully indebted to them for their very valuable inputs during the early stages of the project.

Finally, I must express my very profound gratitude to my parents and to my sister for providing me with unfailing support and continuous encouragement throughout my years of study. Having a lab filled with colleagues like Venky, Ali, Megumi, Jinhan, Sridhar, Alex, and Vasudha made me look forward to work every single day. Thanks to Garvit, Arjun, Pradhuman, and Kunal for being my brothers away from home and making my life at ASU fun and enjoyable. Finally, this thesis would be incomplete without mentioning Juhi for always being there for me through thick and thin. This accomplishment would not have been possible without them. Thank you!

Siddhant Prakash

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

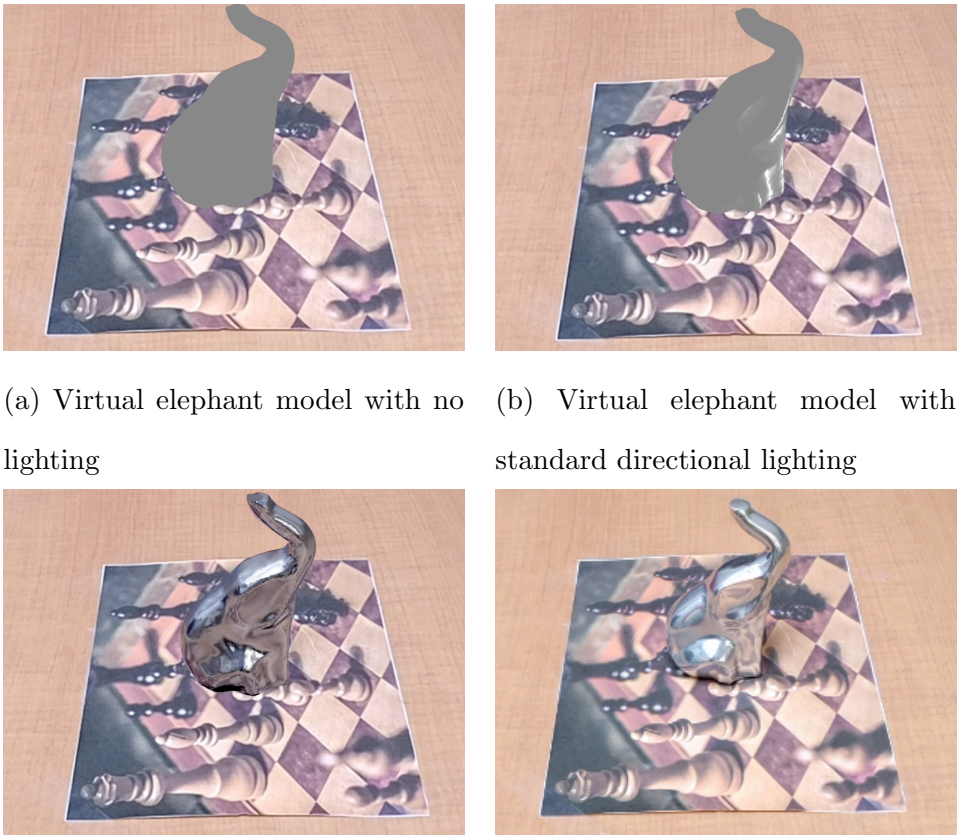
Chapter 1

INTRODUCTION

Light estimation is a critical component for realistic rendering of virtual scenes. For mixed-reality, a merging of virtual and physical worlds, accurate light estimation is especially important; inaccuracies in light estimation create noticeable visual inconsistencies in the lighting of the virtual scene and physical scene, as shown in Figure 1.1. Failure to properly render directional lighting, e.g., shadows and highlights, is particularly noticeable to users [8, 23, 36]. Such lighting inconsistencies remove the user from the immersive experience, whether or not the user is consciously aware of the inaccuracy in the scene illumination.

Sufficient light estimation requires not only intensity of light, but also directionality of light. Furthermore, light estimation must be updated in real-time, adjusting to changes in the dynamic environment of a physical setting. Consequently, while light estimation is on the forefront of the minds of mixed-reality application developers, current approaches to light estimation have thus far been inadequate. At this time, released Apple ARKit [3] and Google ARCore [10] implementations provide coarse illumination estimation through ambient light sensing of average pixel values in a scene. Meanwhile, advanced academic research solutions sample light transmissions from the scene geometry [27, 28, 40] and use machine learning inference to estimate directional light intensity [9, 21]. These solutions can be computationally expensive, slow to update, and prone to inaccuracy when filling in missing information.

Thus, in this work, we take a different approach: use reflections off of physical surfaces to estimate dynamic real-time illumination from all directions. By observing images of reflected light off of geometrically-tracked hand-held controllers, headsets,



(a) Virtual elephant model with no lighting

(b) Virtual elephant model with standard directional lighting

(c) Virtual elephant model with GLEAM (our) estimated lighting

(d) Real elephant figurine with real environmental lighting

Figure 1.1: Mixed-reality scene consists of virtual object(s) (in this case an elephant model) overlaid on a background camera image. GLEAM estimates lighting at the location where the virtual scene should be rendered which is determined by an image-marker (chess image in this scene).

game pieces, or other physical objects, a mixed-reality camera system can estimate the intensity of light associated with different angles of incoming light. A single camera viewpoint will provide a basis of illumination estimation, but can lack information not captured in its viewpoint. Thus, when available, multiple mixed-reality devices can share their viewpoint-specific data to jointly improve the illumination estimation.

To streamline and optimize the collaborative illumination estimation, we design our system *GLEAM*, which has the task of *Generating Light Estimation Across Mixed-reality devices*.

The contribution of GLEAM is to bridge the gap between offline techniques on desktops/servers and real-time techniques on mobile mixed-reality systems. Through various techniques, our GLEAM design prioritizes computational efficiency and real-time update. We describe various policies to allow application developers and/or users to utilize trade-offs between illumination resolution, coverage, and update interval.

Our design and implementation of GLEAM use the Unity Game Engine for portability across smartphones, tablets, and headsets. These devices can use GLEAM either as singular mixed-reality devices for single viewpoint illumination estimation or as networked mixed-reality devices for collaborative illumination estimation. Notably, GLEAM can theoretically work in tandem with existing light estimation techniques, providing continuous computationally inexpensive updates in real-time where others can provide additional high-fidelity illumination estimation through inference on scene geometry.

The effectiveness of GLEAM in generating realistic illumination is highlighted through our user study which showed an overwhelming preference (66.67%) towards GLEAM's estimates. GLEAM performed well in most of the scenes with users marking GLEAM illuminated scenes as more realistic. Even for scenes which were illuminated with directional light placed roughly near the actual physical light, the participants show confusion highlighting the level of realism achieved using GLEAM for estimating illumination.

In this thesis, Chapter 2 covers the background and related work of illumination estimation. Chapter 3 covers an overview of our GLEAM design. Chapter 4 describes policies for trade-offs between quality, density, and update interval. Chapter 5 de-

scribes our Unity-based multi-threaded implementation. Chapter 6 covers our system evaluation and user studies. In Chapter 7, we discuss future avenues of research and conclude our work in Chapter 8.

Chapter 2

BACKGROUND

2.1 Illumination estimation models

In computer graphics, rendering generates an image of a virtual scene captured from the perspective of a virtual camera. The image is formed when a ray of light originating at a light source irradiates the object and reflects back into the camera. The incoming radiance L at a spatial location $\mathbf{x}$ as observed from a direction ω is computed by solving the *rendering equation* [14]

$$L(\mathbf{x}, \omega) = \int_{\mathbf{\Omega}} f_r(\mathbf{x}, \omega_i, \omega) L_{in}(\mathbf{x}, \omega_i) cos\theta_i \partial\omega_i \tag{2.1}$$

where $\mathbf{\Omega}$ defines the upper hemisphere oriented around the surface normal at $\mathbf{x}$, ω_i is the incoming radiance direction, f_r is the bi-directional reflectance distribution function (BRDF), and θ_i is the angle between surface normal at $\mathbf{x}$ and ω_i, as shown in Figure 2.1. All modern rendering engines [7, 24, 31, 35, 37] are capable of rendering

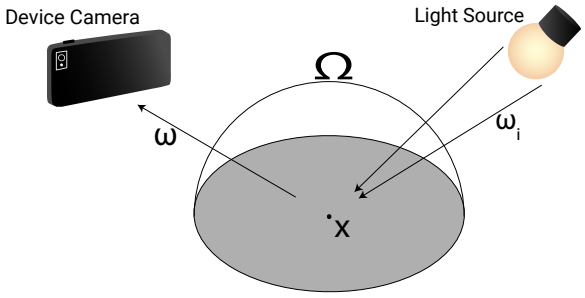


Figure 2.1: The total radiance coming out of a spatial point $\mathbf{x}$ is described by the rendering equation given an illumination model for incoming radiance ($L_{in}(\mathbf{x}, \omega_i)$) and a BRDF.

