LitAR: Visually Coherent Lighting for Mobile Augmented Reality

YIQIN ZHAO, Worcester Polytechnic Institute, USA
CHONGYANG MA, Kuaishou Technology, China
HAIBIN HUANG, Kuaishou Technology, China
TIAN GUO, Worcester Polytechnic Institute, USA

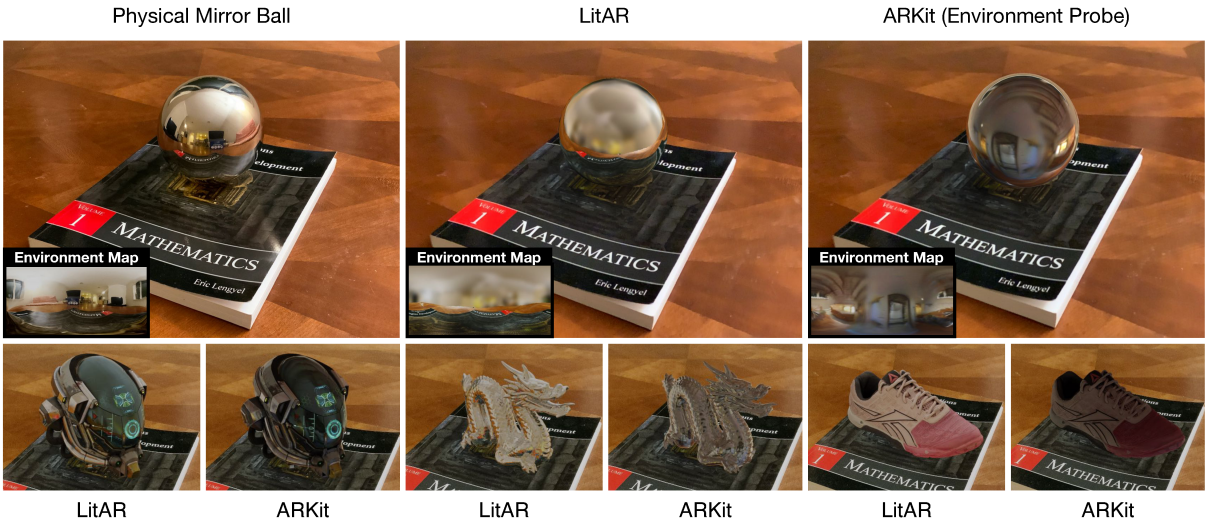


Fig. 1. Rendered objects with LitAR compared to a physical mirror ball and with ARKit [34]. *Row 1:* LitAR produces visually-coherent reflective rendering while ARKit fails; the bottom part of the ARKit-rendered ball does not faithfully reflect the book cover. The top part of the LitAR-rendered ball has an intentionally gradient blurring effect for quality-performance trade-offs (see §4.2). *Row 2:* LitAR achieves more realistic and visually coherent rendering effects than ARKit for objects with different materials.

An accurate understanding of omnidirectional environment lighting is crucial for high-quality virtual object rendering in mobile augmented reality (AR). In particular, to support reflective rendering, existing methods have leveraged deep learning models to estimate or have used physical light probes to capture physical lighting, typically represented in the form of an environment map. However, these methods often fail to provide visually coherent details or require additional setups. For

Authors' addresses: Yiqin Zhao, Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA, USA, yzhao11@wpi.edu; Chongyang Ma, Kuaishou Technology, 6 Shangdi West Road Haidian, Beijing, Beijing, China, chongyangm@gmail.com; Haibin Huang, Kuaishou Technology, 6 Shangdi West Road Haidian, Beijing, Beijing, China, jackiehuanghaibin@gmail.com; Tian Guo, Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA, USA, tian@wpi.edu.

2474-9567/2022/9-ART153 $15.00
https://doi.org/10.1145/3550291

example, the commercial framework ARKit uses a convolutional neural network that can generate realistic environment maps; however the corresponding reflective rendering might not match the physical environments. In this work, we present the design and implementation of a lighting reconstruction framework called LitAR that enables realistic and visually-coherent rendering. LitAR addresses several challenges of supporting lighting information for mobile AR.

First, to address the spatial variance problem, LitAR uses two-field lighting reconstruction to divide the lighting reconstruction task into the spatial variance-aware near-field reconstruction and the directional-aware far-field reconstruction. The corresponding environment map allows reflective rendering with correct color tones. Second, LitAR uses two noise-tolerant data capturing policies to ensure data quality, namely guided bootstrapped movement and motion-based automatic capturing. Third, to handle the mismatch between the mobile computation capability and the high computation requirement of lighting reconstruction, LitAR employs two novel real-time environment map rendering techniques called multi-resolution projection and anchor extrapolation. These two techniques effectively remove the need of time-consuming mesh reconstruction while maintaining visual quality. Lastly, LitAR provides several knobs to facilitate mobile AR application developers making quality and performance trade-offs in lighting reconstruction. We evaluated the performance of LitAR using a small-scale testbed experiment and a controlled simulation. Our testbed-based evaluation shows that LitAR achieves more visually coherent rendering effects than ARKit. Our design of multi-resolution projection significantly reduces the time of point cloud projection from about 3 seconds to 14.6 milliseconds. Our simulation shows that LitAR, on average, achieves up to 44.1% higher PSNR value than a recent work Xihe on two complex objects with physically-based materials.

CCS Concepts: • **Computing methodologies → Mixed / augmented reality**; • **Human-centered computing → Ubiquitous and mobile computing systems and tools**; • **Computer systems organization → Distributed architectures**.

Additional Key Words and Phrases: mobile augmented reality; lighting estimation; 3D vision

ACM Reference Format:
Yiqin Zhao, Chongyang Ma, Haibin Huang, and Tian Guo. 2022. LitAR: Visually Coherent Lighting for Mobile Augmented Reality. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 6, 3, Article 153 (September 2022), 29 pages. https://doi.org/10.1145/3550291

1 INTRODUCTION

Mobile augmented reality (AR) has attracted increasing interest from academia and industry to better engage users by allowing seamless integration of physical and virtual environments [3, 27, 35]. The current mobile AR ecosystem is infused with new hardware development [32], improved frameworks [34], advancing vision and graphics algorithms [64], as well as end-user facing applications ranging from e-commerce ones to educational ones [2].

Given the interactive nature of AR applications, users often prefer virtual objects of high visual quality. The rendered virtual objects should look realistic and feel like they belong to the physical surroundings, a property commonly referred to as *visual coherence*. For example, virtual sunglasses that are overlaid on a user's face should look like physical sunglasses (realistic) and reflect the correct physical environment (visually coherent). To achieve both realistic and visually coherent rendering, mobile AR applications require access to an accurate representation of omnidirectional environment lighting (often represented as *environment map* for image-based lighting) at the user-specified rendering position [47, 64].

However, obtaining a high-quality environment map for mobile AR has to overcome several key challenges. First, the inherently spatial variation of indoor environment lighting makes the environment map at the *observation position*—which can be more easily reconstructed with more camera observations—a poor approximation of the *rendering position* environment map. This challenge was demonstrated in prior work [24, 64] and our motivating example (see Figure 3). Second, the natural user mobility of mobile AR usage can induce noise to necessary data (such as 6DoF tracking and RGB image data) for lighting estimation. For example, we observe that the tracking data provided by ARKit can show that consecutive camera frames are misaligned, although they represent the same physical space. Third, mobile devices can have heterogeneous sensing capability, e.g., in terms of cameras'

field-of-view (FoV) or their depth-sensing ability, which makes it necessary to consider auxiliary components (such as depth estimation [1, 48, 63]) for obtaining accurate lighting. Fourth, mobile devices have relatively limited resources compared to their desktop counterpart, while the interactivity nature often requires 30 fps rendering. The computational limitation makes it challenging to directly use computational-intensive models designed to run on powerful GPU servers [52], and motivates minimal usage or optimization of time-consuming operations (such as point cloud registration [9] and mesh reconstruction [8]).

In this work, we investigate the problem of *providing high-quality lighting information for mobile AR* by addressing the above four key challenges. Our key goal is to support realistic and visually coherent rendering of virtual objects with various geometries and materials. Figure 1 compares the visual effect of virtual objects rendered with lighting information obtained with our proposed system called LitAR and ARKit. We show that LitAR achieves high-quality rendering with structurally similar reflections with the physical object and more visually coherent reflection than objects rendered with ARKit.

LitAR involves a novel technique called *two-field lighting reconstruction* and several complementary components that work together to deliver a high-quality environment map with low-performance impact. We design the two-field lighting reconstruction with the insight of dividing the camera observations into two types, i.e., the near-field and far-field observations, to speed up lighting reconstruction while maintaining the visual quality. This technique shares a similar spirit to the well-known screen space reflection [42] and is tailored to mobile AR by fully exploring user mobility. Specifically, LitAR generates a multi-view dense point cloud to represent *near-field* observations, corresponding to the portion of the environment map that receives more accurate and higher confidence depth information surrounding the rendering position. This design helps produce geometrically accurate lighting transformation between the observation and rendering positions, thus supporting key rendering features like reflections and providing visually coherent results. Furthermore, LitAR leverages far-field observations to handle the anisotropic lighting property by reconstructing sparse point clouds to reduce visual errors.

On top of the two-field lighting reconstruction, we incorporate two noise-tolerant data capturing policies, i.e., guided bootstrapped movement and motion-based automatic capturing, to improve the data quality. The *guided bootstrapped movement* policy directs the camera views to capture required near-field and far-field observations efficiently. This policy also brings other benefits, such as enlarged FoVs and reduced user movement, for reconstructing high-quality lighting. It is worth noting that LitAR can leverage new observations, e.g., device orientation and user movement, to improve the quality of the environment map progressively. The *motion-based automatic capturing* policy leverages multi-sensory information to capture spatially and temporally new observations. Moreover, we propose two performance optimizations that significantly reduce the time reconstructing the final environment map from the intermediate 3D point clouds. The first optimization uses a lightweight multi-resolution projection instead of the traditional expensive mesh reconstruction to generate the near-field portion. The second optimization uses a unit sphere-based approach called *anchor extrapolation* to generate gradient coloring and blurring effect of the far-field portion. Lastly, LitAR supports reconstruction quality and time trade-offs to account for dynamic lighting conditions. By default, LitAR provides three quality presets for mobile AR developers.

We implement LitAR as an edge-assisted framework that consists of about 2.2K lines of code running on both the mobile device and the edge server. Specifically, the client-side component works with various sensors, including color, depth, and motion sensors, to capture near-field and far-field observations. The resulting data is encoded and sent to the edge server to generate a fixed-size unit sphere point cloud and multi-view dense point clouds with good alignment and visual pixel continuity. Mobile AR applications built with Unity ARFoundation can directly use LitAR to render realistic virtual objects. We also implement a reference iOS AR application for our testbed-based evaluation. To evaluate LitAR in a controlled environment, we develop a simulator based on Unreal Engine that exposes multiple knobs for controlling physical factors such as camera location and FoV while

providing ground truth lighting information. The testbed-based system evaluation shows that LitAR achieves more visually coherent rendering results and higher PSNR/SSIM values than ARKit for three real-world scenes. The end-to-end latency measurements show that LitAR can generate about 22 environment maps per second, effectively supporting 22 fps which is sufficient for most mobile AR applications [61, 62]. Our simulator-based evaluations include one realistic indoor scene and six virtual objects of different shapes and materials. We evaluate the performance of LitAR with various observation-rendering position pairs. We show that it achieves 36.7% and 17.1% higher rendering PSNR compared to a recent deep learning-based lighting approach [65] and the environment lighting captured by 360° cameras at the observation position, respectively.

Related work on generating spatially-varying lighting includes classical physical probe-based techniques [15, 16, 47] and learning-based solutions [24, 51, 52, 64]. Physical probe-based techniques often produce high-quality environment maps but have more constrained usage scenarios since they require additional setup [51]. On the other hand, the applicability of learning-based solutions is often limited by the access to extensive training datasets, e.g., Matterport3D [11], and their suitability to run on heterogeneous mobile devices [52]. Another side effect is the difficulty of conducting comprehensive comparisons due to the lack of publicly available source code and benchmark dataset [51]. In this work, we are interested in designing a mobile-specific lighting framework that circumvents the above-mentioned limitations by considering mobile characteristics from the outset. Compared to a recent method by Somanath et al. [51] that generates HDR environment maps using a neural network based on adversarial training, LitAR has the advantage of simplicity yet achieving good visual coherence.

In summary, we make the following key contributions:

- We design a novel technique called two-field lighting reconstruction, which generates high-quality environment maps from mobile cameras with limited FoV. Each environment map consists of near-field and far-field portions, separately constructed from near-field and far-field observations. The resulting environment map captures spatial and directional variances and is suitable for reflective rendering.
- We develop several complementary approaches to handle mobility-induced noise, limited mobile sensing capabilities, and the computation intensity that naturally arises during the lighting reconstruction process. For example, our multi-resolution projection and anchor extrapolation techniques efficiently project the intermediate 3D point clouds to the final 2D environment maps. These techniques ensure high data input quality, good usability, and low reconstruction time.
- We implement the entire framework as an edge-assisted system called LitAR and develop a simulator based on Unreal Engine for evaluation purposes. The system implementation provides a platform to compare LitAR to the commercial framework ARKit. The simulator facilitates controlled experiments and allows easy comparisons between lighting techniques and ground truth lighting. Our source code and related artifacts are available at https://github.com/cake-lab/LitAR to encourage follow-up research.
- We evaluate the performance of LitAR on a small-scale testbed using the simulator. The testbed-based system evaluation shows that LitAR (at all three quality presets) outperforms ARKit in three real-world indoor scenes. LitAR also delivers environment maps at 22 fps or even higher, depending on the quality settings. The simulation-based evaluation shows that LitAR can achieve up to 36.7% higher PSNR values on objects with various geometries and materials than a recent lighting framework [65].

