Xihe: A 3D Vision-based Lighting Estimation Framework for Mobile Augmented Reality

Yiqin Zhao
Worcester Polytechnic Institute
yzhao11@wpi.edu

Tian Guo
Worcester Polytechnic Institute
tian@wpi.edu

ABSTRACT

Omnidirectional lighting provides the foundation for achieving spatially-variant photorealistic 3D rendering, a desirable property for mobile augmented reality applications. However, in practice, estimating omnidirectional lighting can be challenging due to limitations such as partial panoramas of the rendering positions, and the inherent environment lighting and mobile user dynamics. A new opportunity arises recently with the advancements in mobile 3D vision, including built-in high-accuracy depth sensors and deep learning-powered algorithms, which provide the means to better sense and understand the physical surroundings. Centering the key idea of 3D vision, in this work, we design an edge-assisted framework called Xihe to provide mobile AR applications the ability to obtain accurate omnidirectional lighting estimation in real time.

Specifically, we develop a novel sampling technique that efficiently compresses the raw point cloud input generated at the mobile device. This technique is derived based on our empirical analysis of a recent 3D indoor dataset and plays a key role in our 3D vision-based lighting estimator pipeline design. To achieve the real-time goal, we develop a tailored GPU pipeline for on-device point cloud processing and use an encoding technique that reduces network transmitted bytes. Finally, we present an adaptive triggering strategy that allows Xihe to skip unnecessary lighting estimations and a practical way to provide temporal coherent rendering integration with the mobile AR ecosystem. We evaluate both the lighting estimation accuracy and time of Xihe using a reference mobile application developed with Xihe's APIs. Our results show that Xihe takes as fast as 20.67ms per lighting estimation and achieves 9.4% better estimation accuracy than a state-of-the-art neural network.

CCS CONCEPTS

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

KEYWORDS

mobile augmented reality; lighting estimation; 3D vision; deep learning; edge inference

MobiSys '21, June 24–July 2, 2021, Virtual, WI, USA

ACM ISBN 978-1-4503-8443-8/21/06…$15.00
https://doi.org/10.1145/3458864.3467886

ACM Reference Format:
Yiqin Zhao and Tian Guo. 2021. Xihe: A 3D Vision-based Lighting Estimation Framework for Mobile Augmented Reality. In *The 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '21), June 24–July 2, 2021, Virtual, WI, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3458864.3467886

1 INTRODUCTION

Augmented reality (AR), overlaying virtual objects in the user's physical surrounding, has the promise to transform many aspects of our lives, including tourism, education, and online shopping [10, 15]. The key for AR to success in these application domains heavily relies on the ability of *photorealistic rendering*, a feature which can be achieved with access to omnidirecitional lighting information at rendering positions [6]. For example, a virtual table should ideally be rendered differently depending on the user-specified rendering positions—referred to as *spatially-variant rendering*, to more accurately reflect the environment lighting and more seamlessly blending the virtual and physical worlds.

However, obtaining such lighting information necessary for spatially-variant photorealistic rendering is challenging in mobile devices. Specifically, even high-end mobile devices such as iPhone 12 lack access to 360° panorama of the *rendering position*. Even though with explicit user cooperation, it is possible to obtain the 360° panorama of the *observation position* via the use of ambient light sensors and front-/rear-facing cameras. Directly using the lighting information at the observation position, i.e., where the user is at, to approximate the lighting at the rendering position, i.e., where the virtual object will be placed, can lead to undesirable visual effects due to the inherent lighting spatial variation [7].

One promising way to provide accurate omnidirectional lighting information to mobile AR applications is via 3D vision support. With the recent advancement in mobile 3D vision including built-in high-accuracy Lidar sensors [14] and low-complexity high-accuracy deep learning models [23, 32, 33], we are bestowed upon a new opportunity to provide spatially-variant photorealistic rendering! In this work, we design the first 3D-vision based framework Xihe that provides mobile AR applications the ability to *obtaining accurate omnidirectional lighting estimation in realtime*. Our design can be broadly categorized into three parts: *(i)* algorithm and system design to support spatially-variant estimation; *(ii)* per-frame performance optimization to achieve the real-time goal; and *(iii)* multi-frame practical optimization to further reduce network cost and to integrate with existing rendering engines for temporal coherent rendering. We implement the framework Xihe on top of Unity3D and AR Foundation as well as a proof-of-concept reference AR application that utilizes Xihe's APIs. Figure 1 compares the rendered AR scenes using Xihe and prior work [21].