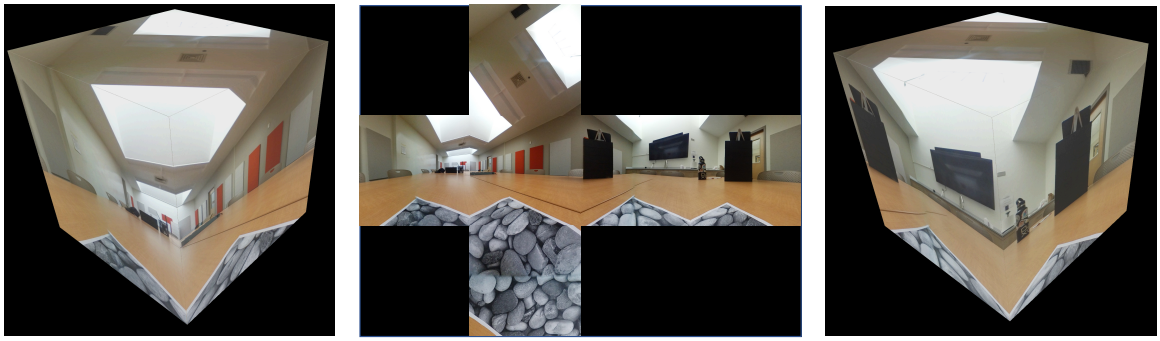


Figure 2.2: Cubemap representation (center) of environment maps. The figures on the left and right show the cubemap wrapped as a cube from two different viewpoints.

scenes from the BRDF of objects and an illumination model. Thus, we need to estimate the illumination model of incoming radiance $L_{in}(\mathbf{x}, \omega_i)$ to render an object at $\mathbf{x}$.

Illumination models are often formulated under the "distant scene assumption": the intensity of incoming ray depends on the direction of incidence only. Thus, modeling illumination boils down to mapping directions in the 3D space to ray intensity. Under the distant scene assumption, illumination models are usually represented in the form of environment maps, especially in Image-based Lighting [4]. Such environment maps consist of mapping of incoming ray intensity to ray direction, representing $L_{in}(\omega_i)$.

One of the most commonly used representations for environment maps is a "cubic environment map" or "cubemap", as shown in Figure 2.2. Each spatial location on a cubemap face maps to a discrete direction. Thus, mapping directions as the vector between the center of a unit cube and cubemap pixels, a cubemap is able to store intensities spanning the entire 3D space.

Specular materials reflect most of the light incident on the surface. Thus, to maintain details, environment maps should have reasonably high resolution. Although, it has been established that most scenes do not need high resolution in environment

maps for believable illumination, as in [26], having high resolution maps is necessary for specular objects, including liquids and mirror-like surfaces.

2.2 Related work

Illumination estimation is an extensively well-studied computer graphics problem. Here, we discuss various facets of illumination estimation.

2.2.1 Measuring and estimating lighting

Measured lighting captures accurate illumination, generally using physical reflection probes. Image-based lighting, introduced in [4, 5], uses reflection probes to generate High-Dynamic Range (HDR) environment maps. Further exploration through temporal image-based lighting [11, 13, 33] led to HDR or RGBD (RGB + Depth) videos of reflection probes being captured and used for illumination estimation over time. Sparse [1, 29] as well as dense [32, 34] sampling were employed through a combination of custom light probe devices and capture techniques to improve estimation for different situations. This thesis is inspired by these works; we introduce reflective objects for radiance sampling in our system design because of its effectiveness in capturing environmental illumination.

In *estimated lighting*, the goal is to measure lighting without the use of an explicit light probe device. Various works use image features to regress an illumination model [12, 18, 22]. Others have exploited human perception to approximate illumination [9, 15, 16, 17], which is not always accurate to the physical environment. Estimating illumination for outdoor scenes with perceptual or implicit methods can give very wrong results. To improve outdoor illumination estimation, explicit methods have been developed by [19, 20, 39]. We study their method of evaluating perceptual illumination, especially in the design of our user studies.

2.2.2 Vision for mixed-reality

Advances in vision tasks, such as structure-from motion (SfM), depth and pose estimation on mobile devices have led to the development of systems that exploits these to estimate lighting for real-time mixed-reality. Fish-eye cameras can model illumination from multiple-viewpoints and offloads computation to a PC server for estimating illumination [28]. Removing the need for fish-eye cameras, commercial RGB-D Kinect sensors can estimate illumination from 3D reconstruction of everyday objects [27]. Advancing these works for dynamic estimation, GLEAM is designed for commercial mobile devices and uses marker-based pose estimation to surpass need for 3D reconstruction methods.

Recent commercial interest in mobile mixed-reality devices are fueling the progress of mobile mixed-reality. For augmented reality illumination estimation Google Inc. has introduced the ARCore library [10] and Apple Inc. has released ARKit library [3]. Both ARCore and ARKit provide light adaptation to the scene lighting. However, the light adaptation is coarse, scaling pixel intensities with the average intensity of the scene. The beta for ARKit 2.0 includes real-time illumination estimation from camera inference. We compare GLEAM against these commercial products and report the comparison in the evaluation section (§6.2).

Chapter 3

GLEAM SYSTEM DESIGN

As mentioned in §2.1, sufficient rendering needs an environment map of illumination that represents the intensity of incoming light rays towards the scene. We design the GLEAM system to sample and interpolate points in the environment map by visually observing reflective objects across multiple mobile devices to model accurate scene illumination. The end result is a rich environment map that dynamically updates to continually reflect changes in the physical environment and scene objects.

The GLEAM system, illustrated in Figure 3.2, estimates illumination through the integration of multiple operations, including modules to: (*i*) sample illumination through reflective geometry, (*ii*) network devices for multi-viewpoint sample transfer, and (*iii*) generate environment maps through interpolation. GLEAM performs these operations on incoming camera frames to dynamically update the environment map. This section describes these components in further detail.

3.1 Sampling radiance through reflective geometry

Environment maps associate illumination radiance intensities and colors to the angular directions of the incoming light towards the scene. Captured images of specular objects with known spatial surface meshes can geometrically reveal such radiance information as the object surfaces reflect light into the camera. Thus, to capture radiance samples for an environment map, we use augmented reality positioning markers to spatially position specular objects in a physical scene. Figure 3.1 shows the use of a reflective rhino as a target specular object. The GLEAM system visually inspects images of these specular objects to compute illumination samples through reflective

Figure 3.1: GLEAM observes physical reflective objects to estimate accurate scene lighting. In this figure, GLEAM observes a reflective rhino figurine (top). The estimates are used to realistically illuminate a virtual rhino model (bottom).

geometry.

Specular reflection follows a strict geometric pattern: the angle of the reflected ray from a surface normal θ_{out} matches the angle of the incident ray from the surface normal θ_{in}. As illustrated in Figure 3.2a, GLEAM leverages this principle to estimate radiance samples using the following process:

1. Project a virtual ray from the pixel along its camera ray into the virtual scene.

2. Determine if and where a collision occurs between the ray and a specular surface in the virtual scene.

3. Reflect the ray over the specular surface normal to generate an incoming ray vector.

4. Associate the camera pixel color with the angle of the incoming ray into the

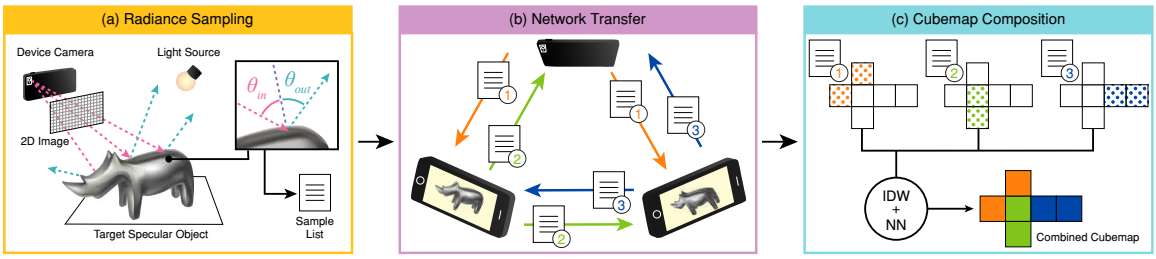


Figure 3.2: GLEAM illumination estimation system comprises three modules: (a) The **radiance sampling** module generates a collection of radiance samples by observing reflections off of a target specular object; (b) The **network transfer** module shares samples among multiple devices in the system; (c) The **cubemap composition** module interpolates collected radiance samples to create a combined high quality cubemap.

scene. This association becomes a radiance sample.

By performing this sequence of computations on each pixel in the scene, GLEAM captures a set of radiance samples with every camera frame.

As GLEAM's technique relies on reflective geometry, the shape of the specular object carries implications on the success of the technique. Concave surfaces – such as the neck and horns of the rhino – present difficulties due to their inter-reflections; camera rays reflect back into the object instead of into the environment. To mitigate this concern, concave surfaces can be removed from the virtual mesh of potential specular collisions. On the other hand, round convex surfaces work well with the reflective geometry, as they provide a variety of surface normals that GLEAM can use to generate samples for a broad set of illumination angles. For these reasons, metallic spheres work well as the target specular object. However, as we show through our implementation and evaluation, GLEAM works with multiple object shapes with and without concavities and round surfaces.