2 BACKGROUND: LIGHTING FOR MOBILE AR

Obtaining lighting information is a classic problem in computer vision and computer graphics [20]. Access to accurate environment lighting information is crucial to many applications related to photorealistic rendering and image manipulation, such as 3D object composition [23] and portrait relighting [46].

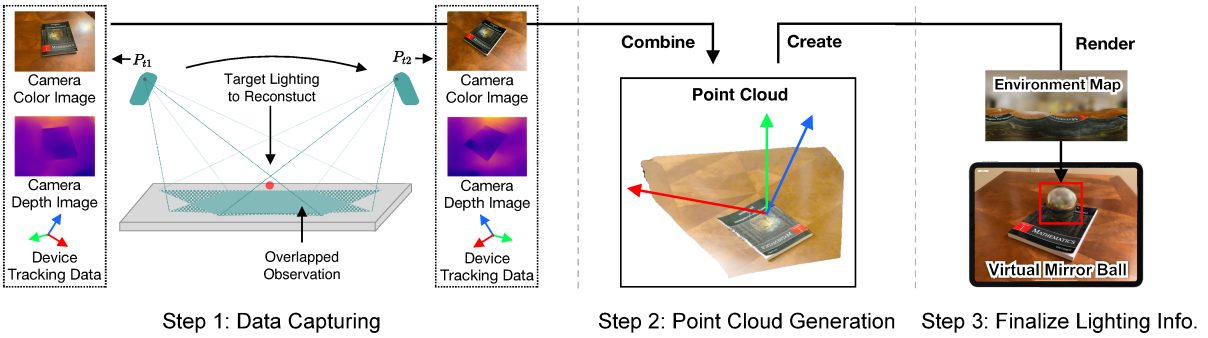


Fig. 2. A simplified workflow of mobile AR lighting reconstruction. In this example, the mobile device starts at position P_{t1} and points toward the rendering position on the table specified by the mobile user. Data such as color and depth images, as well as device tracking data, will be captured. The device will further capture more data (e.g., color/depth images from a different viewing direction) at position P_{t2}, which naturally overlaps with the data captured at P_{t1}. The data is maintained in a fixed-size buffer and will be aggregated to generate a global multi-view point cloud. We will update the point cloud as new data arrives and use it to generate the final environment map for virtual object rendering.

As the capability of mobile devices improves and AR re-emerges in user-facing applications [2, 35], obtaining environment lighting for photorealistic rendering has gained increasing interests in various research communities [12, 47, 51, 64, 65]. In addition to these mobile-specific lighting techniques, researchers have investigated image-based lighting [14, 15, 36], generating environment lighting representations from camera videos [28, 30, 58], and assisted lighting reconstruction with physical probes [16], object cues [54], or scene geometry [6, 41]. Recent work is mostly deep learning-based and can be broadly categorized based on the output: estimating low-frequency lighting [24, 65] or high-quality lighting [22, 52].

This paper aims to provide lighting support for mobile augmented reality (AR), an emerging application that augments the real world by overlaying with virtual objects in indoor scenes. Our work will develop a non-learning-based approach by leveraging *lighting reconstruction* (§2.1) to address the following two issues: *(i)* the typical limitations of deep learning based methods such as training data availability and inference performance on heterogeneous mobile resources; *(ii)* the underexploited mobile characteristics. Note that techniques commonly used for creating realistic virtual worlds, such are screen space rendering [42] that only ray traces what is being presented on the screen, can fall short in delivering the visual coherence required by AR. This shortcoming is mainly due to the key difference in perceivable lighting impact on virtual objects; unlike in fully immersive environments, AR users can observe environment lighting outside screen space and potentially perceive incoherent visual artifacts.

Furthermore, we will focus on developing lighting understanding techniques for image-based rendering [15], e.g., representing lighting in the form of an *environment map* to render virtual objects with different materials. Our key goal is to generate high-quality environment maps for visually coherent rendering for mobile AR while keeping the time cost low. Specifically, we aim to reduce the overall time to obtain the final environment map and the component-wise time for intermediate outputs (such as point cloud projection).

2.1 Lighting Reconstruction Primer

In this section, we present the critical information of *multi-view lighting reconstruction* (refer to Figure 2), which serves as the basis for understanding our proposed lighting reconstruction framework LITAR. At a high level, we

define *lighting reconstruction* as a task similar to multi-view 3D reconstruction [10, 43] with the key difference in the reconstruction target. Our key insight is that by leveraging multiple captures of the physical environment that are often required by commercial AR frameworks [25, 34], we can use typical techniques used by 3D reconstruction to understand the environment lighting. Conceptually simple, we have to address several challenges specific to the mobile AR environment (detailed in §3). Next, we describe the general procedure for the lighting reconstruction task.

2.1.1 Step 1: Capture Environment Data. Several different types of data, e.g., color images and depth information, are needed in lighting reconstruction. The common way to obtain these required data is to leverage a modern mobile device with a reasonable camera directly and to have the mobile AR users move the mobile device to scan the surroundings manually. The resulting captured data is often in the format of LDR or HDR images, which can then be used to reconstruct the environment's appearance and geometry. To improve the reconstruction quality and performance, one can also resort to additional setups such as using a physical chrome ball [15, 47] or additional mobile sensors such as depth sensors [31, 32] and accelerometers. Ambient light sensors can also be used to observe the ambient color, which helps match the object's color tone with the environment's lighting. In this work, we focus on mobile devices that can capture color and depth images and provide device tracking data, e.g., a LiDAR-equipped iPad Pro. Data will be captured from different viewing positions and used in the next step for generating a multi-view point cloud.

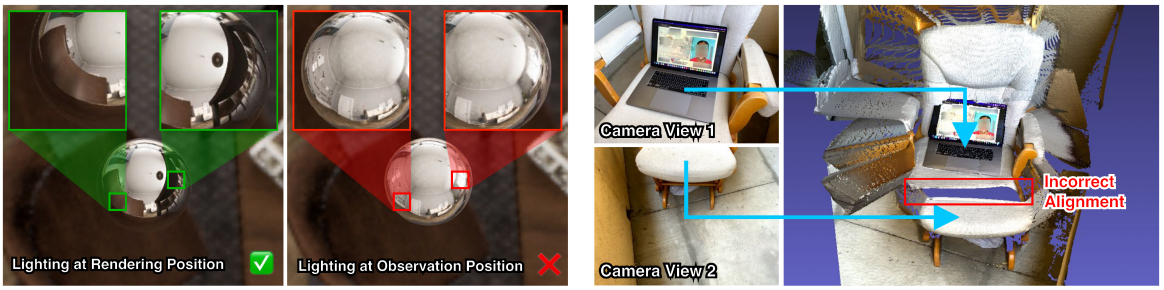
2.1.2 Step 2: Generate a Point Cloud. Similar to other 3D reconstruction tasks [10, 43], we convert the camera color, depth images, and device tracking data, into a point cloud-based representation in the world space. The point cloud data structure allows us to combine the subsequent view data more efficiently than directly stitching 2D images. Two practical issues often need to be addressed. First, real-world device tracking data can be noisy; one way to handle this issue is to use point cloud registration techniques such as the iterative closest point registration [9] to align the points. Second, some points might not have accurate depth information; to ensure the reconstruction quality, only points with high depth confidence values, which measure the accuracy of depth data, should be used. Note that we will update the point cloud based on newer data; conceptually, such an update helps deal with both spatial and temporal variance by initializing/overriding points in the 3D space at different times.

2.1.3 Step 3: Finalize Environment Lighting. The generated multi-view point cloud consists of rich environment information and is equivalent to having an enlarged virtual camera FoV at the rendering position. It is worth noting that enlarging the camera FoV at the observation position can also increase the camera observation coverage, though less effective than multi-view enlargement. To directly use modern rendering engines to support realistic rendering, we convert the point cloud to lighting formats, such as spherical image format or environment map. For example, one can project the point cloud into a 2D environment map that captures the omnidirectional environment lighting.

3 MOTIVATION AND CHALLENGES

3.1 Spatial and Temporal Variance

Indoor lighting can be both spatially and temporally varying [24, 53, 64]. Rendering virtual objects using lighting information from locations other than the *rendering position* may lead to potential visual degradation. To demonstrate the impact of such variances on the rendering effect, we compare a virtual object rendered with the lighting information at the rendering position and at the *observation position* in Figure 3a. We can see that the mirror ball on the right does not have the desired visual appearance, i.e., neither of the zoomed-in views contains the correct reflections of the chair and the table. Thus, it is crucial to account for spatial variance when designing the reconstruction framework. Intuitively, lighting can change temporally even at the same rendering position. In this work, we handle the temporal variance by periodically reconstructing lighting.



(a) Spatial lighting variance

(b) Mobility-induced noise

Fig. 3. Motivating examples. (a) The same mirror ball object rendered with environment maps at the AR mobile device's position and the mirror ball placement position can have very different visual appearances. (b) The mobility-induced tracking noise leads to a misaligned point cloud generated from two consecutive frames.

3.2 Mobility-induced Noise

The inherent user mobility can naturally introduce noise to data required by the lighting reconstruction pipeline. For example, when a mobile user engages with an AR application, the user might move around, introducing measurement errors into the 6DoF tracking data or leading to blurred RGB images. Other factors, such as the mobile camera's position relative to the rendering position, can also impact the quality of camera observations. Both tracking and camera data are commonly used for lighting estimation [64, 65]. Figure 3b shows that the point cloud can have incorrect alignment when naively using two consecutive color and depth images. We note that misalignment is a common error in current AR applications that uses ARKit's world tracking data [18].

3.3 Limited Sensing Capability

Mobile vision sensors have become more potent over the past few years, especially with the newly equipped depth-sensing capability. However, most commercial mobile cameras still have limited FoVs ($< 120°$) [5, 26], much less than the desired $360°$ cameras commonly used by the movie industry to reconstruct lighting information [38]. Furthermore, many modern phones still do not have access to depth sensors and rely on different algorithms to estimate depth [19, 63]. Depth estimation errors can significantly impact the 3D point cloud generation process. In short, the limited mobile sensing capability can make capturing high-quality data for lighting reconstruction challenging.

3.4 Resource-quality Trade-offs

The task of generating high-quality environment lighting information can be very compute-intensive. State-of-the-art lighting models, which support visually coherent reflection, often require running on a powerful GPU server to achieve reasonable performance [52]. With an additional setup of physical light probes, for example, GLEAM can take from 30ms to 400ms to update scene lighting estimation [47]. By sacrificing the lighting quality, Zhao et al. achieved real-time lighting estimation (20.1ms) with low-fidelity spherical harmonics coefficients [65]; Somanath et al. trained a deep learning model that can generate an HDR environment map in less than 9ms on recent iPhones but severely sacrifices visual coherence [51]. It is challenging to navigate the resource-quality trade-offs in providing visually coherent lighting.

4 LITAR DESIGN

4.1 Overview

We design LitAR to address the challenges mentioned above to reconstruct high-quality environment lighting information. LitAR is an adaptive framework that progressively, e.g., as AR users naturally move around the indoor environment, reconstructs environment lighting for any user-specified reconstruction positions. In contrast to prior estimation-based work [22, 39, 52, 64], the core of LitAR lies in how to effectively *reconstruct* environment lighting information from a sequence of limited camera observations. Our reconstruction-based approach promises to obtain more accurate lighting information and achieves better visual results without requiring expensive data collection, model training, or physical setup [47].

Specifically, LitAR proposes a novel *two-field lighting reconstruction* technique (§4.2) to produce geometrically accurate transformations to handle the challenge of spatial variance by transforming indirect scene observations to the desired lighting information. Figure 4 presents an overview of LitAR. To mitigate the impact of mobility-induced noise on the reconstruction quality, LitAR proposes two policies for guiding bootstrapped device movement (§4.3.1) and automatically capturing required data based on motion (§4.3.2). To account for limited mobile sensing capability, LitAR only requires depth information on some camera observations (i.e., *near-field observations* that have the reconstruction position in the view) and applies point cloud registration to correct the device tracking errors (§4.5.2). The resource intensity is dealt with from the outset with a mobile-centered design. Specifically, LitAR divides camera observations and has them go through two separate execution branches to a multi-view dense point cloud and a fixed-size point cloud. This two-branch design effectively reduces the computational cost and memory consumption of LitAR. We propose two novel performance optimization techniques, i.e., multi-resolution projection (§4.4) and anchor extrapolation (§4.4.2), to render environment maps in real time at the edge. While many knobs can impact the quality and efficiency of lighting reconstruction, LitAR allows mobile AR developers to make such trade-offs via a configurable design (§4.5.1).

4.2 Two-Field Lighting Reconstruction

At the high level, our two-field lighting reconstruction technique divides the task of lighting reconstruction into two sub-tasks: one that leverages depth information to produce high-quality lighting from near-field observations and one lightweight task for reconstructing lighting from far-field observations. In other words, LitAR will generate two intermediate point clouds from camera observations for rendering environment maps. A *multi-view dense point cloud* is the outcome of judiciously applying the geometrically accurate transformation and dense sampling on near-field observations; A *unit-sphere point cloud* is the sampling outcome of the sparse point clouds from the far-field observations and the dense point clouds. Recall that we divide the camera observations into two types, *near-field observation* that includes the reconstruction position in the view and *far-field observation* that does not. As explained below, such division is based on the key insight that camera observations are subject to varying levels of spatial variance.