(a) Left: ARKit vs. right: GLEAM (obtained directly from [21])

(b) Left: ARKit vs. right: Xihe

Figure 1: *Rendered AR scenes.* Both GLEAM and Xihe achieve better visual effect compared to ARKit. Xihe better captures the spatially-variant lighting difference without needing the physical probe, compared to GLEAM [21]. Note we compared to ARKit's ambient light sensor based lighting estimation.

To support the key goal of spatially-variant lighting estimation, we design an end-to-end pipeline for 3D data processing, understanding, and management. Specifically, we devise a novel sampling technique called *unit sphere-based point cloud* technique to preprocess raw 3D data in the format of point cloud. This technique is derived based on our empirical analysis using a recent 3D indoor dataset [33]; our analysis shows the correlation between the incomplete observation data (i.e., not 360° panorama) and the lighting estimation accuracy. Further, we redesign a recently proposed 3D vision-based lighting estimation pipeline [33] by leveraging our unit sphere-based point cloud sampling technique to transform raw point clouds to compact representations while preserving the observation completeness. To better support mobile devices of heterogeneous capacity and simplify the client design, we centralize the tasks, including point cloud and lighting inference management, into a stateful server design. Our edge-assisted design also facilitates sharing among different mobile users and therefore provides opportunities to improve lighting estimation with *extrapolated point cloud data*, e.g., via merging and stitching different observation data to increase the completeness.

To achieve the real-time goal, we develop a tailored GPU pipeline for processing point clouds on the mobile device and use an encoding technique that reduces network transmitted bytes. Specifically, we leverage the property of point cloud in which the computation for each point can be parallelized and devise a strategy that trade-offs storage to improve runtime performance. In essence, we pre-generate densely sampled sphere coordinates in the unit sphere-based point cloud and pre-compute their distances to *anchors*—a set of uniformly distributed surface points in a unit sphere. At runtime, instead of trying to search for the closest anchor to each projected point, we simply search within the densely sampled sphere coordinates. We refer to these densely sampled sphere coordinates as *acceleration grids*. Further, we encode each colored anchor of the unit sphere-based point cloud with unsigned 8bits int and half-precision 16bits float where appropriate based on the practical characteristics such as common image format of LDR and the depth sensor precision. Our unit sphere-based point cloud design also allows easy stripping of unnecessary data by removing any uninitialized anchors ,i.e., anchors that are not colored due to incomplete raw observation point cloud.

Finally, we present an adaptive triggering strategy that allows Xihe to skip unnecessary lighting estimations and a practical way to provide temporal coherent rendering integration with the AR ecosystem. The key idea of the triggering strategy is to leverage an easy-to-obtain and fast-to-compute metric to determine directly on the mobile device whether the lighting condition has changed sufficiently to warrant a new lighting estimation at the edge. We use a sliding window-based approach that compares the unit sphere-based point cloud changes between consecutive frames. To achieve temporal-coherent visual effects, we leverage additional mobile sensors including ambient lighting and gyroscope to better match the lighting estimation responses with the current physical surroundings. We also detail steps to leverage a popular rendering engine to apply the spatially-variant lighting on virtual objects.

Spatially-variant lighting information can be traditionally extracted using physical probes [6, 21], and more recently estimated with deep neural networks [7, 8, 28, 33]. For example, Debevec et al. demonstrated that spatially-variant lighting can be effectively estimated by using reflective sphere light probes to extrapolate camera views. More recently, Prakash et al. developed a mobile framework that provides real-time lighting estimation using physical probes [21]. On a different vein, new deep learning-based approaches that do not require the use of physical probes have demonstrated efficiency in estimating spatially-variant lighting. The early efforts mostly focus on model innovation but still incur high computational complexity, making them ill-suited to run on mobile devices [7, 8, 28]. Until very recently, Zhao et al. proposed a lightweight 3D vision-based approach that takes advantage of new mobile depth sensors and shows promise of being mobile-friendly [33]. Our work leverages the advancement of mobile 3D vision and presents the first framework that supports accurate omnidirectional lighting estimation in real time via algorithm and system co-design. Moreover, our work does not require the use of physical reflective probes at runtime, thus can support a wider range and more practical AR application scenarios.