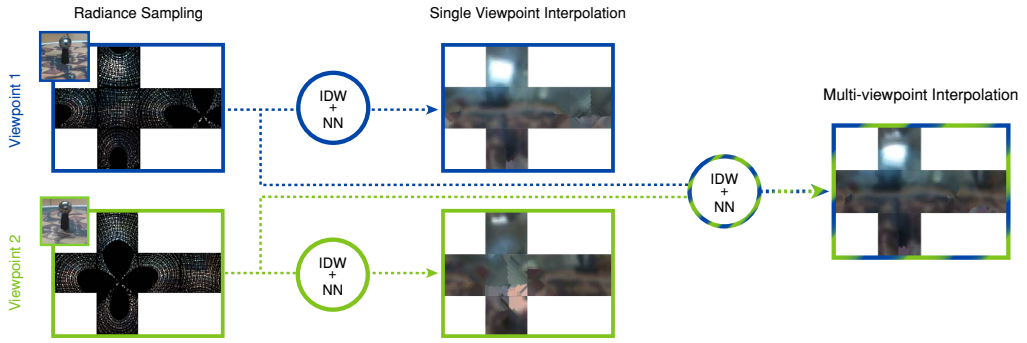


Figure 3.3: GLEAM computes radiance samples using a reflective object such as the specular ball shown in the inset. For each viewpoint, GLEAM can use Inverse Distance Weighting (IDW) interpolation and Nearest Neighbor (NN) interpolation to generate a cubemap from **single viewpoint interpolation**. Alternatively, to reduce the distortions from single-viewpoint capture, GLEAM can interpolate samples from both viewpoints through a high quality **multi-viewpoint interpolation**.

3.2 Networking devices for sample transfer

Radiance samples generated from a single viewpoint will only cover partial regions of the environment map. For full coverage, samples from multiple viewpoints can contribute to jointly populate the environment map. GLEAM uses a local network to share illumination information among multiple mobile devices, as illustrated in Figure 3.2b. Together, these mobile devices form a distributed multi-viewpoint system for radiance sample collection and distribution.

The requirements for networking are simple: upon sample generation, a GLEAM device will transmit sets of samples to all other GLEAM devices that observe the same target. Local multiplayer game engines typically adopt a client-server model, using the server to synchronize information among multiple clients. To remove the need for a dedicated server, the server behavior is often hosted on one of the client applications, which becomes a multiplayer "host". This and other standard network-

ing architectures satisfy the needs for transferring radiance samples among GLEAM devices.

Transferring GLEAM samples is constrained by network bandwidth. This can be mitigated by sending fewer samples over the network. We discuss various policies to reduce network overhead in §4.

3.3 Composing environment maps through sample interpolation

The generated and received radiance samples form a sparse estimation of illumination. To create a usable environment map, GLEAM spatially interpolates the samples into the environment map space, as illustrated in Figure 3.2c. While choosing interpolation algorithms, we need to consider not only the quality of interpolation, but also the computational overhead. This is especially important because the environment map updates on every newly processed list of samples, repeatedly incurring the interpolation overhead.

Environment maps typically use cubemap texture formats for omnidirectional representation. Cubemaps are composed of six faces – representing six faces of a cube – each a grid of "texels". Each texel represents the intensity corresponding to an angle of incoming light.

To fill the cubemap, GLEAM uses a combination of inverse distance weighting (IDW) interpolation [30] and nearest neighbor (NN) interpolation [2]. IDW interpolation operates on each texel, computing a weighted average of nearby samples, each sample weighted by its distance from the texel. We find the interpolated sample intensity u of texel $\mathbf{x}$, from nearby samples $u_i = u(\mathbf{x_i})$ for $i = 1, 2, ...N$ where N is the total number of neighborhood samples for texel $\mathbf{x}$, using the IDW function given by,

$$u(\mathbf{x}) = \begin{cases} \frac{\sum_{i=1}^{N} w_i(\mathbf{x}) u_i}{\sum_{i=1}^{N} w_i(\mathbf{x})}, & \text{if } d(\mathbf{x}, \mathbf{x_i}) \neq 0 \quad \text{for all } i; \quad w_i(\mathbf{x}) = \frac{1}{d(\mathbf{x}, \mathbf{x_i})} \\ u_i, & \text{if } d(\mathbf{x}, \mathbf{x_i}) = 0 \quad \text{for some } i \end{cases}$$

For low-complexity, we use Manhattan Distance as our weighting function given by,

$$d(\mathbf{x}, \mathbf{x_i}) = ||\mathbf{x} - \mathbf{x_i}||_1 = \sum_{j=1}^{n} |\mathbf{x}_j - \mathbf{x_i}_j|$$

To computationally perform IDW interpolation, we iterate over our list of samples, adding each sample's weighted intensity value and distance weight to all cubemap texels within a neighborhood radius. We then iterate over the cubemap texels, dividing the sum of weighted pixel values by the sum of distance weights to generate the interpolated texels. Larger IDW neighborhood radius parameters allow IDW to fill greater portions of the cubemap, but come at the expense of computational time.

IDW will leave cubemap gaps in texels that do not occupy any sample neighborhoods, especially with smaller IDW radiuses. To fill the remaining gaps (texels having no value), we use a nearest neighbor algorithm to assign missing cubemap texels to the values of their nearest assigned neighbors.

IDW and nearest neighbor are two of many interpolation mechanisms that can satisfy the needs for cubemap interpolation. Other strategies, e.g., structural inpainting, hole filling algorithms, or neural network-based methods, are also viable solutions, with potentially higher quality at the expense of computational complexity. To prioritize for reduced computational complexity, we leave the full exploration of such algorithms as a future research avenue. By interpolating radiance samples into a full cubemap on a per-frame basis, the GLEAM system provides a dynamically updating environment map for the renderer to use to illuminate a scene. We demonstrate the combined working principles of the GLEAM system in Figure 3.3.

Chapter 4

POLICIES FOR TRADE-OFFS BETWEEN QUALITY FACTORS

GLEAM environment maps provide dynamic illumination estimation for augmented reality rendering. However, the quality of the rendering relies on the nature of the generated environment maps, especially with regard to the update interval, resolution, sample coverage, and dynamic freshness of the estimated illumination, as defined in §4.1.

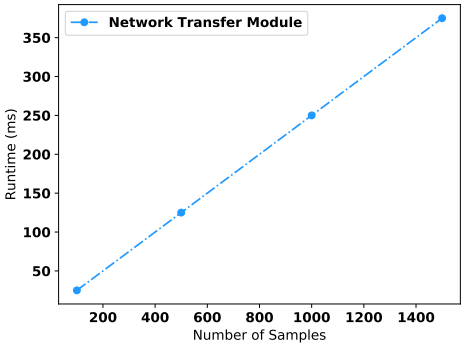
Ensuring these aspects of quality is constrained by system limitations in computational and networking resources. Built on an understanding of these overheads, we provide tradeoff policies in §4.2 that allow developers using GLEAM to dynamically prioritize various aspects of quality, based on the needs befitting their specific augmented scenes.

4.1 Quality factors

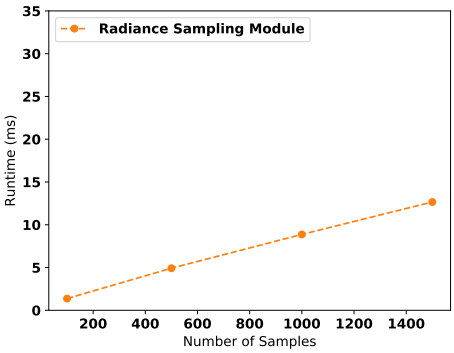
4.1.1 Update interval: temporal resolution

To model a dynamic scene with changing illumination, GLEAM will periodically refresh its illumination estimation after an update interval. With smaller update intervals, dynamic changes in scene lighting will be reflected in the virtual scene, creating visual harmony between the physical and virtual scene. However, the update interval of the cubemap is limited by the bandwidth of the network channel and the computational runtime of the system.

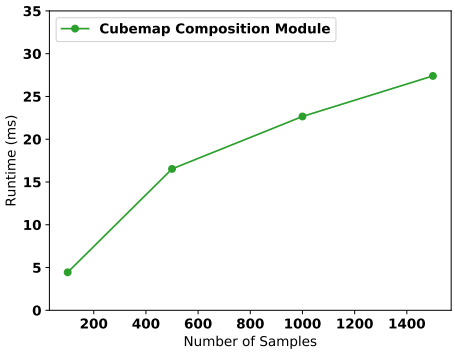
With a fixed network bandwidth B, the update interval U is limited by the amount



(a) Time taken to transfer samples shows linear increase with number of samples at constant bandwidth.



(b) Sampling radiance takes more time with increase in number of samples generated due to iterative ray casting.



(c) Cubemap composition shows a dip in this linear trend for larger number of samples because of reduced load.

Figure 4.1: Runtime characterization of (a) Network transfer module, (b) Radiance sampling module, and, (c) Cubemap composition module of our GLEAM implementation as a function of number of samples.

of data transferred per update interval $S_{network}$:

$$S_{network}/U \leq B$$

Thus, to maintain a low update interval, GLEAM must limit the number of radiance samples being transferred among devices, which can be observed from Figure 4.1a.