Figure 5 illustrates the different importance of considering spatial variance, depending on the relative position of the interested pixel to the observation and reconstruction positions. Assume a position P_{env} in the physical environment. To render P_{env} on a virtual object surface, P_{env} should be observable from the reconstruction position P_{rec}. To perceive any position P_{env} in the environment, one has to observe light emitted/reflected from P_{env}. Without loss of generality, in Figure 5a, we show the intersection l_{rec} of vector $\langle P_{env}, P_{rec}\rangle$ and the surface of the unisphere (with P_{rec} being the center) represents the desired reflection. However, if we directly reconstruct the light ray from the camera observation position P_{obs}, we end up with l_{obs}, the intersection between vector $\langle P_{env}, P_{obs}\rangle$ and the surface of the P_{obs}-centered unisphere. If we translate l_{obs} to the P_{rec}-centered unisphere by applying the vector $\langle P_{obs}, P_{rec}\rangle$, we will get a third intersection l'_{obs}. Observe that the light ray represented by $\langle P_{rec}, l'_{obs}\rangle$ can differ significantly (i.e., larger $\Delta\alpha$) from $\langle P_{rec}, l_{rec}\rangle$, the light ray that should be perceived at

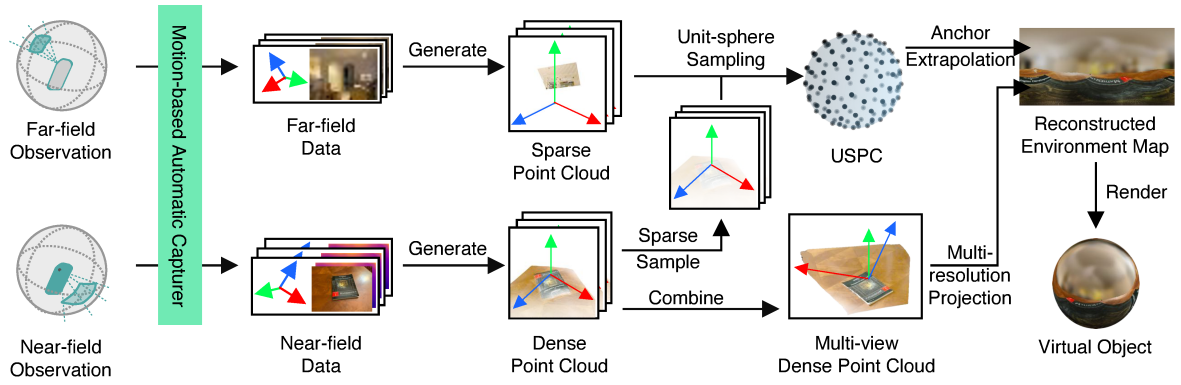


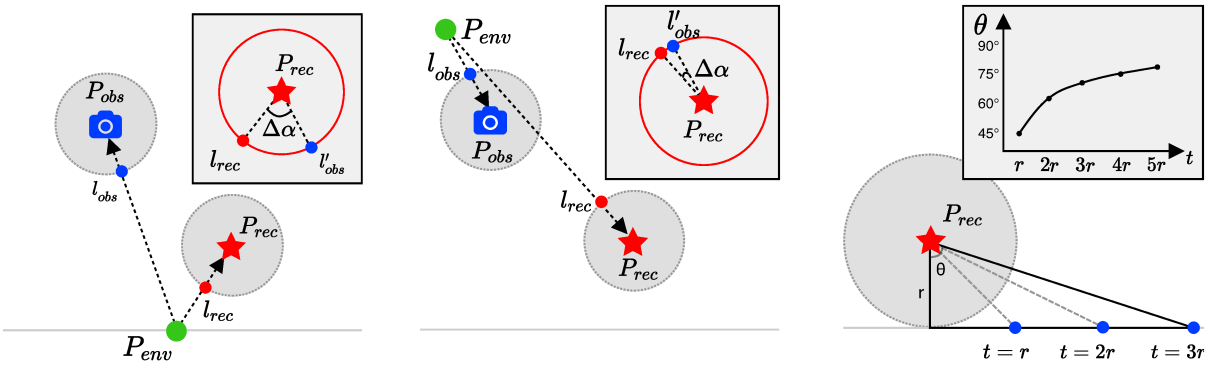
Fig. 4. LITAR's overview. LITAR separately reconstruct for the near-field (§4.2.1) and far-field (§4.2.2) regions from camera observations strategically guided and automatically captured. The near-field data, which has a higher impact on spatial variance and consists of more accurate depth information, will be used to generate a multi-view dense point cloud. The far-field data will be projected to color a unit-sphere point cloud (USPC) with the sampling technique from [65], and then will be extrapolated to fill neighboring pixels in the environment map. Finally, the dense point cloud will be projected to multi-resolution environment maps which will be combined, using our *multi-resolution projection* technique (§4.4.1).

P_{rec}. In short, near-field observations are impacted much more by spatial variance. On the contrary, as shown in Figure 5b, if the P_{env} is in the far-field, l'_{obs} can be a good approximation for l_{rec} (i.e., much smaller $\Delta\alpha$). In short, far-field observations are impacted much less by spatial variance.

Other benefits of separating camera observations into near-field and far-field include better tolerance of limited mobile depth-sensing capability and resource efficiency. A naive alternative design of applying the same reconstruction pipeline to all camera observations can lead to incorrect point clouds and demand resources proportional to the indoor scene space (i.e., the total number of points). In contrast, our design of processing far-field observations demands less memory and computation resources by using a fixed-size point cloud [65].

4.2.1 Spatial Variance-Aware Near-Field Reconstruction. To effectively transform camera observation(s) to the environment map at the reconstruction position, LITAR leverages the increasingly popular depth sensor in mobile camera system [31, 32]. Depth sensors enable the possibility of capturing geometrically accurate environment observations. First, we densely sample color and depth buffers of the camera images for each near-field observation. Then, the image buffers and the camera transformation matrix (i.e., rotation and translation) are used to generate the geometry and color of a dense 3D point cloud. In our implementation, we use the camera transformation matrix information provided by commercial AR frameworks. Such information is often referred to as device tracking data. The point cloud generation process can be time-consuming. To speed up this process, we separate the geometry and color generation tasks and then execute both tasks on the GPU. Finally, the output dense point cloud (i.e., geometry and color information) is written to a global point cloud buffer that maintains a multi-view point cloud for the current reconstruction session.

To support multi-view reconstruction, LITAR uses a motion-based automatic capturing scheme (§4.3.2) to supply the reconstruction pipeline with new near-field observation. LITAR assigns a unique indexing identifier for each near-field observation. This identifier is subsequently used for other data, including the camera transformation



(a) Spatial variance in near-field (b) Spatial variance in far-field (c) Near-field boundary size

Fig. 5. Illustration of spatial variance impact. We visualize the near-field and far-field observations in a 2D front view. P_{obs} and P_{rec} represent observation position and reconstruction position, respectively. (a) & (b) It is more important to consider spatial variance for pixels observed in near-field than ones in far-fields, i.e., larger $\Delta\alpha$. (c) Increasing the near-field boundary size (e.g., from r to $2r$ vs. $2r$ to $3r$) has a diminishing impact on the reflection surface.

matrix and the derived dense point cloud. The global point cloud buffer is updated with the least-recently observed policy, i.e., the points associated with the oldest near-field observation will be replaced.

We define *near-field boundary* as a constrained cubic space that contains points belonging to the near-field observations that will undergo further processing. For points outside the near-field boundary, we will not perform multi-resolution projection (§4.4.1). Theoretically, this boundary can be as big as the indoor scene. However, having a too large boundary may have undesirable implications on both the memory and computation consumption and is often unnecessary. Figure 5c shows that increasing the near-field boundary size has a diminishing return on the reflection surface. For example, increasing the boundary from 2X to 3X of the virtual object size only increases the coverage by 8.13°, a 1.13X. In short, we use a configurable near-field boundary for a trade-off between resource consumption and reconstruction quality. We set the near-field boundary side length to two meters based on the AR virtual objects we use for testing. It is worth noting that AR developers should adjust this boundary for large virtual objects or divide the large object into smaller objects to have multiple reconstruction positions. The dense near-field point cloud generation allows us to produce geometrically accurate camera observation transformations and produce continuous lighting representations.

Our spatial variance-aware observation transformation still generates an approximated observation at the reconstruction position due to the lighting variance from different observation directions. We assume that the light does not change between different observation directions, which suffices for mobile AR rendering in most cases. However, such an assumption may lead to visually incorrect results if the environment around the reconstruction position contains reflective physical objects, e.g., mirrors. It is possible to further address such concerns by meticulously selecting light ray directions between observations, which we leave as future work.

4.2.2 Directional-Aware Far-Field Reconstruction. Mobile depth sensors usually only capture the surrounding environment within a limited range. For example, the LiDAR sensor on iPhone 13 Pro can capture depth up to 5 meters away [50]. Thus, it might not be suitable for sensing large physical environments. However, the desired environment lighting for AR rendering varies directionally depending on the surrounding physical

world environment. To address the directional variations, i.e., the anisotropy of environment lighting, LitAR reconstructs far-field lighting by sparsely sampling camera observation to provide omnidirectional lighting.

Even with the recent advancements in hardware, modern cameras still have small FoVs and thus capture only a small portion of the environment covering limited directions. However, reconstructing a dense point cloud to address the anisotropic lighting property from far-field observations is impractical as generating a dense point cloud for a far-field environment can be potentially unconstrained regarding computation and data storage. In addition, objects in the far-field observations may exceed the range limit of mobile depth sensors, which makes it difficult to obtain geometrically accurate transformation. The inherent nature of the far-field environment thus leads to lower confident far-field depth observations, which makes it ill-suited for dense point cloud reconstruction.

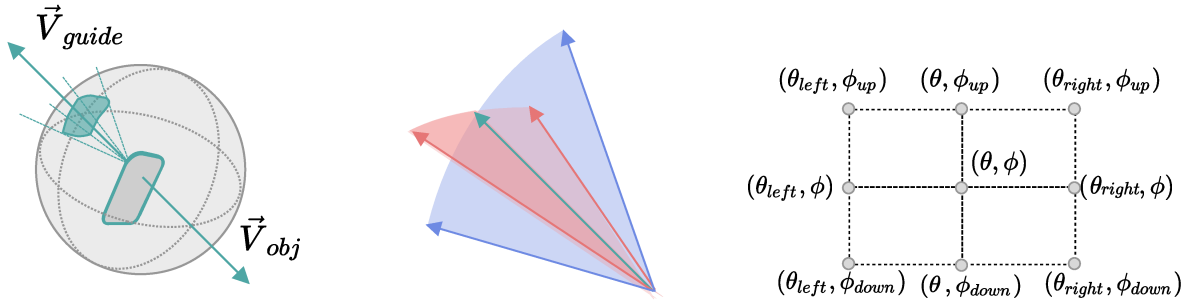
To address these limitations, we design a lightweight process to reconstruct far-field lighting from sparsely sampled camera images. Recall that a camera observation is considered a far-field observation if the reconstruction position falls outside the camera view. For a far-field observation, we sparsely sample a low-resolution camera image and obtain the current camera transformation matrix, similar to §4.2.1. Note that we do not capture depth data for the far-field observation as its depth information may be inaccurate, and the spatial variance has less impact.

To generate the sparse point cloud, we assume the depth of all pixels to be one and scale the camera intrinsic values accordingly. We use a similar design to [65] by projecting the sparse point cloud to a set of uniformly distributed points, referred to *anchors*, on a unit sphere. The resulting data structure is a *unit sphere-based point cloud* (USPC). As demonstrated in prior work [65], the design of USPC is aware of directional lighting variance and thus addresses the anisotropy property of environmental lighting. In this work, we set the number of anchors of the USPC to be 1280, the same as prior work [65]. The anchor points are colored by combining the color data from the sparse point cloud and ambient light sensor readings. Recall that we want USPC to represent the lighting from all directions, including near-field observations. Therefore, we sparsely sample the dense point cloud generated from the near-field observation; then, the resulting sparse point cloud is similarly projected to the same USPC. In short, the reconstructed far-field lighting is represented as a unit-sphere point cloud with a much smaller memory footprint (proportional to the anchor size) while still providing sufficient directional-aware lighting information.

4.3 Noise-tolerant Data Capturing Policies

4.3.1 Guided Bootstrapped Movement. Another key design of LitAR to reconstruct high-quality lighting is to exploit user movement, a feature of mobile AR. We observe that commercial mobile AR frameworks such as ARKit have built-in support for explicitly guiding mobile AR users to scan their physical surrounding environment before using the app. Note such practice is often used for calibrating world tracking data, but not for lighting estimation [4]. However, this commonly adopted movement practice typically leads to a biased sampling of environment lighting in concentrated observation directions. This biased sampling is due to the narrow focus on increasing the observation of the nearby environment around the reconstruction position. Although commercial frameworks use deep learning-based models to estimate environment lighting from observations, such biased observations create a barrier to more accurate estimation. As we will show in §6.2.3, increasing observations with the common practice shows little improvement in rendering results.