Our main contributions of this paper are:

- We design and implement a 3D vision-based framework Xihe that allows mobile AR applications to obtain spatially-variant lighting estimation and to achieve temporal-coherent rendering, fast and accurately. The relevant research artifact is available at: https://github.com/cake-lab/Xihe.
- We propose a novel point cloud sampling technique that effectively compresses the observation point cloud without impacting the estimation accuracy. This sampling technique is used in conjunction with a lightweight neural network to provide the spatially-variant lighting estimation.
- To achieve the real-time goal, we propose two per-frame optimizations, namely a tailored GPU pipeline for point cloud operations on the mobile devices and a practical data encoding scheme. This allows Xihe generate lighting estimations as fast as 20.67ms. We

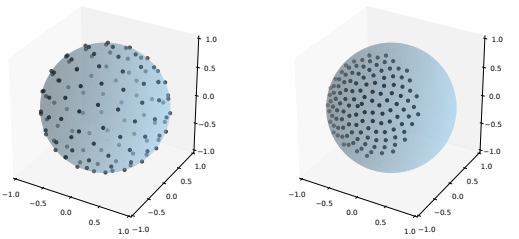
design an adaptive triggering strategy that effectively reduces up to 76.24% estimation requests by allowing Xihe client to identify lighting condition changes directly on the mobile devices.
- We conduct a comprehensive evaluation with a real-world 3D indoor RGB-D dataset on several mobile devices and network conditions and show that Xihe can achieve better visual effects than two existing approaches, i.e., GLEAM [21] and ARKit [13]. Further, we also present a detailed performance breakdown of Xihe under different configurations and use cases, reporting mobile, network, and edge time. Our study reveals a number of important factors such as number of anchors and estimation positions. Lastly, our lab-based evaluation showcases Xihe's ability to effectively generate lighting estimations by adapting to both environment lighting and user movement dynamics.

2 PROBLEM AND SOLUTION OVERVIEW

In this paper, we set out to address the key problem of providing spatially-variant photorealistic rendering in the context of mobile AR. *Photorealistic rendering* at a high level is about displaying life-like virtual 3D objects which mobile users cannot easily distinguish from physical objects. The key challenge to achieving photorealistic rendering lies in obtaining the omnidirectional lighting of a geometric space where the virtual objects will be displayed. Larger objects therefore require more lighting information at different points in the geometric space. The geometric center of the virtual object, referred to as the *placement position*, is often assumed to be specified by the user at runtime. A placement position can be extrapolated to multiple *estimation positions* where each position corresponds to a lighting representation, e.g., SH coefficients. The advent of *3D vision*, the ability to perceive both color and depth information, creates a new opportunity to transfer the measurable lighting information at the observation position to the placement position i.e. rendering positions.

Challenges. We leverage the key idea of 3D vision, and address three key challenges in designing a 3D vision-based lighting estimation framework called Xihe. The first challenge lies in accurately representing the spatially-variant lighting given the inherent constraints of mobile AR scenarios including limited field-of-view, user mobility, and environment lighting changes. The second challenge is to provide such accurate lighting estimation fast enough so that rendering engines can utilize this information for each frame if necessary. The third challenge is to provide temporal-coherent rendering that considers cross-frame visual harmony when utilizing estimated lighting information.

Solution Overview. In this work, we design a 3D-vision based framework Xihe that provides mobile AR applications the ability to *obtaining accurate omnidirectional lighting estimation in real time*. See Figure 6 for an architecture design of Xihe. Our design can be broadly categorized into three intertwined parts. We first introduce the algorithm and system design to support spatially-variant estimation in Section 3 and then describe our per-frame performance optimization to achieve real-time goal in Section 4.1. We further describe our cross-frame practical optimization to reduce network cost and to integrate with a popular rendering engine for temporal-coherent rendering in Section 4.2.



(a) Uniform distribution. **(b)** Non-uniform distribution.

Figure 2: *Impact of anchor point distributions on observation completeness.*

3 SPATIALLY-VARIANT ESTIMATION

Spatially-variant lighting allows representation of lighting at different world positions. As such it has the promise to provide more photorealistic renderings of virtual objects, which is especially important in the realm of augmented reality. For example, when used in a furniture shopping app, spatially-variant lighting can more effectively render a piece of couch with different outlooks depending on the user's physical environment (well-lit room or darker room), the rendering position, and the couch size. Figure 1 visualizes rendering examples with and without spatially-variant lighting information.

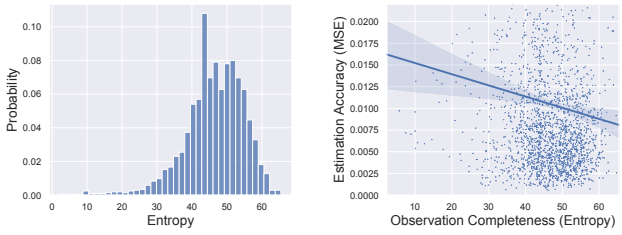
3.1 Unit-Sphere based Point Cloud Sampling

Xihe enables real-time efficient lighting estimation for mobile devices with 3D vision-based deep learning model that takes point cloud generated from on-device camera captured RGB-D image as input. Such point cloud data is usually large in size and can contain redundant information [27]. Therefore, down-sampling point cloud is beneficial to computation and network efficiency. However, directly down-sampling the raw point cloud using techniques such as uniform sampling can negatively impact lighting estimation accuracy. In this section, we present our novel unit sphere-based point cloud sampling technique which preserves observation field-of-view (FoV) as much as possible. Our design is informed by an empirical analysis that demonstrates the negative correlation between observation completeness and lighting estimation accuracy.