Similarly, the update interval must be larger than the per-frame computational runtime of the GLEAM system. GLEAM must limit the number of radiance samples being generated and being used to compose an illumination map, as shown in Figure 4.1b, 4.1c. As seen in the figure, the network transfer overhead outweighs computational overhead by an order of magnitude, and is thus the bottleneck to rapid update intervals.

We further explore the effect that decreasing the number of samples has on the environment map quality through three other factors, i.e. resolution, coverage and freshness, in order to maintain interactive updates.

4.1.2 Spatial resolution: density of samples

A high resolution environment map, i.e., a cubemap with dense faces of texels, affords the ability to represent illumination details, resulting in high fidelity rendering of highlights and shadows from the environment. However, simply increasing a cubemap's resolution will not automatically improve the quality of the cubemap; a high resolution cubemap needs a dense collection of radiance information detail to populate the cubemap.

GLEAM populates cubemap texels with interpolated combinations of nearby samples. Naturally, a sparse sampling will yield averaged values, whereas a dense sampling will yield a more precise interpolation, due to smaller distances from texels to radiance samples. The need for sampling is exacerbated for high resolution cubemaps, as a

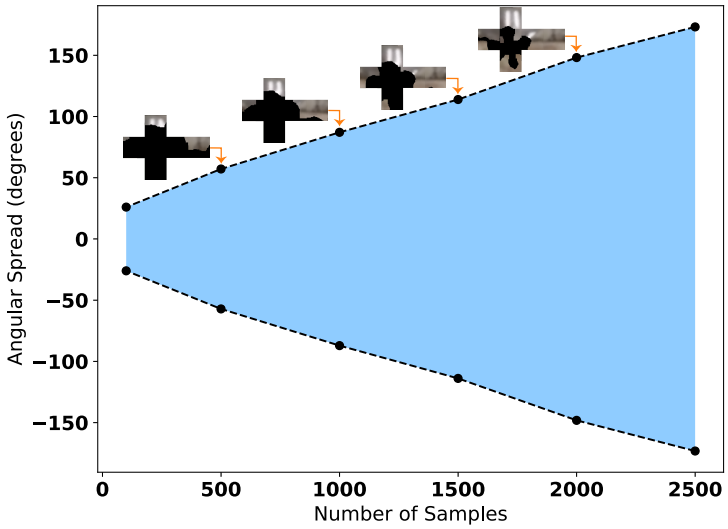


Figure 4.2: Capturing more samples gives GLEAM a wider angular spread of illumination estimation for composing the environment map.

poor sampling density will result in each radiance sample contributing to large neighborhoods of texels. Thus, raising the resolution of the environment map demands a high density of radiance sampling to more accurately populate each cubemap texel.

4.1.3 Coverage: angular deviation of samples

Successful environment maps represent incoming illumination from all directions, allowing graphics engines to render reflections of environmental light off of various virtual material surfaces. Therefore, sample coverage is an important quality aspect; environment maps should embody information from a wide angular deviation of radiance samples.

We can measure the amount of coverage of samples by calculating their angular deviation from the viewpoint of device. We characterize the number of samples generated with their angular deviation in a single Full HD capture of a specular sphere

in Figure 4.2. Through sample transfers over a network, multiple viewpoints can contribute their coverage of samples to construct an environment map with reasonably high coverage. In the absence of multiple devices, and/or for any remaining gaps in coverage, environment map interpolation will fill in the holes with averaged neighboring sample data. However, this is subject to inaccuracy, especially for wide gaps in coverage.

4.1.4 Freshness: age of samples

GLEAM continually updates the environment map with newly captured samples and samples received over the network. Retaining and accumulating samples over time can help GLEAM raise the resolution and coverage of the environment map as devices move around to capture different spatial perspectives of the object. However, samples should not be indefinitely retained; the relevancy of the sample may degrade with time as an environment's lighting and objects dynamically change.

Many natural situations can cause dynamic environmental changes; a light may turn off, a passerby may cast a shadow, or a nearby placed object may introduce colored reflections off of its surface. By retaining samples that took place before these dynamic changes, the environment map may falsely incorporate stale illumination information. Thus, the age of the radiance sample is an important measure of the freshness of the environment map.

4.2 Situation driven policies

Different lighting conditions as well as environmental changes over time may require optimization for different factors. Based on the scene composition, number of participants in the system, and network bandwidth, GLEAM can be optimized to achieve different policies. We define three criteria: (i) number of samples per map,

(*ii*) pixel skip factor, and (*iii*) sample expiration period, to define and evaluate three policies that prioritize resolution, coverage, and freshness.

4.2.1 Resolution priority

GLEAM can be configured to optimize for maximum cubemap resolution for high sample density. Optimizing for high resolution provides us with a high quality environment map. High quality environment maps are suitable for scenes that exhibit complex lighting phenomena. For example, the simulation of liquids requires high quality environment maps to accurately model light interactions with the liquid surface.

To achieve this policy, we use all possible generated radiance samples to compose an environment map. Furthermore, we allow samples to persist over time, allowing sample age to increase.

However, as the raised number of samples increases computational overhead, a maximum resolution policy sacrifices update interval, creating environment maps that are slow to update. This can be mitigated by reducing the age of samples to create reasonable updates.

4.2.2 Coverage priority

Having a full coverage is desired in almost all situations, fully capturing the illumination in the environment. Using collaborative sensing, GLEAM can obtain greater coverage of the scene from most angles. However, constrained by the network bandwidth, there is a limit on the number of samples that can be transferred in a given update interval. Thus, GLEAM must discard samples prior to transmission to allow for complete coverage of the scene.

Samples of early indices are concentrated in the center of the viewpoint, reaching

wider angular deviation with higher-indexed samples. To allow a fixed number of samples to cover a greater spread of angles, we subsample our list, e.g., skipping every other sample, to allow larger angular deviations.

Notably, using this policy, if the number of devices in the system increases, the number of samples required to transfer decreases along with increase in coverage. This reduces the network burden, allowing faster updates.

4.2.3 Freshness priority

GLEAM is also designed to model dynamic scene lighting in mixed-reality. Some dynamic environments may call for rapid updates to illumination conditions, e.g., an AR museum showing a 3D model exhibit with constant movement of people around the exhibit. GLEAM can achieve this freshness of illumination by purging samples above a certain age threshold.

When purging old samples, GLEAM reduces the number of samples that are used to compose its cubemap, retaining only the most recent samples. In addition to keeping the illumination estimation fresh, this promotes reduced update intervals. Unfortunately, this policy sacrifices cubemap resolution and coverage.

Chapter 5

IMPLEMENTATION

5.1 Unity-based Implementation

We develop software through the Unity 3D game engine [35] and PTC Vuforia [6] to provide the graphics rendering and augmented reality tracking infrastructure for our GLEAM implementation. Unity supports cross-platform deployment, which allows us to harness the versatile design of GLEAM for Android, iOS, macOS, and Windows deployments. As of this writing, GLEAM has been designed and tested on Nvidia Shield K1 Tablet, iPhoneX, iPad 10.5 inch, Samsung Galaxy S8, OnePlus 3T, Macbook Pros, and Windows 10 computers.

5.1.1 GLEAM environment mapping

In our implementation, GLEAM uses Vuforia SDK's marker-based pose estimation to track reflective objects. Vuforia, in addition to generating pose estimates, also provides the camera frame that is used to obtain the correspondences between the camera and an image marker. When generating radiance samples, we extract the pixel intensities of the reflective objects from this frame.

GLEAM represents environment maps as Unity's `Cubemap` objects. `Cubemap` objects in Unity are *Texture2D* objects indexed using a `CubemapFace` value and two floating-point values for spatial location on each cubemap face. The intensity of each sample in the `Cubemap` is stored as Unity's `Color32` object, which stores the red, blue, green and alpha channel intensities as a byte value within the range of 0-255. The color and intensity of the corresponding pixel obtained from the camera frame is used

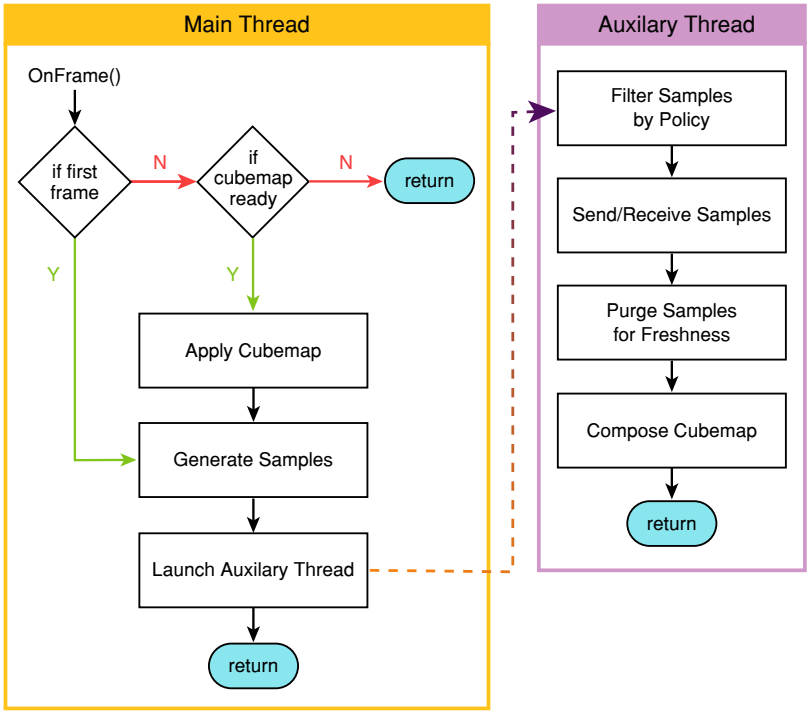


Figure 5.1: Multi-threaded GLEAM implementation. An auxiliary thread keeps the main thread free for interactive display frame rates.

as the intensity of the radiance sample generated. To store the direction, samples store a single `CubemapFace` value and two spatial floating-point values, which are directly used as `Cubemap` indices.