Instead, we propose a novel yet simple guided movement policy to look at the *backward environment*, i.e., observable from the opposite direction to the virtual object viewing direction. This guided movement is designed to increase the observation directions rather than the observation overlapping and to help address the anisotropic lighting property. In other words, our guided movement policy provides bootstrapped data at the AR application startup time to increase the far-field observations. As illustrated in Figure 6, our policy guides the user to look



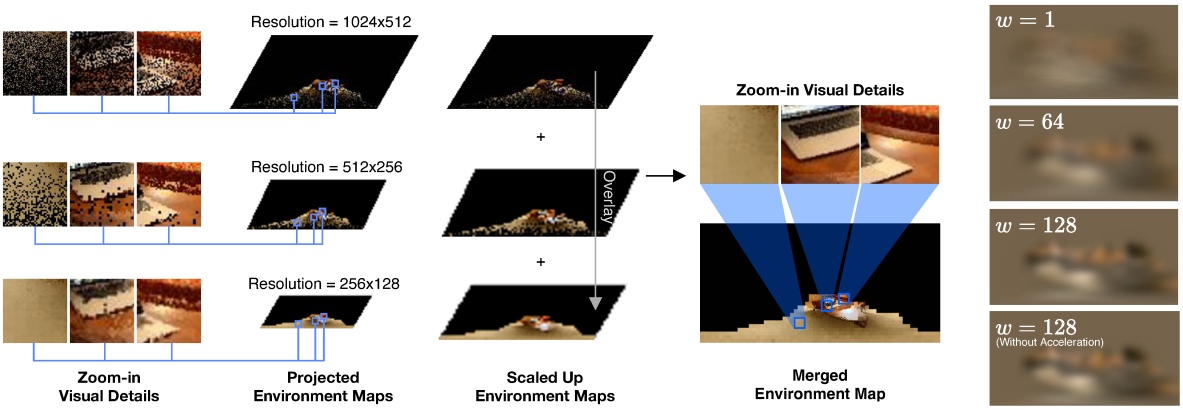
(a) Front & back facing observations (b) Horizontal & vertical movement (c) Movement direction grid

Fig. 6. Guided bootstrapped movement. Our guided bootstrapping movement technique directs users to point the mobile camera toward the opposite direction $\vec{V}_{guide}$ of the virtual object viewing direction $\vec{V}_{obj}$ and rotate along horizontal and vertical directions to increase the far-field observations. The movement follows a grid-style pattern, where (θ and ϕ) denote the observation direction in spherical coordinates.

at the backside of the intended virtual object observation direction. We use a grid-style pattern to partition the observation directions and provide guided viewing directions for the user. Specifically, as shown in Figure 6c, the optimal choices on the number of observations could be $\{1, 3, 5, 9\}$ based on the horizontal and vertical direction partitions. The number of observations corresponds to the combination of moving *left* and *right* in horizontal direction, as well as *up* and *down* in vertical direction. We currently use an angular difference of 30 degrees in horizontal and vertical directions. Furthermore, as our far-field reconstruction method does not reconstruct detailed observations, our guided movement can be performed without the user focusing on each camera observation orientation for a long time. In § 6.2.4, we will show that by using the guided movement, LitAR can find more accurate color tones to fill unseen areas and produce more accurate renderings.

4.3.2 Motion-based Automatic Capturing. Continuously capturing all camera observations or relying on the mobile AR user to manually capture them can lead to poor usability, low-quality data (e.g., images with motion blur), and high consumption of mobile resources. For example, prior work has demonstrated that motion blur is a common occurrence in mobile AR—which we also observe—and can lead to low accuracy for AR tasks [40]. Additionally, a recent low-frequency lighting estimation framework has demonstrated that strategically skipping the capture of specific camera frames has little impact on the estimation accuracy [65]. Intending to provide good usability, capture high-quality data, and reduce resource consumption, we design a *motion-based automatic capturing* technique that leverages multi-sensor data to automatically select camera frames and AR data for lighting reconstruction. In a nutshell, this technique will only capture observation data that is new spatially (i.e., by checking device position and rotation information) and temporally (i.e., by updating a previously captured frame with the same device information).

Specifically, LitAR uses a simple timer-based policy to assess the need to capture new data by checking if the mobile device has exhibited significant movement. In this work, we leverage the device accelerometer and gyroscope sensors to maintain a moving window of the device's most recent K position and rotation information (i.e., 6DoF). Every C milliseconds, LitAR will compare the device 6DoF information at the current frame to the ones in the moving window to assess the likelihood of motion blur. If the device pose has changed for more than 10cm and 10°, the device is considered to have significant movement in a short time window, and the current frame is skipped. LitAR will re-run the check every frame until a new frame is found while the device is relatively stable. Otherwise, the current frame is captured. The timer will be reset once a new frame is captured in both



(a) Multi-resolution projection (b) Anchor extrapolation

Fig. 7. Illustration of multi-resolution projection and anchor extrapolation. (a) We show an example of multi-resolution projection with three levels. The dense point cloud will be projected three times. The higher projection resolution leads to more visual details but has more discrete pixels; the lower projection resolution has more continuous pixels but fewer visual details and exhibits pixelation. We obtain a high-quality final environment map by scaling and overlaying each intermediate environment map. (b) We show that a larger w value leads to a less blurry environment map and that our nearest-anchor acceleration has minimal visual impact.

cases. Our motion-based automatic capturing will produce new data at least every C millisecond, depending on the mobility. Both the moving window and capturing frequency can be configured. In our implementation, we set $K = 5$ and $C = 300$.

4.4 Real-time Environment Map Rendering Techniques

Thus far, our two-field lighting reconstruction has generated two intermediate point clouds. To support high-quality multi-view lighting reconstruction for mobile AR, we need to convert the point cloud presentation into a lighting representation commonly supported by modern rendering engines. In this work, we choose *environment map* as the final lighting representation, which most mobile rendering engines can directly use. At the high level, to generate a high-quality environment map (i.e., visually continuous pixels) from a point cloud that consists of discrete points, one often needs to handle occlusion and inter-point connection. One common way to recreate the inter-point connections and calculate occlusion is to resort to conventional 3D reconstruction methods, e.g., Ball-Pivoting surface mesh reconstruction algorithms [8]. However, such a method is ill-suited for real-time applications as surface mesh reconstruction can be computationally expensive, e.g., we observe that it takes about 3 seconds to perform mesh reconstruction on a five-view dense point cloud. In this section, we describe two novel techniques called *multi-resolution projection* and *anchor extrapolation* for generating near-field and far-field portions of the environment map in real-time.

4.4.1 Multi-resolution Projection. We propose a lightweight technique called multi-resolution projection to convert the near-field dense point cloud to the respective portion in the environment map. We use the common equirectangular format to present the environment map. Specifically, multi-resolution projection projects a point cloud into a set of environment map images with decreasing resolutions and addresses the inter-point connection and occlusion at the 2D-pixel level. When projecting a point cloud onto one environment map image, multi-resolution projection first converts the position of the point cloud from the Cartesian coordinate

system to the Spherical coordinate system. Then, for each point, we calculate its 2D projection coordinate on the environment map based on the angle values of its spherical coordinates. Then, we assign the point cloud color to the corresponding pixel on the environment map. As multiple points can be projected to the same pixel of the environment map, for each pixel, we handle the point occlusion by selecting the shortest-distant projected point to color the pixel. Figure 7a illustrates an example of three-level projection.

However, when the point cloud density is low, e.g., due to low capturing resolution, projecting point cloud only onto one environment map image resolution may lead to degraded visual quality. For example, it might result in an image with discretely projected points rather than a continuous view of the scene, and it might not adequately represent the inter-point occlusion. To address these issues, we assign different size values for projected points via multi-resolution image projection. We first project the point cloud into a series of images with decreasing resolutions. Then we scale all the projected images to the largest resolution via the nearest pixel interpolation. Finally, the multi-resolution projection results are merged into a single environment map by selecting the shortest-distant projected point to the reconstruction position from each projected image per pixel. If multiple projections have the same distance, we select the one from the highest resolution as it has more visual details.

We note that the number of resolution levels and per-level resolution can be adjusted for different combinations of dense point clouds and reconstruction positions. However, our design of the near-field boundary described previously in §4.2.1 suggests that all reconstructed near-field dense point clouds will be confined to a cubic space. Thus, it is possible to have a relatively fixed configuration to handle various scenes. In this work, we choose two resolution levels with per-level resolution as 1024x512 and 512x256, unless otherwise specified.

4.4.2 Anchor Extrapolation. Recall that by now, our two-field lighting reconstruction has generated a colored unit sphere-based point cloud (USPC) for the far-field lighting. To generate the corresponding environment map in the equirectangular format, we use the anchor points to color each environment map pixel. However, the USPC, by design, only has a fixed number of anchor points. Therefore, directly projecting anchor points to the environment map is likely to lead to many empty pixel values. To address this problem, we design an *anchor extrapolation* technique that calculates each pixel value as a weighted average of USPC anchor values. This technique, in essence, assigns color value to pixels by extrapolating from their nearby anchor colors and will result in a gradient coloring and blurring effect.

Specifically, we first initialize each pixel of the environment map with a normal vector, i.e., a unit vector from the sphere center to the pixel position. The initialization is feasible as a pixel in the equirectangular format of an environment map can be easily presented in the spherical coordinate system. We then calculate the i^{th} pixel color c_i using the following equation:

$$c_i = \frac{2}{N} \sum_{j=1}^{N} \max(\vec{p}_j \cdot \vec{n}_i, 0)^w c_j, \tag{1}$$

where $\vec{n}_i$ represents the pixel normal vector, N is the number of anchors, $\vec{p}_j$ and c_j are the normal vector and color for the j-th anchor, respectively. Note that the dot product between $\vec{p}_j \cdot \vec{n}_i$ is effectively the cosine value of the angle between these vectors, as $|\vec{p}_j| = |\vec{n}_i| = 1$. The max function effectively filters out all the anchor points in the hemisphere opposite the i^{th} pixel. Furthermore, w is an exponent controlling the blurring level of far-field reconstruction. Intuitively, a smaller w value will lead to more anchor points used for the pixel calculation. Thus, a smaller w value will result in a blurrier environment map, while a larger w will produce a clearer environment map, as demonstrated in Figure 7b. In this work, we set w to be 128.

Note that calculating the pixel color using Equation (1) can be time-consuming as the weighted average has to iterate through all anchor points. However, anchors do not contribute equally to the pixel color calculation. Intuitively, an anchor j that has a smaller $\max(\vec{p}_j \cdot \vec{n}_i, 0)$ decreases more quickly with the power w. Such anchors

are also farther away from the pixel of interest than anchors with a larger $\max(\vec{p}_j \cdot \vec{n}_i, 0)$ value. In fact, we find that when $w = 128$, only the 32 nearest anchors out of the 1280 contribute significantly (i.e., $\max(\vec{p}_j \cdot \vec{n}_i, 0) > 0.1$). Thus, to speed up the pixel color calculation, we precompute the 32 nearest anchors for each pixel and their respective cosine values. The precomputation effectively reduces the number of anchors by a factor of 40 and allows the use of cached results for the weighted average calculation. Figure 7b shows that our acceleration has minimal visual impact.

4.5 LitAR Quality-Performance Configurations

4.5.1 Reconstruction Session Settings and Initialization. LitAR uses a *lighting reconstruction session* to manage each multi-view reconstruction task. A lighting reconstruction session has the same lifecycle as its corresponding virtual object; the session is created when a virtual object placement request is issued and is destroyed when the placed object is removed from the scene. As AR applications might need multiple virtual objects in the view, LitAR supports multiple active lighting reconstruction session per AR session (i.e., during the AR application's lifetime). At the beginning of each session, LitAR collects static device-specific information, e.g., camera intrinsic, current ambient lighting data, and camera image native resolutions, to bootstrap subsequent lighting reconstruction operations.

LitAR supports configuring several knobs, including color image sampling rate, number of views, multi-resolution projection resolution levels, and environment map size, that trade-off visual quality and reconstruction performance. These knobs can be categorized into three types, i.e., data capturing, two-field lighting reconstruction, and environment map rendering. Thus, the startup latency of each session and the subsequent near/far-field reconstruction depend on the specific configurations. The users (e.g., mobile AR developers) can configure each lighting reconstruction session based on performance requirements or select one of the three presets: low, medium, and high. In §6.1.2, we will show that all three presets achieve better visual quality than ARKit but take an increasing amount of time to generate an environment map.

4.5.2 Point Cloud Management. To achieve low-latency point cloud operations, LitAR leverages the edge to generate, manage, and transform both the sparse and multi-view dense point clouds. To exploit the inherent parallelism of point cloud operations, LitAR performs these operations on the GPU. However, even with unified memory, the managed memory still must be copied to the GPU memory (by the driver) for data access. A naive implementation may lead to expensive GPU memory access overhead. Thus, we carefully design the memory layout using a continuous memory buffer to store the multi-view dense point cloud and a fixed number of anchor points. When new view data is processed, LitAR overwrites the point cloud memory buffer by replacing the oldest data for *temporal* consistency or replacing the data with the same view identifier for *spatial* consistency. This fixed-view design keeps the memory layout unchanged, thus avoiding paging setup overhead while still producing high-quality environment maps.