5.1.2 Reflective objects and meshes

To generate samples, GLEAM uses a reflective object placed at a known spatial location with respect to an image marker. A 3D mesh of the reflective object is placed at the same location with respect to the marker in the Unity scene. Currently, this requires calibration to align the 3D mesh with the real object. Tracking reflective objects is an active research challenge [25, 38], which falls outside the scope of our work.

5.1.3 Multi-device sample transfer

GLEAM uses Unity's UNet High Level API to transfer samples between devices. Using this API, GLEAM spawns a new `GameObject` corresponding to a new device when it joins the network. This GameObject holds a locally created sample list along with a collection of sample lists from other devices in the system. Each device generates a new sample list from an input frame and broadcasts it to all other devices in the network. Using this implementation, GLEAM is able to hold multiple lists locally, from which it can select samples to compose the cubemap. Thus, only one sample list needs to be transferred at a given time, reducing network cost.

5.1.4 Cubemap composition

After the transfer of samples is complete, GLEAM uses collected sample lists to compose an environment map. All samples which are to be used in the environment map are composed into a Unity `Cubemap` object. GLEAM uses a Unity material which has a "skybox shader" as the environment map to light the scene. By setting the main texture of the material to the cubemap, GLEAM achieves realistic lighting in every frame.

5.1.5 Multi-threading

To achieve interactive display frame rates and smooth cubemap updates, we employ multi-threading, as shown in Figure 5.1. Unity's *main thread* includes operations to compute the game state and render frames to the screen. Thus, to preserve fast frame rates, we aim to minimize operations performed on the main thread. Sample generation requires main thread operation to perform game physics raycasting. Cubemap application requires main thread operation to influence rendering opera-

tions. All other GLEAM operations, e.g., sample network transfer, environment map composition, are performed on an auxiliary thread so as not to block the main thread during operation. As we later show in our microbenchmarking, this sufficiently allows frame rates, limited only by the overhead of Vuforia tracking.

Chapter 6

EVALUATION

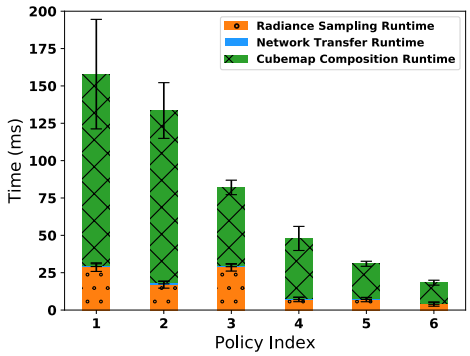
In this section, we evaluate our overall system on generating realistic illumination estimates for scenes targeted towards AR use-cases. §6.1 characterizes the runtime of GLEAM operations. Then, in §6.2, we perform qualitative analysis via multiple user studies that aim to answer the question, *"If we render an augmented scene with our illumination estimates, how realistic do virtual objects look?"* Since our system is targeted for AR use-cases, our evaluations are done on static as well as dynamic scene lighting.

6.1 Microbenchmarks

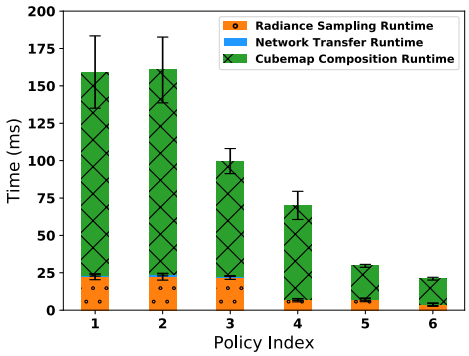
We execute microbenchmarks for different modules in our GLEAM implementation and observe how different policy decisions affect the individual runtime of each module and the overall computational runtime of the system. To analyze the performance implications of various trade-offs, we design six different policies with different combinations of quality factors, as reported in Table 6.1. We perform our comparative analysis of these policies on an Nvidia Shield K1 Tablet, a Samsung Galaxy S8, and an iOS iPad Pro 10.5" 2017 (A1709), shown in Figure 6.1. We report the execution runtimes of modules using Nvidia Shield K1 Tablet in Table 6.2.

6.1.1 Computational overhead per module

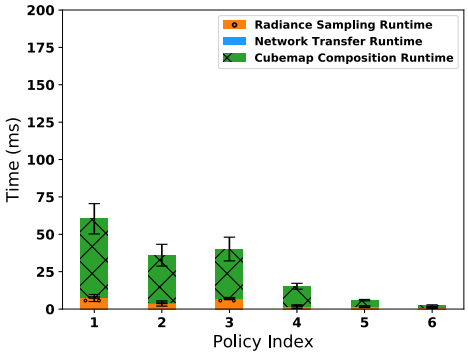
The computational overhead of the cubemap composition module dominates the overall execution time, consuming more than **82**% of GLEAM's execution time across all devices and policies. This is primarily due to the iterative computational expense



(a) Nvidia Shield (Android N)



(b) Samsung Galaxy S8 (Android O)



(c) iPad 10.5 inch (iOS 12.0)

Figure 6.1: The figure reports runtimes of different modules in our GLEAM implementation on multiple mobile devices. Each bar corresponds to 6 different policy for quality factors mentioned in §4.1. Runtimes corresponding to the three modules: radiance sampling, network transfer and cubemap composition are shown corresponding to each policy. We observe a reduction in runtime as the number of samples decreases, skipping factor increases, and demand for freshness increases. Execution runtimes on Nvidia Shield K1 Tablet with standard deviations over 100 runs are reported in Table 6.2.

of interpolation across cubemap texels.

Meanwhile, the computational overhead of the radiance sampling module only consumes up to **16.9**% of GLEAM's execution time. Radiance sampling only requires a constant set of geometric operations for each image pixel destined to be a radiance sample.

The execution time of the network transfer module is negligible. The Unity engine handles network transmissions on a separate non-blocking thread, allowing minimal execution overhead. However, while it does not block execution, the transfer itself is not instantaneous, taking 0.25 ms per sample, as discussed in §4.1.1.

6.1.2 Policy implications on execution time

The execution time of the radiance sampling module and cubemap composition module are both related to the number of samples, as shown in §4.1. We see the effect of this implication on the execution time of the different modules and the overall execution time.

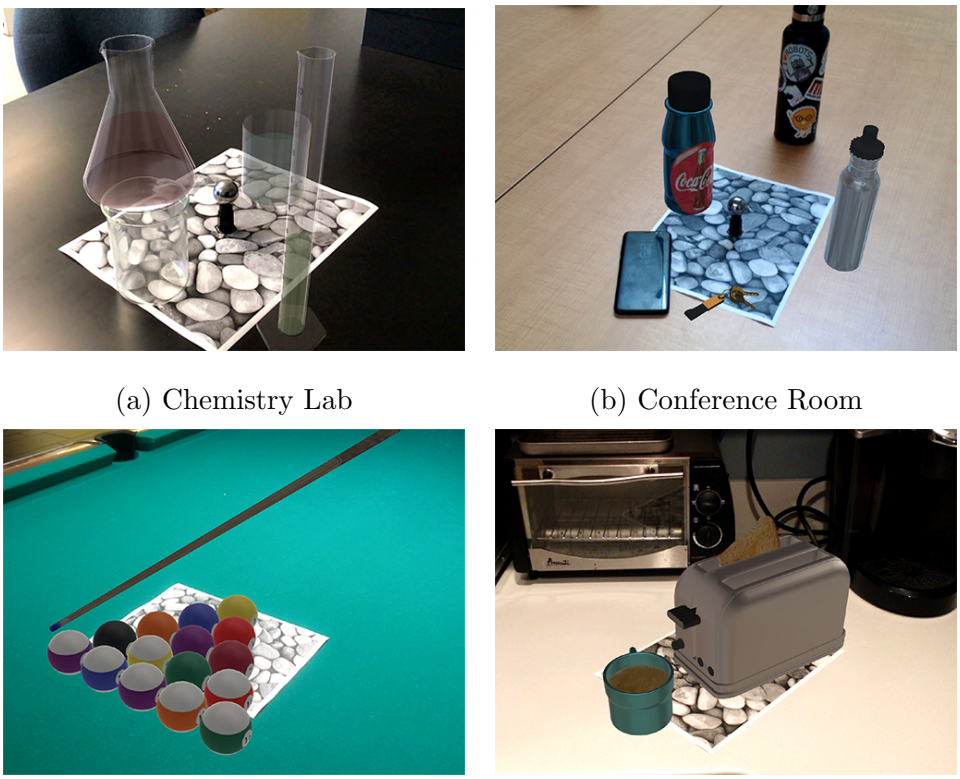
Policies that prioritize resolution increase execution time, due to the increased number of samples. Meanwhile, policies that prioritize coverage allow faster sampling intervals by decreasing the number of samples. To fully minimize update interval, the policies that prioritize freshness retain very few samples, further reducing execution time. This noticeably reduces the overhead of cubemap composition, allowing for rapid update cycles. For each policy, we assign conservative update intervals to relieve the relative computational burden of GLEAM for smooth application execution. We study the experiential effect of these policies on user perception in §6.2.2.