Additionally, LitAR includes an asynchronous point cloud registration to address the mobility-induced noises, which can lead to misaligned point positions. In other words, LitAR runs point cloud registration in parallel to the main two-field lighting reconstruction and will update the environment map with the aligned point cloud once the registration completes. We note that the point misalignment is mainly due to the inaccurate device tracking data provided by the AR framework, in this case, ARKit. Providing accurate device tracking information is an essential but orthogonal research question; prior work such as ORB-SLAM2 [44] and Edge-SLAM [7] can achieve good tracking in about 26ms-50ms. In this work, we use the iterative closest point registration [9] to mitigate the impact of noisy tracking data on the lighting reconstruction. During our preliminary study, we found that point cloud registration is not always necessary (e.g., when mobile AR users are relatively static) and can take significantly longer than other operations (e.g., 200ms for handling five views with 1024x768 points). In our implementation, the point cloud registration is turned off by default.

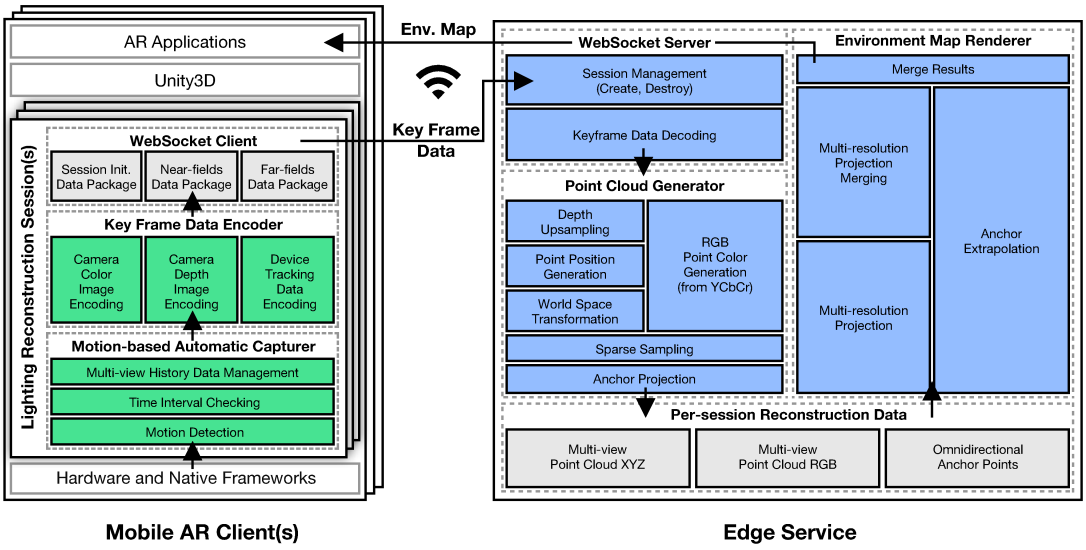


Fig. 8. A system architecture overview of LitAR. The framework provides high-quality lighting as an environment map for mobile AR applications with two logical components: client-side and edge service. LitAR can support lighting for multiple reconstruction positions per AR session on the client-side and allows multi-user scene data sharing via the edge service.

5 IMPLEMENTATIONS

We implement all the techniques described in §4 in a prototype system called LitAR (§5.1). To facilitate experiments in a controlled manner, we also develop a mobile AR simulator based on Unreal Engine (§5.2).

5.1 Edge-Based Prototype

We implement LitAR in C# and Python with about 2.2K lines of code. Figure 8 shows an architecture overview of LitAR. LitAR consists of two logical modules: a client-side that captures, encodes, and sends the data necessary for the lighting reconstruction sessions; an edge-side that decodes the keyframe data, generates intermediate point clouds, and renders the environment map. LitAR currently supports AR applications built with Unity3D [60] and AR Foundation [59].

5.1.1 Client-side. The client module of LitAR is implemented as a Unity package. We provide an entry script called *ARLightingReconstructionManager* as a *MonoBehavior* subclass to allow the developer to specify system configurations and visual quality-related information via the Unity editor UI. *ARLightingReconstructionManager* also manages the memory usage and function calls of all lighting reconstruction sessions in an application's lifecycle. We implement the *motion-based automatic capturer* by leveraging the AR device tracking data provided by AR Foundation to automatically capture environment data that will not be subjected to camera motion blurs. We perform the nearest neighbor sampling on the device's native color and depth image data for the visual data. Specifically, for the AR application running on our testing device, iPad Pro, we sample color and depth images in the format of YCbCr 4:2:0 and float32, respectively. The captured color and depth images are then encoded with a one-byte unique package identifier and device tracking information into three binary data packages, i.e., session initialization, near-field, and far-field. We refer to these data packages as *keyframe data*. LitAR also manages a

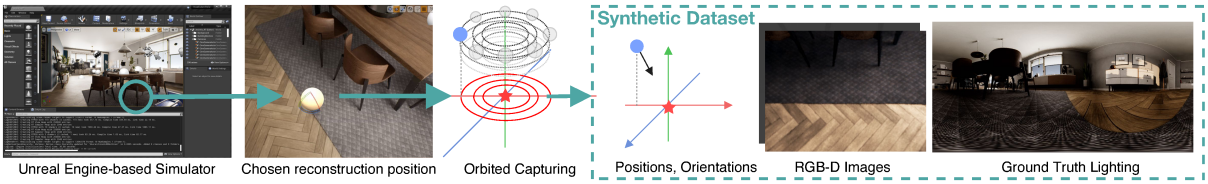


Fig. 9. Unreal engine-based simulator and synthetic dataset generation process. We generate an indoor synthetic dataset using an Unreal Engine-based simulator. We use an orbit trajectory at each manually chosen position to build observations around the position and place the virtual camera to capture images. We extract camera positions, orientations, RGB-D images, and ground truth environment map.

WebSocket session for low latency communication with the edge server, i.e., sending keyframe data and receiving environment map.

5.1.2 Edge-side. On the edge side, we implement a Tornado-based [56] WebSocket service to communicate with the client. The WebSocket server dispatches the package to different operations for each received binary data package based on its package identifier. We leverage NumPy [29] to decode and convert the received binary packages into different data types and structures. To improve the performance of point cloud-related operations, including point cloud generation, multi-resolution projection, and anchor extrapolation, we implement them in CUDA kernels using the Numba library [37]. As such, these operations run on the edge GPU. In our edge system memory management, we leverage the unified memory [13] to avoid time-consuming data copying between CPU and GPU. Our edge server implementation can also fall back to traditional GPU memory without any modification to support other GPU hardware that does not have unified memory.

5.1.3 Mobile-edge communication. We design a low-overhead and compact networking communication scheme to support the goal of low-latency lighting reconstruction. Both the near and far-field data are serialized into binary formats. Specifically, we stream only the device's native color and depth data for near-field data and reconstruct the point cloud on the edge side. Compared to directly streaming float32-encoded point cloud in the XYZRGB format, we need at most 22.9% of the bytes. Also, as we only capture sparse camera images during far-field reconstruction, the far-field keyframe data is significantly smaller than the near-field counterpart (1189 bytes vs. 270389 bytes).

5.2 Unreal Engine-Based Mobile AR Simulator

We implement a mobile AR simulator by leveraging a high-fidelity 3D graphics rendering engine, Unreal Engine [21]. Figure 9 presents the workflow of our simulator. First, we use Unreal Engine 4 to create a photorealistic indoor scene based on the *ArchViz*[1] project, which provides high-quality architectural visualization for interior design. Next, we create a virtual camera using the Blueprints Visual Scripting[2] system that takes controlled variables to modify the camera's movement and internal properties, e.g., FoV. We simulate the device/user movement by moving the virtual camera within a photorealistic 3D indoor scene. Finally, we can configure the simulator with desired variable values to generate a synthetic dataset for rendering purposes, including camera observations and environment physical properties. Our simulator can be extended to support future studies and bears the following advantages over a device-based setup: *(i)* our simulator makes it easier to extract high-quality environment lighting and physical information to serve as the ground truth. At the same time, obtaining such

[1]https://docs.unrealengine.com/4.27/en-US/Resources/Showcases/ArchVisInterior/
[2]https://docs.unrealengine.com/5.0/en-US/blueprints-visual-scripting-in-unreal-engine/

information can be expensive or unpractical due to physical limitations on measurement and observation. *(ii)* it is easier to study individual factors in isolation by applying controlled changes to the scene environment and simulated mobile devices.

5.2.1 AR Virtual Object Rendering. We develop a browser-based renderer using the Three.js rendering framework [55] to automate the process of rendering virtual objects of interest. Specifically, our renderer uses information, including reconstructed lighting, the camera position, and properties, from our synthetic dataset to render a 3D virtual object at the resolution of 1024x768. The renderer then trims empty pixels outside rendered objects to remove the object-to-frame size impact on PSNR calculation when using different camera FoV settings. The resulting images of rendered objects serve as the basis for comparing different lighting reconstruction methods.

6 EVALUATION

We evaluate the performance of LitAR using a lab testbed and the simulator. The lab testbed includes a LiDAR-enabled iPad Pro serving as the client and a Jetson Xavier NX [45] board serving as the edge server. The iPad and the Jetson board communicate via resident WiFi with an average latency of 7.08 ms (± 3.31 ms) and network bandwidth of 508 Mbits/sec (± 12 Mbits/sec). For testbed-based experiments, we choose three different indoor scenes and compare LitAR with three different baselines: *(i)* ARKit 5 [34], a commercial AR framework developed by Apple; *(ii)* LitAR with *point cloud registration* turned on; *(iii)* LitAR with *mesh reconstruction* instead of the lightweight multi-resolution projection module. We use the Environment Probe [33] feature of ARKit to generate environment maps. The lighting estimation feature of ARKit is backed up by EnvMapNet [51]. We measure both the reconstruction time and visual quality for all the methods. We use the Peak signal-to-noise ratio (PSNR) and Structural Similarity Index (SSIM) for quantitative visual quality comparison. The PSNR and SSIM values of each method are calculated by comparing the rendered virtual object to the physical object. In this work, we use the classical physical mirror ball as it can be easily acquired. The higher the values of PSNR and SSIM, the better the visual performance.

We use the simulator to evaluate LitAR's performance in a wider range of scenarios. Our simulator allows easy extraction of ground truth lighting information at any reconstruction position in a photorealistic 3D indoor scene. For simulation-based evaluations, LitAR is evaluated with six objects of different shapes and materials and is compared to two baselines: *(i)* using a 360° camera at the observation position, akin to [57]; and *(ii)* Xihe, a recent academic framework that produces real-time low-frequency lighting estimation from RGB-D images [65]. We describe the synthetic dataset used in our study in §6.2.1.

To provide an in-depth evaluation of LitAR's performance, we also conduct a number of ablation studies that demonstrate the quality-performance trade-offs (§6.1.2), highlight our design choices for near-field and far-field reconstructions, as well as identify applicable scenarios (§6.2.3 and 6.2.4). The three quality presets for near-field reconstruction are configured as following: *(i)* LitAR (low): number of views is 3, color image resolution is 256x192, multi-resolution projection resolutions are [512x256, 256x128, 64x32], environment map resolution is 512x256; *(ii)* LitAR (medium): number of views is 4, color image resolution is 512x384, multi-resolution projection resolutions are [768x384, 384x192], environment map resolution is 512x256; *(iii)* LitAR (high): number of views is 5, color image resolution is 1024x768, multi-resolution projection resolutions are [1024x512, 512x256], environment map resolution is 1024x512. All three presets for far-field reconstruction have the color image resolution of 32x24 and share the same environment map resolution configurations as near-field reconstruction.

6.1 Testbed-Based System Performance

6.1.1 End-to-end Evaluation. We compare the end-to-end rendering visual results and the runtime performance of LitAR and ARKit. As shown in Figure 10 (last two columns), the virtual mirror balls rendered with LitAR have more reflection details and better color tune than those rendered with ARKit's learning-based method.

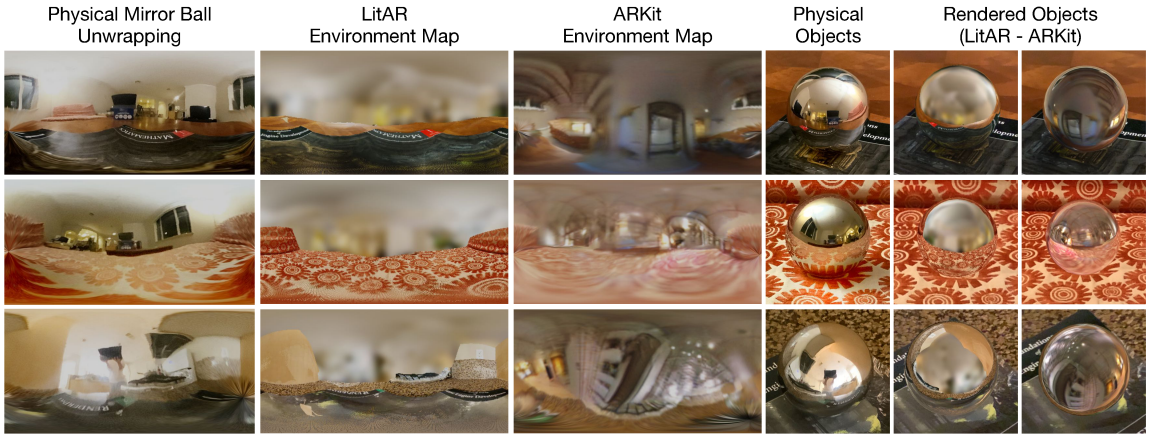
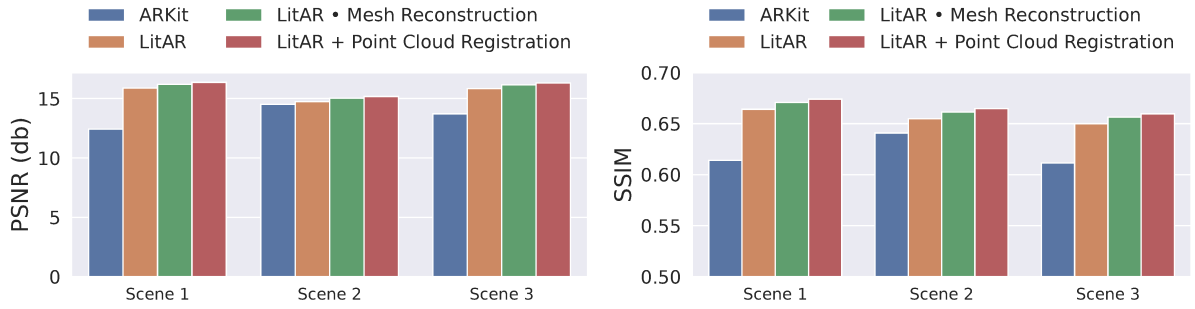


Fig. 10. Qualitative comparison between LitAR and ARKit on three real-world indoor scenes. Each row represents an indoor scene. *Column 1:* the panorama view of a scene at the reconstruction position by unwrapping the physical mirror ball reflection [15]. *Column 2:* LitAR's environment maps have good visual quality and rich details for the near-field portion while maintaining the structural similarity to the corresponding physical scene. *Column 3:* ARKit's environment maps show varying performance, sometimes completely different from the scene (the first row), while others with less visual details.