Table 6.1: Policy configurations used for evaluation.

Policy	#Samples per List	Skipping factor	Age (ms)	Update Int. (ms)
1: Resolution Priority	4000	0 pixel	1000	1000
2: Resolution Priority	2000	0 pixel	1000	500
3: Coverage Priority	4000	1 pixel	500	1000
4: Coverage Priority	800	2 pixels	500	200
5: Freshness Priority	800	0 pixel	25	200
6: Freshness Priority	400	1 pixel	25	100

Table 6.2: Execution runtimes on NVIDIA Shield.

Policy	Radiance Sampling (ms)	Network Transfer (ms)	Cubemap Composition (ms)
1: Resolution Priority	28.71 ± 2.86	1.31 ± 0.91	127.86 ± 36.66
2: Resolution Priority	17.05 ± 2.37	1.29 ± 0.82	115.2 ± 18.64
3: Coverage Priority	28.6 ± 2.49	1.26 ± 0.84	52.22 ± 4.86
4: Coverage Priority	7.04 ± 1.46	0.77 ± 0.83	40.12 ± 8.07
5: Freshness Priority	7.03 ± 1.48	0.66 ± 0.59	23.29 ± 1.70
6: Freshness Priority	3.99 ± 1.39	0.58 ± 0.69	13.77 ± 1.65



(a) Chemistry Lab

(b) Conference Room

(c) Billiards Table

(d) Kitchen

Figure 6.2: Different scenes and lighting environments used to evaluate perception of rendering with GLEAM estimation. (a),(b): indoor scenes with incoming natural daylight. (c),(d): indoor scenes with LED lights.

6.2 Qualitative user studies

6.2.1 Realism

With the help of a qualitative online user study, we evaluated the realism of a virtual scene illuminated using GLEAM's illumination estimation. The user study focused on the static visual fidelity of the virtual objects, as well as the adaption of virtual objects to illumination changes in the environment.

The scene setup for the online user study included 2 naturally lit indoor scenes and

2 artificially lit indoor scenes as shown in Figure 6.2. Surrounding physical objects included table surfaces, appliances, and glassware among other objects. For the online user studies, we used the maximum resolution policy for the static augmented scenes as well as dynamic composed videos.

For targeting static scene lighting, we created a dataset of images inserted with virtual objects in the real scenes. Users were asked to compare virtual objects illuminated with GLEAM's estimated environment map with those illuminated by (a) a scene with no illumination, (b) a scene illuminated using a single white directional light, (c) a ground truth environment map, and (d) ARKit 1.0 light estimation. The ground truth environment map was obtained using a Samsung Gear 360 panoramic capture with the camera placed in the scene at the point where the illumination is to be estimated.

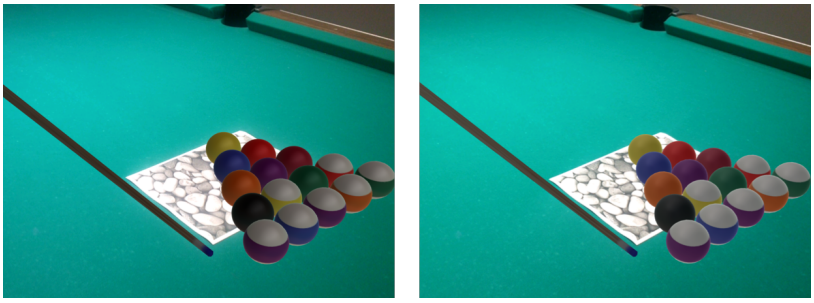
We used 26 pairs of images from the 4 different categories for the study. The images presented were similar to the 4 pairs of images shown in Figure 6.3. The users were informed that either one or both of the images were illuminated with correct illumination estimates. The users had to choose which image looked more realistic of the pair or indicate both as equally realistic.

To study how dynamic environmental illumination affects perceived realism, we added two one-minute videos to our online study. The videos were illuminated using GLEAM's estimates with a resolution priority policy. The participants were asked to indicate if they perceived the illumination changing similarly and in sync with surrounding objects along with if they found the overall scene realistic.

A total of 99 participants took the online user study. Overall we see a preference in illumination estimated by GLEAM with **66.67%** of users preferring GLEAM across all image pairs. We see the preference of GLEAM in all 4 lighting categories, as shown in Figure 6.4a.

(a) No Lighting (left) vs. GLEAM (right)

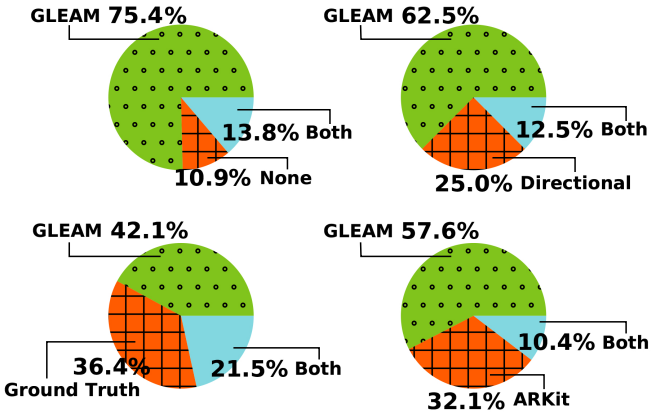
(b) GLEAM (left) vs. Directional Lighting (right)



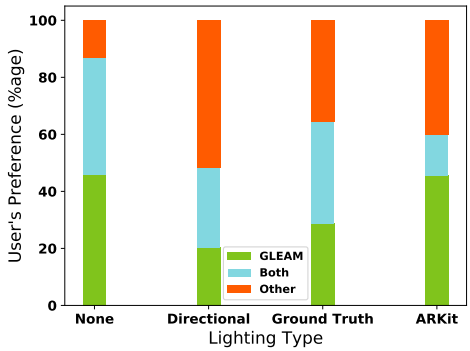
(c) Ground Truth Lighting (left) vs. GLEAM (right)

(d) ARKit 1.0 (left) vs. GLEAM (right)

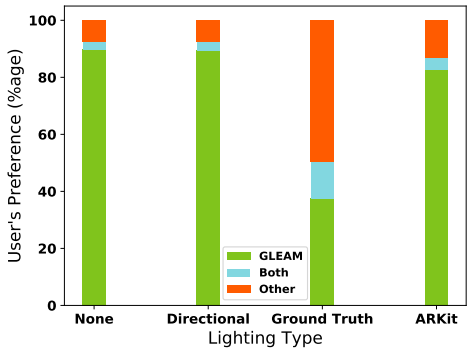
Figure 6.3: 4 of the 26 questions from the online survey in which user were asked to choose the more realistic scene from each pair of images.



(a) Overall user perception towards different lighting conditions for perceived realism. Users find scenes illuminated by GLEAM estimates to be more realistic compared to no lighting, directional lighting, ground truth, and ARKit 1.0 lighting.



(b) User preference against different lighting conditions for the **billiards table** scene.



(c) User preference against different lighting conditions for the **conference room** scene.

Figure 6.4: Breakdown of results of the online user study showing user perception of realism based on the 4 categories of lighting compared, i.e (i) GLEAM vs. No lighting, (ii) GLEAM vs. Directional lighting, (iii) GLEAM vs. Ground truth lighting, and, (iv) GLEAM vs. ARKit 1.0 lighting.

In the conference room scene (Figure 6.2b, Figure 6.3b, Figure 6.4c), a significant **88.28%** of users preferred GLEAM over other illumination alternatives, likely due to GLEAM's ability to capture the complexities of the skylight illumination in the virtually reflective objects.

Notably, not all scene comparisons favored GLEAM. In particular, in the billiards scene (Figure 6.2c, Figure 6.3c, Figure 6.4b), users preferred directional lighting over the GLEAM lighting (and apparently over the ground truth lighting). We attribute this outcome to our orientation of the scene's virtual directional light, which we positioned in the same direction as the physical LED lights in the environment, as well as user expectation to see strong directional glints off of round spheres. This motivates a possible future direction to enhance realism by inferring accurate directional lighting from GLEAM's estimated environmental lighting.

For the two dynamic videos, we observed mixed responses with users indicating the first scene (conference room) as more dynamic (**72.22%**) and more realistic (**61.61%**) while the other (chemistry lab) as less dynamic (**42.42%**) and less realistic (**24.24%**). This behavior could be due to the jitter of hand-held tracking, which made the lighting changes in the chemistry lab scene unrealistic.

6.2.2 Policy trade-offs

The perceived effect of illumination estimation should differ based on different scene illumination. To evaluate this, we conducted an in-person user study in a controlled environment (shown in Figure 6.5) to understand how different policies affect visual perception. The different configurations used to study the policies on the constant network bandwidth of 64kB/sec. are summarized in Table 6.1.

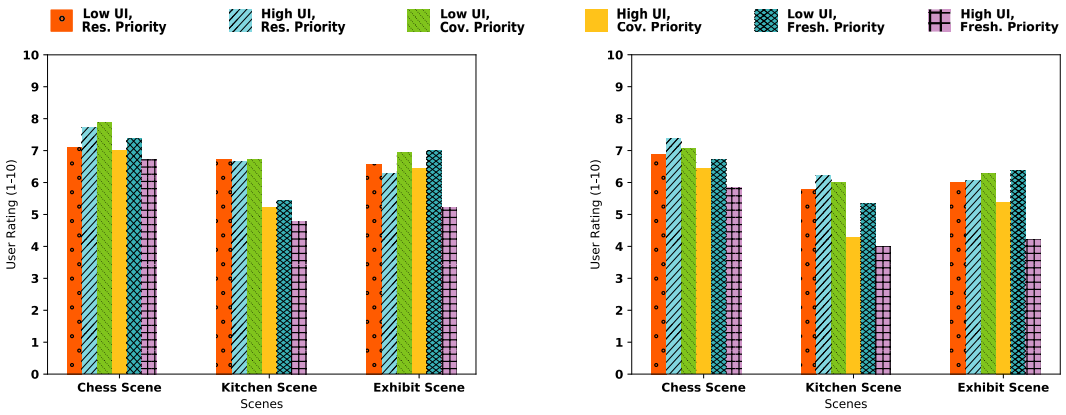
We recruited 18 participants for this study who were asked to observe our system for 3 different scenes on Nvidia Shield tablets. The participants were paired up and we

Figure 6.5: Environment used for in-person user studies. The three scenes were illuminated with dynamic colored lights and/or static white ambient lights.

used multi-viewpoint GLEAM with each participant contributing to the illumination estimation.

The lighting on the scenes was varied between a) static directional lighting, b) dynamic directional lighting and c) ambient light + dynamic directional lighting. The participants were asked to observe each scene with all 6 policy configurations defined in §6.1. Each scene was rated on the perceived visual fidelity and dynamic smoothness of the system on a scale of 1-10 for every policy. For consistent rating, each participant also experienced the scene without lighting and with incorrect directional lighting. To establish a baseline, participants were asked to give a rating of 0 if the scene looked like it had no lighting, 3-4 with an incorrect directional lighting and 10 if it looked like real objects.

The survey revealed a general preference towards resolution and quality of illumination. As seen in Figure 6.6, we observe both configurations with resolution priority



(a) User-perceived visual fidelity across policy.

(b) User-perceived smoothness of lighting changes across policy.

Figure 6.6: Breakdown of user study results on perceived visual fidelity and smoothness of different policies on 3 different scenes {chess scene, kitchen scene, exhibit scene} under 3 different lighting conditions {static directional, dynamic directional, and static ambient + dynamic directional}.

and low update rate with coverage priority were often rated higher than freshness priority. This indicates users' acceptance towards high visual fidelity, even on the cost of low update intervals. This result concurs with our results from the online study, revealing realism is often associated with the quality of lighting information provided in the scene.

6.2.3 Comparative analysis with ARKit

Recently, the Apple ARKit 2.0 Beta released a module for augmented environment map sensing. This puts GLEAM in contention with ARKit 2.0's illumination estimation system. Notably, GLEAM is compatible with all forms of mobile computing devices with cameras, whereas ARKit is limited to modern iOS devices.

As a second component of our in-person user study, we compared GLEAM with

Table 6.3: User perception of ARKit 2.0 illumination estimation module vs. GLEAM.

	GLEAM (#users)	ARKit 2.0 (#users)
Visual fidelity	9/18 (50%)	9/18 (50%)
Smoothness	12/18 (66.7%)	6/18 (33.3%)

an implementation of ARKit 2.0's environmental sensing. We deployed marker-based AR applications using ARKit 2.0 having environmental sensing and GLEAM. Both applications used the same scenes running on an iPad 10.5 inch with iOS 12.0. The participants were asked to observe the lighting on different scenes on both systems and indicate their preference on perceived visual accuracy and dynamic smoothness. The results are summarized in Table 6.3.

GLEAM was marked as equally realistic in terms of visual fidelity as compared to ARKit with users choosing either option **50%** of the time. For dynamic smoothness, users preferred GLEAM with a **66.7%** majority. This shows that our system achieves realistic as well as dynamic estimation against the commercial solution.

6.3 Post-evaluation analysis

Our evaluations prove efficacy of the GLEAM system in providing realistic illumination for virtual scenes. Through runtime characterization of modules on different devices, we observe the number of samples as being the decisive factor in determining the execution time of individual modules. However, as we observe through our characterizations in §4.2, network transfer is a substantial bottleneck, taking more time than all 3 computational modules put together.

Our user studies reveal interesting trends in perceptual realism. Although a majority of users indicate GLEAM as their preferred choice for illumination modeling,

the high preference for directional lighting in the billiards scene makes us realize that directional lighting is as important as ambient estimated lighting for a feeling of realism. With the in-person user study, participant preference for high resolution and visual fidelity indicates the importance of correctly lit virtual objects too. Moreover, the studies reveal the importance of visual fidelity over dynamic update in scene illumination, which is an unexpected outcome and needs to be studied in more detail.

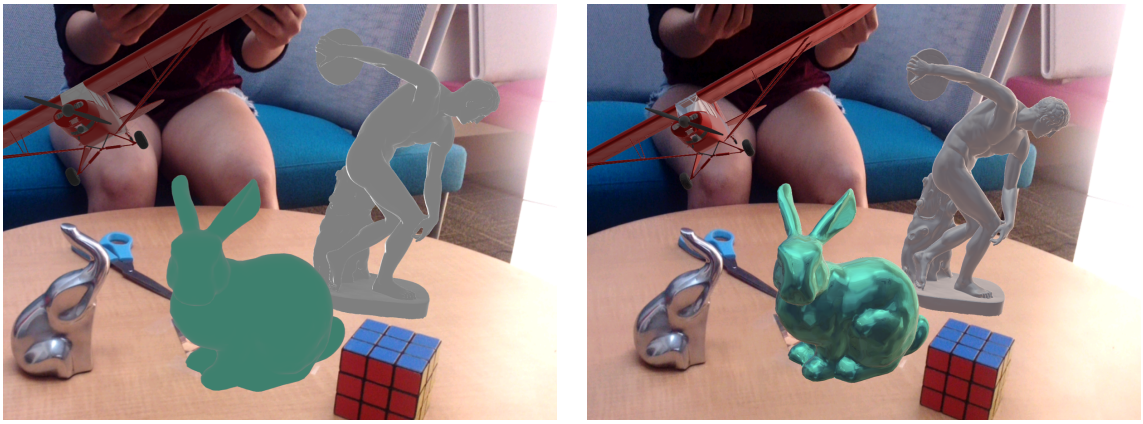
Chapter 7

CONCLUSION

Through our system design and implementation we demonstrate the power of GLEAM in bringing mobile mixed-reality closer to physical reality. Still, GLEAM is only an early step to illumination estimation, serving as a framework to open the door for several future opportunities, some of which we discuss here:

Distributed sampling for smoother collaborative sensing: Radiance sampling from multiple overlapping viewpoints leads to redundant sampling, i.e., the same portions of the cubemap are captured multiple times. We can exploit this redundancy to optimize for different qualities. For *dynamic range*, different devices can capture radiance samples at different exposure settings, collectively revealing illumination details at a wider range of intensities. For *low update intervals*, devices can opt not to send samples in overlapped regions to reduce the networking bottleneck of sample transfer. Deeper investigation into GLEAM workload distribution across devices and infrastructure, e.g., edge computing, could also reveal interesting opportunities towards distributed illumination estimation and rendering.

Integration into rendering pipeline: In our Unity and Vuforia-based GLEAM implementation, tracking positions and camera frames for radiance sampling are processed after the game engine renders a frame. This makes GLEAM estimates stale by the time they are used. To synchronize illumination estimation with rendering, we plan to investigate a deeper integration of GLEAM into the capture and rendering pipeline. By inserting computationally inexpensive – and potentially approximate – estimates early into the pipelines, we can reduce the latency of updates. This will



(a) Standard directional lighting.

(b) GLEAM estimated lighting.

Figure 7.1: A typical mixed-reality scene with multiple virtual objects made of different materials (plastic airplane, marble statue, and metallic bunny) with standard directional lighting compared against GLEAM estimated lighting. The realism imparted to the scene due to estimated lighting is apparent.

further improve the dynamic nature of GLEAM's realistic lighting.