(a) PSNR comparison

(b) SSIM comparison

Fig. 11. Quantitative comparison between LitAR and ARKit in terms of PSNR and SSIM. LitAR outperforms ARKit for all three real-world scenes. Further, using the conventional mesh reconstruction in place of our lightweight multi-resolution projection only increases the PSNR/SSIM values slightly by 2%/1%. Similarly, turning on the asynchronous point cloud registration has only minor improvement by 3%/1.5%. Recall that mesh reconstruction and point cloud registration can take a few seconds (§4.4) and a couple hundred milliseconds (§4.5.2), respectively. In short, LitAR achieves the best trade-off between visual quality and runtime performance.

Specifically, for all three scenes, LitAR's virtual balls have higher visual similarity than physical mirror balls. In contrast, the virtual balls rendered using ARKit either reflect incorrect indoor scenes (especially on the far-field portion of the environment) or lack fine-grained visual details, e.g., the text on the book cover. We can more easily observe such visual quality differences by comparing the generated environment maps (the second and third columns) to the unwrapped images from the physical mirror ball. Note that the unwrapped images do not

represent the ground truth lighting as they are often distorted but can serve as a visual guide of the panorama view at the reconstruction position. By comparing the environment maps generated by LitAR to the unwrapped images, we see that LitAR can accurately reconstruct scene elements from near-field observations while faithfully recovering environmental geometry and color tone information from far-field observations.

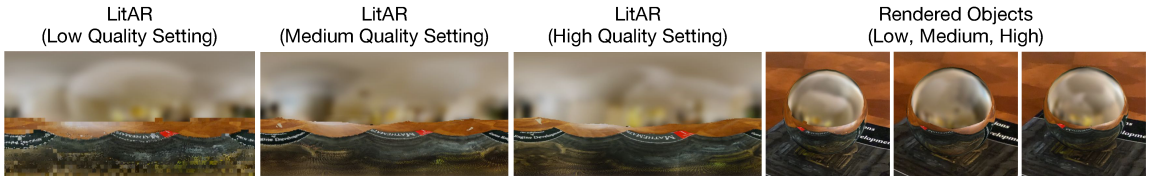
Figure 11 quantifies the visual quality using two commonly used image metrics, i.e., PSNR and SSIM, by comparing the rendered object to a physical mirror ball image at the same reconstruction position. We see that LitAR outperforms ARKit on all three real-world captured scenes, with up to 14.3% higher PSNR and 5.5% higher SSIM. When replacing our lightweight multi-resolution projection component of LitAR with the Ball-Pivoting surface mesh reconstruction [8], we only notice a minor increase in the PSNR/SSIM values. This observation demonstrates the effectiveness of multi-resolution projection for generating high-quality near-field reflections. Similarly, we do not observe significant improvement when running LitAR with the point cloud registration component. We suspect this is because AR frameworks such as ARKit can provide reasonable device tracking data in most cases with slow movement. We observe that the tracking data of ARKit often drifts in cases of fast movement, making the point cloud registration component integral. We omit their visual effect comparisons as both of the LitAR's variations do not show noticeable visual quality differences to LitAR.

Finally, the average end-to-end latency of near-field and far-field reconstruction is 134.4 ms and 57.5 ms, respectively. Detailed component-wise time breakdown is discussed in the next section. These latencies translate to updating high-quality lighting roughly at 22 fps, i.e., every 134.4 ms LitAR can provide one near-field and two far-field environment maps. Such update frequency should be sufficient for most AR applications [61, 62]. For AR applications that require higher update frequency, we can either resort to more powerful edge servers (currently using an energy-efficient Jetson board) or use a lower quality setting, as discussed in the next section.

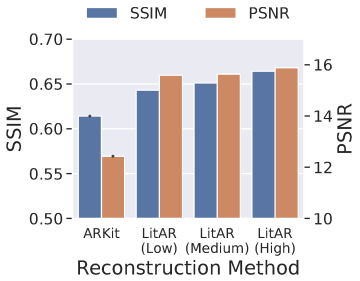
6.1.2 Trade-Offs Between Rendering Quality and Runtime Performance . We compare the rendering quality and latency of LitAR under different presets. Figure 12a shows the corresponding visual results. We note that the environment maps generated at all three settings present visually coherent near-field reflection and correct anisotropic far-field color tones. We can observe some pixelation effects in the environment map and the rendered mirror ball object for the low-quality preset due to low capturing resolution. All three settings achieve better quality than ARKit in terms of PSNR and SSIM values. For example, the low-quality preset has a 4.5% higher SSIM value than ARKit. Moreover, the difference in visual quality among the three presets is marginal, with only up to 3.5% between low and high quality.

Furthermore, we measure the time breakdown of LitAR's near-field and far-field reconstructions. Table 12c shows the average performance over three runs. For near-field reconstruction, the processing time of each component increases with the quality setting. For example, the time to encode the camera observations sees similar increases as the capturing resolutions, about 10X with 16X more pixels. We note that with the high-quality setting, the total time to encode and upload data takes 68.4 ms, about 1.9X of the environment map downloading time, even though the uploading/downloading resolution ratio is 1.5X. In contrast, in the medium-quality setting, with the same uploading/downloading resolution ratio, it is 21.7% faster to encode and upload data than to download. This is because the device data is uploaded in the format of YCbCr 4:2:0, which has a smaller data size than the RGB environment map under the same resolution. Note that we are sending back the uncompressed RGB environment map for quality consideration. This observation suggests an interesting trade-off presented by the data encoding scheme in a real-world deployment. Moreover, this result also demonstrates that network-related operations (an artifact of using the GPU-based edge device) take up most of the end-to-end time, at 62.6%, 72.6%, and 77.5% for low, medium, and high-quality settings, respectively. The network performance bottleneck implies an immediate performance gain by directly using mobile GPU to run the entire reconstruction pipeline.

The far-field reconstruction presents a similar but slower upward increase in total time with the quality presets. In particular, the time to decode/offload image data and generate a sparse point cloud is the same for all three



(a) Rendering visualization on different quality presets



(b) Visual quality comparison

System Component	Near-field (ms)			Far-field (ms)		
	low	Medium	High	low	Medium	High
Data Encode	5.11	15.01	50.13	0.18	0.18	0.18
Data Offload	4.07	7.78	18.24	0.55	0.55	0.55
Dense Point Cloud Generation	8.39	9.71	15.62	N/A	N/A	N/A
Multi-resolution Projection	9.02	9.89	14.56	N/A	N/A	N/A
Sparse Point Cloud Generation	N/A	N/A	N/A	0.03	0.03	0.03
Anchor Extrapolation	N/A	N/A	N/A	19.03	20.32	21.72
Environment Map Download	20.01	29.11	35.83	20.02	29.71	35.03
End-to-end Reconstruction	46.6	71.5	134.38	39.81	50.79	57.51

(c) Reconstruction latency

Fig. 12. LitAR rendering quality-performance trade-offs. We show the time breakdown for LitAR to generate a near-field and far-field environment map. All quality settings achieve better SSIM than ARKit. A large portion of time, at least 26.46%, was spent on edge-related operations (data encode/offload and environment map download). This observation suggests that LitAR has the promise to deliver high-quality environment maps directly on the device as mobile device GPU becomes more powerful.

quality presets. In all settings of far-field reconstruction, we downsample the native color images from 1920x1440 to a fixed resolution of 32x24. The anchor extrapolation and environment map downloading time increase slightly with the environment map size. To put the latency performance of LitAR in context, we note that a recent physical probe-based framework requires 30~400 ms to generate environment map [47]. To understand the ARKit's performance, we use the inference time of its underlying deep learning model EnvMapNet [51] since ARKit does not expose APIs to measure the end-to-end environment map generation. Even though the EnvMapNet model can run in 9 ms, as reported by Somanath et al. [51], we have shown previously that it often leads to inferior visual quality compared to LitAR. Moreover, the total time for ARKit to generate an environment map is likely to be similar to LitAR if accounting for other necessary steps, including data capturing and memory copying. In short, this detailed breakdown analysis demonstrates that far-field reconstruction can achieve real-time performance for all presets; when used in conjunction with near-field reconstruction, the mobile AR applications can receive a sufficient number of environment map updates per second.

6.1.3 Impacts of User Movement and Dynamic Scene. We investigate the impact of user movement on the rendering quality of LitAR. Specifically, we are interested in understanding the need for our motion-based automatic capturing policy. One of the authors (referred to as the participant) used the LitAR-powered AR app on the iPad to perform a virtual object placement following a pre-determined trajectory. The participant was instructed to keep the same distance to the placement position and only to move the iPad around the placement position, resulting in a semi-circular trajectory. Further, the participant was asked to pause the movement every 30 degrees and to keep the camera views centered at the placement position. The experiment was repeated with three timer values, i.e., disabled, 300 ms, and 500 ms, which control the frequency to check the movement.

First, we observe that our motion-based automatic capturing policy successfully detected the movement and resulted in six captured views (captured when the participant was static). Second, when comparing the generated environment map to the ground truth environment map captured by the mirror ball (at the same placement position), we did not observe noticeable visual quality differences for all three timer values. For example, the PSNR values for the timer=300 ms stayed relatively the same for the entire experiment, at 13.37 db (± 0.25 db). Our observations suggest that LitAR can provide visually coherent renderings under user movement when the physical scene is static. Additionally, increasing the number of views (i.e., from one to six) provides limited visual quality improvement. This is intuitive as most near-field observations can be captured in a single view when the environment is simple and static.

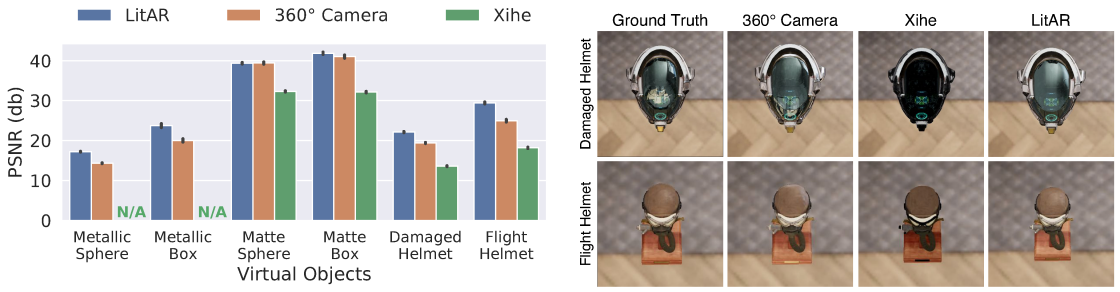
In a second experiment, we created a simple dynamic scene by manually moving the physical object within the near-field observations. Specifically, the participant was asked to fix the iPad's position and select the placement position on a math book (similar to Figure 1). While using the AR app, the participant moved the book in various directions. We observed that LitAR could update the virtual object reflection to present details of different parts of the book. Even though the lighting reconstruction task does not block the rendering task, we still observed slightly choppy reflections. Please refer to the accompanying video for the visual quality demonstration. Two key factors impact the choppiness: *(i)* the physical scene change rate and *(ii)* the reconstruction time. If the physical scene changes very rapidly (e.g., faster than the reconstruction time), the virtual object reflection will be perceived to lag. In addition to further speed up the lighting reconstruction, we suspect techniques that smooth the transition between two distinct environment maps (e.g., image fade in) and policies that pipeline the lighting reconstruction requests to mask network latency can also improve the user-perceived performance. We leave such investigations as future work.