Automatic specular object tracking: GLEAM currently requires pre-defined and calibrated target specular objects. We imagine that a future extension of the same illumination estimation principles would leverage *existing* specular objects within an environment. In such a system, the camera device, perhaps on a wearable headset, would continually identify, track, and observe reflective objects, inferring object meshes to compute reflective geometry. Detecting reflective objects is a non-trivial problem, but has seen recent advances [38]. In addition to the obvious benefit of improving portability across infrastructure – obviating the need to carry a metal object with you – a wider reflective sampling would allow illumination to be sampled for multiple spatial points in the scene, i.e., the $\mathbf{x}$ in $L_{in}(\mathbf{x}, \omega_i)$ from Equation 2.1.

We present GLEAM, a system which estimates environmental lighting to illuminate a virtual scene with accurate scene illumination and achieve visual realism as

shown in Figure 7.1. The GLEAM system comprises three major modules: (*i*) radiance sampling, to generate radiance samples using reflective geometry; (*ii*) network transfer, to share radiance samples among multiple participating devices; and (*iii*) cubemap composition, to interpolate an environment map from the accumulated radiance samples. We presented trade-offs between update interval and visual fidelity to optimize for quality factors of resolution, coverage, and freshness based on situational need.

Our implementation provides an operational prototype of the GLEAM system, based on the Unity Game Engine and Vuforia library. We evaluated GLEAM by microbenchmarking the execution times of different modules and by conducting user studies, finding that users favor the perceptual realism of the virtual scene rendered using GLEAM's estimates. This work on real-time illumination estimation on mobile and wearable systems thus contributes a step towards a richer immersive integration between virtual and physical worlds.

REFERENCES

[1] F. Banterle, M. Callieri, M. Dellepiane, M. Corsini, F. Pellacini, and R. Scopigno. Envydepth: An interface for recovering local natural illumination from environment maps. In *Computer Graphics Forum*, volume 32, pages 411–420. Wiley Online Library, 2013.

[2] A. C. Bovik. *Handbook of image and video processing*. Academic press, 2010.

[3] M. Buerli and S. Misslinger. Introducing arkit-augmented reality for ios. In *Apple Worldwide Developers Conference (WWDC'17)*, pages 1–187, 2017.

[4] P. Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, New York, NY, USA, 1998. ACM.

[5] P. Debevec, P. Graham, J. Busch, and M. Bolas. A single-shot light probe. In *ACM SIGGRAPH 2012 Talks*, page 10. ACM, 2012.

[6] V. Developer. Sdk, unity extension vuforia–7.1 (2018).

[7] Epic. Unreal engine. *Online[Cited: October 1, 2018.]: https://www.unrealengine.com*, 2018.

[8] Y. Feng. Estimation of light source environment for illumination consistency of augmented reality. In *2008 Congress on Image and Signal Processing*, volume 3, pages 771–775, May 2008.

[9] M.-A. Gardner, K. Sunkavalli, E. Yumer, X. Shen, E. Gambaretto, C. Gagné, and J.-F. Lalonde. Learning to predict indoor illumination from a single image. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 9(4), 2017.

[10] Google. Arcore. *Online[Cited: October 1, 2018.]: https://developers.google.com/ar/discover/*, 2018.

[11] T. Grosch, T. Eble, and S. Mueller. Consistent interactive augmentation of live camera images with correct near-field illumination. In *Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, pages 125–132. ACM, 2007.

[12] L. Gruber, T. Langlotz, P. Sen, T. Höherer, and D. Schmalstieg. Efficient and robust radiance transfer for probeless photorealistic augmented reality. In *2014 IEEE Virtual Reality (VR)*, pages 15–20, March 2014.

[13] V. Havran, M. Smyk, G. Krawczyk, K. Myszkowski, and H.-P. Seidel. Interactive system for dynamic scene lighting using captured video environment maps. In *Rendering Techniques*, pages 31–42, 2005.

[14] J. T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM.

[15] K. Karsch, V. Hedau, D. Forsyth, and D. Hoiem. Rendering synthetic objects into legacy photographs. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, pages 157:1–157:12, New York, NY, USA, 2011. ACM.

[16] K. Karsch, K. Sunkavalli, S. Hadap, N. Carr, H. Jin, R. Fonte, M. Sittig, and D. Forsyth. Automatic scene inference for 3d object compositing. *ACM Trans. Graph.*, 33(3):32:1–32:15, June 2014.

[17] E. A. Khan, E. Reinhard, R. W. Fleming, and H. H. Bülthoff. Image-based material editing. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 654–663, New York, NY, USA, 2006. ACM.

[18] N. Kholgade, T. Simon, A. Efros, and Y. Sheikh. 3d object manipulation in a single photograph using stock 3d models. *ACM Trans. Graph.*, 33(4):127:1–127:12, July 2014.

[19] J. Lalonde, A. A. Efros, and S. G. Narasimhan. Estimating natural illumination from a single outdoor image. In *2009 IEEE 12th International Conference on Computer Vision*, pages 183–190, Sept 2009.

[20] J.-F. Lalonde, A. A. Efros, and S. G. Narasimhan. Webcam clip art: Appearance and illuminant transfer from time-lapse sequences. In *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, pages 131:1–131:10, New York, NY, USA, 2009. ACM.

[21] D. Mandl, K. M. Yi, P. Mohr, P. M. Roth, P. Fua, V. Lepetit, D. Schmalstieg, and D. Kalkofen. Learning lightprobes for mixed reality illumination. In *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 82–89, Oct 2017.

[22] K. Nishino and S. K. Nayar. Eyes for relighting. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 704–711, New York, NY, USA, 2004. ACM.

[23] D. Nowrouzezahrai, S. Geiger, K. Mitchell, R. Sumner, W. Jarosz, and M. Gross. Light factorization for mixed-frequency shadows in augmented reality. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 173–179, Oct 2011.

[24] NVIDIA. Nvidia physx library. *Online[Cited: October 1, 2018.]: http://www.nvidia.com/object/physx-9.17.0524-driver.html*.

[25] M. Protter, M. Kushnir, and F. Goldberg. Method and a system for identifying reflective surfaces in a scene, Apr. 6, 2017. US Patent App. 14/872,160.

[26] R. Ramamoorthi and P. Hanrahan. An efficient representation for irradiance environment maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 497–500. ACM, 2001.

[27] T. Richter-Trummer, D. Kalkofen, J. Park, and D. Schmalstieg. Instant mixed reality lighting from casual scanning. In *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 27–36, Sept 2016.

[28] K. Rohmer, W. Büschel, R. Dachselt, and T. Grosch. Interactive near-field illumination for photorealistic augmented reality on mobile devices. In *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 29–38, Sept 2014.

[29] I. Sato, Y. Sato, and K. Ikeuchi. Acquiring a radiance distribution to superimpose virtual objects onto a real scene. *IEEE transactions on visualization and computer graphics*, 5(1):1–12, 1999.

[30] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM National Conference*, ACM '68, pages 517–524, New York, NY, USA, 1968. ACM.

[31] SolidAngle. Arnold renderer, 2014.

[32] J. Unger, S. Gustavson, P. Larsson, and A. Ynnerman. Free form incident light fields. In *Computer Graphics Forum*, volume 27, pages 1293–1301. Wiley Online Library, 2008.

[33] J. Unger, J. Kronander, P. Larsson, S. Gustavson, and A. Ynnerman. Temporally and spatially varying image based lighting using hdr-video. In *Signal Processing Conference (EUSIPCO), 2013 Proceedings of the 21st European*, pages 1–5. IEEE, 2013.

[34] J. Unger, A. Wenger, T. Hawkins, A. Gardner, and P. Debevec. Capturing and rendering with incident light fields. Technical report, University of Southern California Marina Del Rey CA, Inst. for Creative Technologies, 2003.

[35] UnityEngine. Unity game engine-official site. *Online[Cited: October 1, 2018.] http://unity3d.com*, pages 1534–4320, 2018.

[36] Y. Wang and D. Samaras. Estimation of multiple directional light sources for synthesis of augmented reality images. *Graphical Models*, 65:185 – 205, 2003. Special Issue on Pacific Graphics 2002.

[37] G. J. Ward. The radiance lighting simulation and rendering system. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 459–472, New York, NY, USA, 1994. ACM.

[38] T. Whelan, M. Goesele, S. J. Lovegrove, J. Straub, S. Green, R. Szeliski, S. Butterfield, S. Verma, and R. Newcombe. Reconstructing scenes with mirror and glass surfaces. *ACM Transactions on Graphics (TOG)*, 37(4):102, 2018.

[39] G. Xing, X. Zhou, Q. Peng, Y. Liu, and X. Qin. Lighting simulation of augmented outdoor scene based on a legacy photograph. In *Computer Graphics Forum*, volume 32, pages 101–110. Wiley Online Library, 2013.

[40] E. Zhang, M. F. Cohen, and B. Curless. Emptying, refurnishing, and relighting indoor spaces. *ACM Trans. Graph.*, 35(6):174:1–174:14, Nov. 2016.