6.2 Simulation-Based Performance Evaluation

6.2.1 Synthetic Dataset Generation. We describe the methodology we followed to generate a synthetic dataset using the Unity-based simulator (see Figure 9). In a synthetic indoor scene, we first manually choose ten reasonable positions to be considered as lighting reconstruction positions for placing virtual objects. Example reconstruction positions include on the floor or table. We vary several factors for each reconstruction position, including the number of capturing positions, mobile user/device height, and observation distance, to generate 72 camera observations. Specifically, we set up a circular capturing trajectory with eight positions by evenly dividing the trajectory. We decide the height and radius of the capturing trajectory by simulating possible scenarios when the mobile user is holding the device at chest height from a reasonable distance to the reconstruction position. We choose three typical human height values at $\{160, 170, 180\}$ centimeters and calculate the height of the trajectory by multiplying the user's height by 0.8 [17]. We further measure the radius of the trajectory using the number of steps and choose three possible values of $\{0.5, 1, 1.5\}$ steps and use the height multiplied by 0.3 as the step length [49]. For each camera observation, we export the camera HDR observation image, depth image, position, orientation, and ground truth lighting in the format of an equirectangular panorama image.

6.2.2 End-to-end Visual Quality Comparison. We compare the end-to-end rendering performance quantitatively and qualitatively on six different virtual objects. For this experiment, we configure the simulator to run the two-field lighting reconstruction to process one near-field observation and nine far-field observations based on the guided movement policy. For near-field reconstruction, we use mesh reconstruction instead of multi-resolution projection to support the high-quality point projection. The following results showcase the upper bound of visual quality that LitAR can achieve.

Figure 13 shows the comparisons of PSNR values. Specifically, on complex objects with physically-based materials (i.e., *Damaged Helmet* and *Flight Helmet*), LitAR achieves 44.1% and 12.1% higher values of PSNR than a recent deep learning-based AR lighting estimation system [65] and the lighting information captured by a



(a) PSNR comparison

(b) Rendering visual comparison

Fig. 13. Simulator-based visual quality comparison. LitAR achieves better rendering effects, i.e., higher PSNR values (calculated against ground truth rendering), than other techniques for all objects. The PSNR comparisons to Xihe are omitted for the first four virtual objects as Xihe only provides spatial-variant low-frequency lighting estimation [65]. Virtual objects are rendered with lighting provided by ground truth lighting, 360° camera, Xihe [65], and LitAR.

360° camera, respectively. This result indicates that by correctly leveraging user movement and scene geometry information, LitAR can generate highly accurate lighting from limited camera observations. In addition, as we will see in Figure 14, the rendering performance of LitAR is roughly the same with fewer observations. Note that we omit the comparison to the rendering PSNR by Xihe on metallic objects (*Metallic Sphere* and *Metallic Box*) because Xihe only provides low-frequency lighting in SH coefficients format, which does not support reflective rendering.

Figure 13b compares the visual effect of objects rendered with different lighting information. We observe that LitAR produces visually coherent virtual objects. This observation suggests LitAR is effective in generating a complete high-quality environment map. Compared to those rendered with the 360° camera observations, helmets rendered with LitAR exhibit higher structural similarity to the ones rendered with ground truth lighting. For example, the two reflective regions in the lower bottom of the *Damaged Helmet* are visually separated.

6.2.3 Ablation Study of Near-Field Lighting Reconstruction. This section demonstrates the effectiveness of our two-field lighting reconstruction for the near-field observation and highlights the importance of LitAR's far-field design. As we have designed LitAR to progressively improve the intermediate point cloud by naturally exploiting mobile user/device movement, we evaluate the impact of the number of near-field observations on the rendering results We set up the experiment using our simulator as follows: *(i)* for each observation position, we combine camera observations from 3, 5, 7 nearby positions on the orbiting trajectory; *(ii)* we use mesh reconstruction with LitAR for combined observations. To eliminate the impact of far-field observations, we use a single dominant image color to fill the far-field portion of the environment map to simulate the ambient light sensor data. Figure 14 compares the rendering accuracy for different numbers of observations. We observe that the rendering PSNR values only increase *slightly* with the number of observations. This observation suggests that only a small portion of the environment map needs to be processed with depth information, further motivating our design choice of reconstructing near-field and far-field observations separately. Furthermore, we show that rendering SSIM values increase significantly, 0.057 on average across the tested positions in Figure 14, with the number of observations for the *Metallic Sphere* object for all ten tested reconstruction positions. This result is intuitive since more complete near-field reflections will improve the structural similarity. However, higher SSIM values do not

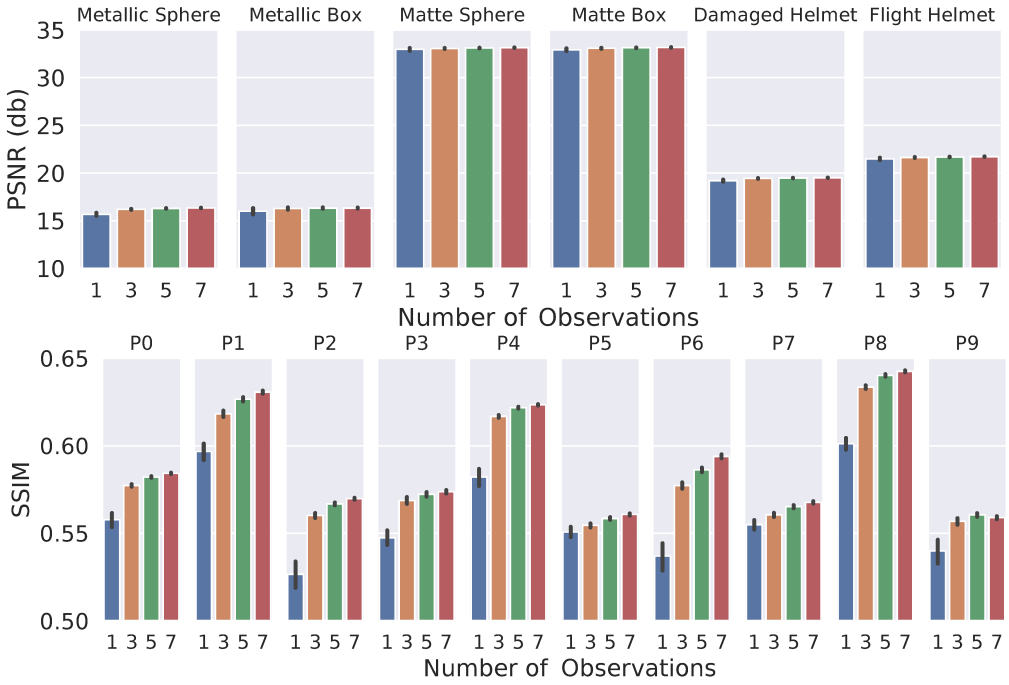


Fig. 14. Quantitative comparison of rendering quality with different numbers of camera observations. The performance of LitAR, measured in PSNR, only exhibits slight improvement with more observations, while the SSIM values increase significantly for reflective materials.

always guarantee better PSNR values, thus implying the necessity to use both metrics in quantitative studies for lighting reconstruction.

6.2.4 Ablation Study of Far-Field Lighting Reconstruction. So far, we have demonstrated the effectiveness of LitAR and its spatial variance-aware near-field reconstruction component. In this section, we evaluate the performance of LitAR's directional-aware far-field lighting reconstruction and the effectiveness of our guided movement policy. We evaluate the rendering performance with different numbers of guided far-field observations. Recall that guided movements naturally increase the observed scene, allowing LitAR to extrapolate environment map pixel color closer to the ground truth. Figure 15 shows the comparison of rendering performance. We observe that for all tested objects, having access to more guided observations improves the PSNR value by up to 31.04%. Furthermore, far-field observations present different levels of impact on objects with different shapes. For example, both box-shaped objects are improved more significantly than spherical objects. This finding, if generalized, can help further improve usability by providing guided movements for different objects.

7 RELATED WORK

7.1 Mobile-specific Lighting Support

As mobile device capability increases and AR re-emerges in user-facing applications [2, 35], obtaining environment lighting for photorealistic rendering has garnered increasing interests in the research communities [12, 47, 51, 64, 65]. For brevity, we only discuss techniques targeted at lighting for indoor scenes. On the more theoretical front,

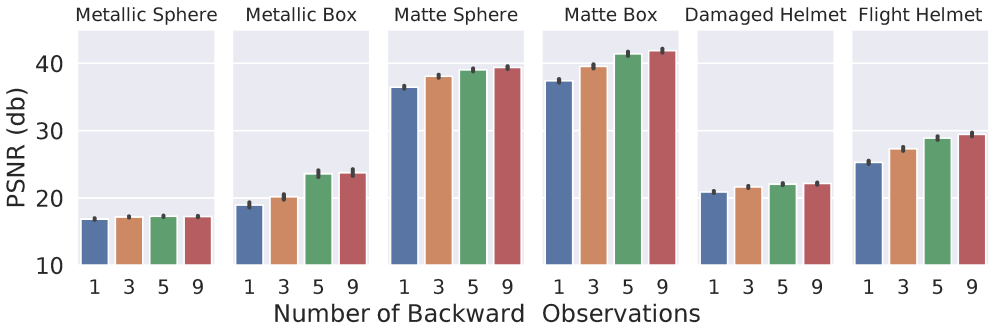


Fig. 15. Quantitative comparison of rendering quality for guided movements. Increasing the number of backward far-field observations have different levels of improvement for tested objects.

Cheng et al. leveraged both the rear and front cameras to estimate spherical harmonics coefficients [12], which is a low-frequency lighting representation that does not support reflection. Zhao et al. proposed a two-staged pipeline called PointAR that leverages the mobile depth sensor to estimate spatially-variant lighting [64]. Somanath et al. introduced an efficient deep learning (DL) model called EnvMapNet that generates HDR environment maps from LDR images [51]. In contrast to previous learning-based approaches, our work LitAR directly generates high-quality environment maps with the core technique of two-field lighting reconstruction and several practical optimizations, including multi-resolution projection. As such, LitAR is not subject to common limitations of DL-based methods such as training data availability and inference performance on heterogeneous mobile resources. We also note that LitAR includes a simulator that leverages 3D indoor scenes to conduct controlled experiments, thus avoiding the need for an expensive manual process of obtaining ground truth lighting information.

7.2 System Supports for Lighting

On the system front, commercial SDKs such as ARKit [34] and ARCore [25] provide easy-to-use lighting estimation APIs for mobile AR application development. Two recent academic frameworks improved upon commercial solutions: GLEAM provides a real-time mobile illumination framework that supports reflective virtual objects with the use of physical probes [47]; Xihe introduced a 3D-vision based framework that provides adaptive lighting estimation [65]. We design LitAR from the outset by considering mobile characteristics, including limited FoVs, natural device/user movements, and leveraging edge GPU assistance, which well positions it for high-quality and efficient reconstruction of environment maps.

7.3 Image-based Lighting

In addition to methods for mobile-specific lighting discussed above, many researchers have investigated image-based lighting [14, 15, 36]. For example, numerous works designed approaches for generating environment lighting representations from camera videos [28, 30, 58], and assisted lighting reconstruction with physical probes [16], object cues [54], or scene geometry [6, 41]. Recent work is DL-based primarily and can broadly fall into two types based on the output, i.e., estimating low-frequency lighting [24, 53] or high-frequency lighting [22, 52]. For example, Gardner et al. proposed to divide the HDR environment map learning task into two subtasks and generated one lighting estimation result per image [22]. Even though this work can handle the rendering of specular objects, it does not consider spatial variance. On the contrary, both Garon et al. [24] and Lighthouse [53] support spatially-variant lighting but are limited in rendering reflective materials. Our work falls in between these two types of work by generating an environment map that consists of near-field and far-field components. This hybrid environment map allows more effective reconstruction within mobile constraints such as user movement

and depth-sensing accuracy while still achieving visually coherent rendering for various objects, including reflective ones.

8 CONCLUSION

In this work, we introduced an end-to-end lighting reconstruction system called LitAR that generates high-quality environment maps for mobile AR applications. As quantitatively and qualitatively demonstrated, AR applications can use environment maps reconstructed by LitAR to render objects of various properties, including reflective materials, with 14.3%/5.5% higher PSNR/SSIM and better visual coherence than ARKit. We showed that LitAR could produce virtual objects with more realistic and visually coherent reflection, as well as fine-grained visual details. We used physical object images for testbed-based experiments to serve as the basis of desired visual quality. Furthermore, using our simulator, we compared against other techniques, including Xihe and 360° camera, by having access to ground truth lighting. We have released our research artifacts at https://github.com/cake-lab/LitAR to facilitate future research work in our community.

Aside from the realistic and visually coherent rendering goal, we designed LitAR with mobile-specific constraints, e.g., limited sensing and data noise, in mind. By exploring mobile user behaviors and working within mobile sensing constraints, we proposed the two-field lighting reconstruction scheme that divides camera observations into near-field and far-field observations based on pixels' relative distance to the reconstruction position. LitAR can work with as few as one camera observation and can progressively improve the quality of generated environment maps, especially for metallic objects, with more camera observations. Keeping usability in mind, we further introduced the motion-based automatic capture and guided bootstrapped movement policies to help AR users capture higher quality data and more suitable camera observations. LitAR significantly speeds up both the near-field and far-field reconstructions by two novel point cloud techniques, i.e., multi-resolution projection and anchor extrapolation. Last but not least, LitAR provides three quality presets and exposes several knobs for mobile AR applications to make reconstruction quality and time trade-offs based on their specific use cases.

We evaluated LitAR's performance with a lab-based testbed and a game engine-based simulator. We observed that LitAR could generate higher-quality environment maps than ARKit and result in rendered objects with up to 14.3%/5.5% higher PSNR/SSIM compared to the physical counterpart. Furthermore, we showed that multi-resolution projection significantly reduces the point cloud projection from 3 seconds (using mesh reconstruction) to 14.6ms. Overall, LitAR can generate about 22 high-quality environment maps per second when point cloud registration is not required. As we design the point cloud registration to run asynchronously, the registration step will not block the main reconstruction pipeline; once completed, LitAR will send an updated environment map to the mobile device. As part of the future work, we will explore techniques to improve the details of the generated environment maps and design runtime policies to handle temporally variant lighting more robustly.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive reviews. This work is partly supported by NSF Grants CNS-1815619, NGSDI-2105564, and VMWare.

REFERENCES

[1] Ibraheem Alhashim and Peter Wonka. 2018. High Quality Monocular Depth Estimation via Transfer Learning. *arXiv e-prints* abs/1812.11941, Article arXiv:1812.11941 (2018). arXiv:1812.11941 https://arxiv.org/abs/1812.11941

[2] Amazon. 2020. Amazon AR View. https://www.amazon.com/adlp/arview. Accessed: 2020-7-2.

[3] Christopher Andrews, Michael K Southworth, Jennifer N A Silva, and Jonathan R Silva. 2019. Extended Reality in Medical Practice. *Curr. Treat. Options Cardiovasc. Med.* 21, 4 (March 2019), 18.

[4] Apple. 2022. ARCoachingOverlayView. https://developer.apple.com/documentation/arkit/arcoachingoverlayview.

[5] Apple. 2022. iPhone 13 Pro Tech Specs. https://www.apple.com/iphone-13-pro/specs/.

[6] Dejan Azinovic, Tzu-Mao Li, Anton Kaplanyan, and Matthias Nießner. 2019. Inverse path tracing for joint material and lighting estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Long Beach, CA, 2447–2456.

[7] Ali J Ben Ali, Zakieh Sadat Hashemifar, and Karthik Dantu. 2020. Edge-SLAM: edge-assisted visual simultaneous localization and mapping. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services (MobiSys '20)*. 325–337.

[8] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. 1999. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics* 5, 4 (1999), 349–359. https://doi.org/10.1109/2945.817351

[9] Paul J Besl and Neil D McKay. 1992. Method for registration of 3-D shapes. In *Sensor fusion IV: control paradigms and data structures*, Vol. 1611. Spie, 586–606.

[10] Dan Cernea. 2020. OpenMVS: Multi-View Stereo Reconstruction Library. (2020). https://cdcseacave.github.io/openMVS

[11] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niebner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. 2017. Matterport3D: Learning from RGB-D Data in Indoor Environments. In *2017 International Conference on 3D Vision (3DV)*. IEEE Computer Society, 667–676.

[12] Dachuan Cheng, Jian Shi, Yanyun Chen, Xiaoming Deng, and Xiaopeng Zhang. 2018. Learning Scene Illumination by Pairwise Photos from Rear and Front Mobile Cameras. *Comput. Graph. Forum* 37, 7 (2018), 213–221. http://dblp.uni-trier.de/db/journals/cgf/cgf37.html#ChengSCDZ18

[13] RidgeRun Embedded Linux Developer Connection. 2022. NVIDIA CUDA Memory Management. https://developer.ridgerun.com/wiki/index.php?title=NVIDIA_CUDA_Memory_Management.

[14] Massimiliano Corsini, Marco Callieri, and Paolo Cignoni. 2008. Stereo light probe. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 291–300.

[15] Paul Debevec. 2006. Image-based lighting. In *ACM SIGGRAPH 2006 Courses*. 4–es.

[16] Paul Debevec. 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*. 1–10.

[17] Devin Larson. 2014. Standard Proportions of the Human Body. https://www.makingcomics.com/2014/01/19/standard-proportions-human-body/. Accessed: 2021-11-5.

[18] Ufuk Dilek and Mustafa Erol. 2018. Detecting position using ARKit II: generating position-time graphs in real-time and further information on limitations of ARKit. *Physics Education* 53, 3 (2018), 035020.

[19] Ruofei Du, Eric Turner, Maksym Dzitsiuk, Luca Prasso, Ivo Duarte, Jason Dourgarian, Joao Afonso, Jose Pascoal, Josh Gladstone, Nuno Cruces, Shahram Izadi, Adarsh Kowdle, Konstantine Tsotsos, and David Kim. 2020. DepthLab: Real-Time 3D Interaction With Depth Maps for Mobile Augmented Reality. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology (UIST)*. ACM, 15 pages. https://doi.org/10.1145/3379337.3415881

[20] Farshad Einabadi, Jean-Yves Guillemaut, and Adrian Hilton. 2021. Deep neural models for illumination estimation and relighting: A survey. *Comput. Graph. Forum* 40, 6 (Sept. 2021), 315–331.

[21] Epic Games. 2021. Unreal Engine - Real-Time 3D Creation Tool. https://www.unrealengine.com. Accessed: 2021-11-5.

[22] Marc-André Gardner, Kalyan Sunkavalli, Ersin Yumer, Xiaohui Shen, Emiliano Gambaretto, Christian Gagné, and Jean-François Lalonde. 2017. Learning to Predict Indoor Illumination from a Single Image. *ACM Transactions on Graphics* (2017).

[23] Marc-Andre Gardner, Yannick Hold-Geoffroy, Kalyan Sunkavalli, Christian Gagne, and Jean-Francois Lalonde. 2019. Deep Parametric Indoor Lighting Estimation.

[24] Mathieu Garon, Kalyan Sunkavalli, Sunil Hadap, Nathan Carr, and Jean-François Lalonde. 2019. Fast Spatially-Varying Indoor Lighting Estimation. *CVPR* (2019).

[25] Google. 2020. ARCore. https://developers.google.com/ar.

[26] Google. 2022. Pixel 6 Tech Specs. https://store.google.com/product/pixel_6_specs?hl=en-US.

[27] Google for Education. 2022. Bringing virtual and augmented reality to school | Google for Education. https://edu.google.com/products/vr-ar/?modal_active=none. Accessed: 2020-7-24.

[28] Thorsten Grosch, Tobias Eble, and Stefan Mueller. 2007. 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*. 125–132.

[29] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[30] Vlastimil Havran, Miloslaw Smyk, Grzegorz Krawczyk, Karol Myszkowski, and Hans-Peter Seidel. 2005. Interactive System for Dynamic Scene Lighting using Captured Video Environment Maps.. In *Rendering Techniques*. 31–42.

[31] HUAWEI. 2021. HUAWEI Mate 30 Pro Specifications | HUAWEI Global. https://consumer.huawei.com/en/phones/mate30-pro/specs/. Accessed: 2020-7-8.

[32] Apple Inc. 2020. iPad Pro 2020. https://www.apple.com/ipad-pro/specs/.

[33] Apple Inc. 2022. Adding Realistic Reflections to an AR Experience. https://developer.apple.com/documentation/arkit/camera_lighting_and_effects/adding_realistic_reflections_to_an_ar_experience.
[34] Apple Inc. 2022. Introducing ARKit 5. https://developer.apple.com/augmented-reality/arkit/.
[35] Inter IKEA Systems B. V. 2017. IKEA Place. https://apps.apple.com/us/app/ikea-place/id1279244498. Accessed: 2020-7-2.
[36] Brian Karis and Epic Games. 2013. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice* 4, 3 (2013).
[37] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. 2015. Numba: A LLVM-Based Python JIT Compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC* (Austin, Texas) *(LLVM '15)*. Association for Computing Machinery, New York, NY, USA, Article 7, 6 pages. https://doi.org/10.1145/2833157.2833162
[38] Junxuan Li, Hongdong Li, and Yasuyuki Matsushita. 2021. Lighting, Reflectance and Geometry Estimation From 360deg Panoramic Stereo. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 10591–10600.
[39] Zhengqin Li, Mohammad Shafiei, Ravi Ramamoorthi, Kalyan Sunkavalli, and Manmohan Chandraker. 2020. Inverse rendering for complex indoor scenes: Shape, spatially-varying lighting and svbrdf from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2475–2484.
[40] Z Liu, G Lan, J Stojkovic, Y Zhang, C Joe-Wong, and M Gorlatova. 2020. CollabAR: Edge-assisted Collaborative Image Recognition for Mobile Augmented Reality. In *2020 19th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 301–312.
[41] Robert Maier, Kihwan Kim, Daniel Cremers, Jan Kautz, and Matthias Nießner. 2017. Intrinsic3D: High-Quality 3D Reconstruction by Joint Appearance and Geometry Optimization with Spatially-Varying Lighting. (Aug. 2017). arXiv:1708.01670 [cs.CV]
[42] Morgan McGuire and Michael Mara. 2014. Efficient GPU screen-space ray tracing. *Journal of Computer Graphics Techniques (JCGT)* 3, 4 (2014), 73–85.
[43] Pierre Moulon, Pascal Monasse, Romuald Perrot, and Renaud Marlet. 2016. OpenMVG: Open multiple view geometry. In *International Workshop on Reproducible Research in Pattern Recognition*. Springer, 60–74.
[44] Raúl Mur-Artal and Juan D. Tardós. 2017. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *IEEE Transactions on Robotics* 33, 5 (2017), 1255–1262. https://doi.org/10.1109/TRO.2017.2705103
[45] Nvidia. 2022. Jetson AGX Xavier Developer Kit. https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit.
[46] Rohit Pandey, Sergio Orts Escolano, Chloe Legendre, Christian Häne, Sofien Bouaziz, Christoph Rhemann, Paul Debevec, and Sean Fanello. 2021. Total relighting: learning to relight portraits for background replacement. *ACM Trans. Graph.* 40, 4 (July 2021), 1–21.
[47] Siddhant Prakash, Alireza Bahremand, Linda D Nguyen, and Robert LiKamWa. 2019. Gleam: An illumination estimation framework for real-time photorealistic augmented reality on mobile devices. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*. 142–154.
[48] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. 2020. Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2020).
[49] Science Buddies. 2013. Stepping Science: Estimating Someone's Height from Their Walk. https://www.scientificamerican.com/article/bring-science-home-estimating-height-walk/. Accessed: 2021-11-5.
[50] Scott Stein. 2021. Lidar is one of the iPhone and iPad's coolest tricks. Here's what else it can do. https://www.cnet.com/tech/mobile/lidar-is-one-of-the-iphone-ipad-coolest-tricks-its-only-getting-better/. Accessed: 2021-11-5.
[51] Gowri Somanath and Daniel Kurz. 2021. HDR Environment Map Estimation for Real-Time Augmented Reality. *CVPR* (2021).
[52] Shuran Song and Thomas Funkhouser. 2019. Neural Illumination: Lighting Prediction for Indoor Environments. *CVPR* (2019).
[53] Pratul P. Srinivasan, Ben Mildenhall, Matthew Tancik, Jonathan T. Barron, Richard Tucker, and Noah Snavely. 2020. Lighthouse: Predicting Lighting Volumes for Spatially-Coherent Illumination. In *CVPR*. 8077–8086. https://doi.org/10.1109/CVPR42600.2020.00810
[54] Tiancheng Sun, Jonathan T Barron, Yun-Ta Tsai, Zexiang Xu, Xueming Yu, Graham Fyffe, Christoph Rhemann, Jay Busch, Paul Debevec, and Ravi Ramamoorthi. 2019. Single image portrait relighting. *ACM Trans. Graph.* 38, 4 (July 2019), 1–12.
[55] Three.js Organization. 2021. Three.js - JavaScript 3D library. https://threejs.org. Accessed: 2021-11-5.
[56] TornadoWeb. 2022. Tornado Web Server. https://www.tornadoweb.org/en/stable/.
[57] Mihran Tuceryan et al. 2019. AR360: dynamic illumination for augmented reality with real-time interaction. In *2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT)*. IEEE, 170–174.
[58] Jonas Unger, Joel Kronander, Per Larsson, Stefan Gustavson, and Anders Ynnerman. 2013. Temporally and spatially varying image based lighting using HDR-video. In *21st European Signal Processing Conference (EUSIPCO 2013)*. IEEE, Marrakech, Morocco, 1–5.
[59] Unity. 2020. AR Foundation 4.2.0-preview.5. https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.2/manual/index.html.
[60] Unity3D. 2022. Unity3D. https://docs.unity3d.com/. Accessed: 2022-5-14.
[61] Jingao Xu, Guoxuan Chi, Zheng Yang, Danyang Li, Qian Zhang, Qiang Ma, and Xin Miao. 2021. FollowUpAR: enabling follow-up effects in mobile AR applications. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services* (Virtual Event, Wisconsin) *(MobiSys '21)*. Association for Computing Machinery, New York, NY, USA, 1–13.
[62] Juheon Yi and Youngki Lee. 2020. Heimdall: mobile GPU coordination platform for augmented reality applications. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking* (London, United Kingdom) *(MobiCom '20, Article 35)*.

Association for Computing Machinery, New York, NY, USA, 1–14.

[63] Yunfan Zhang, Tim Scargill, Ashutosh Vaishnav, Gopika Premsankar, Mario Di Francesco, and Maria Gorlatova. 2022. InDepth: Real-Time Depth Inpainting for Mobile Augmented Reality. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 6, 1, Article 37 (mar 2022), 25 pages. https://doi.org/10.1145/3517260

[64] Yiqin Zhao and Tian Guo. 2020. PointAR: Efficient Lighting Estimation for Mobile Augmented Reality. In *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Cham, 678–693.

[65] Yiqin Zhao and Tian Guo. 2021. Xihe: A 3D Vision-Based Lighting Estimation Framework for Mobile Augmented Reality. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '21)*. 28–40.