3.1.1 Impact of Incomplete Observation Data. To study the potential impact of incomplete observation data on lighting estimation accuracy, we first propose an entropy-based metric to measure the point cloud observation completeness. We define the *observation completeness* as the percentage of colored anchors when projecting a point cloud to unit sphere surface. Further, *anchor points* are defined as a set of uniformly distributed surface points in a unit sphere O. Intuitively, the observation completeness depends on both the points distribution (D) and anchor distribution (A). We define the joint entropy $H(D, A)$ as:

$$\mathrm{H}(D, A) = -\sum_{i \in S} \sum_{j=1}^{i} \mathrm{P}(d_{ij}, a_i) \log_2[\mathrm{P}(d_{ij}, a_i)]. \quad (1)$$

where $P(d_{ij}, a_i)$ is the joint probability of projecting points into a unit sphere with i anchor point, and S is a set of possible anchor



(a) Entropy distribution. **(b)** Entropy vs. accuracy.

Figure 3: *Observation-preserving metric empirical analysis.*

sizes (i.e., number of anchor points). In this work, we choose $S = \{2^k | 1 \leq k \leq 12\}$. By using the Equation (1), we can succinctly measure the point cloud observation completeness. The higher the entropy value, the more complete the observation. Additionally, it allows us to easily distinguish the observation completeness for point clouds of the same size. For example, even though the two projected point clouds have the same number of points, the projected point cloud shown in Figure 2(a) is considered to be more complete than the one shown in Figure 2(b).

Next, we leverage a recently proposed state-of-the-art 3D vision based lighting estimation algorithm and its point cloud dataset [33] to correlate observation completeness with estimation accuracy. The raw point clouds, each has around 82K points, were first uniformly downsampled to 1280 points. We then trained a model (see Section 6.5 for training setup details) based on the original paper's description. Lastly, we obtained the lighting estimation error, represented as Mean Squared Error (MSE), by evaluating the trained model on each point cloud and compare the results to ground truth.

Figure 3(a) depicts the entropy distribution of all raw point clouds. Figure 3(b) shows a negative correlation between the observation completeness measured by e and estimation accuracy. In conclusion, we empirically observed that the coverage observation completeness is the key impact factor to lighting estimation accuracy.

3.1.2 Observation Point Cloud Downsampling. To effectively sample point cloud, we design a novel point cloud sampling method called unit sphere-based point cloud sampling. It first projects every point in a point cloud P onto a unit sphere O, defined as $f(P, O)$, then find the nearest anchor point for each projected point. By establishing the relationship between each point from P and O, f outputs a point cloud distribution D. Finally, we downsample P with a nearest point selection that approximates the depth culling process [1]. That is, P is downsampled by selecting the nearest points, i.e., points with the shortest 3D Euclidean distance to the sphere origin, from D to color corresponding anchor points in O.

Figure 4 provides an example walkthrough of our unit sphere-based point cloud sampling. Considering three points (p_1, p_2, p_3) in the original point cloud, and their corresponding projected points (p'_1, p'_2, p'_3). Each projected point is then matched to an anchor that is closest to itself. We refer to these matched anchors as *nearest neighbors*. In this example, p'_1 will be matched to a_1 while p'_2 and p'_3 will both be matched to a_2. Without loss of generality, if p'_1 is the *only* projected point that a_1 is assigned as the nearest neighbor, then a_1 will be initialized with the RGB values of p'_1 and $D(p'_1, o)$—the distance between p'_1 and the sphere origin o. Similarly, if p'_2 and p'_3 are the only projected points that a_2 is assigned, and p'_2 is closer to the sphere origin than p'_3 is, a_1 will be initialized with the RGB values of p'_2 and $D(p'_2, o)$.

In practice, the points set P is determined at runtime by configurations and camera hardware while the size of O is a configurable system parameter; the ratio $\frac{size(P)}{size(O)}$ represents the down-sampling ratio. In other words, we will leverage the unit sphere-based point cloud as the basis for estimating the spatially-variant lighting information instead of directly using the observed point cloud. In an AR session, we need to perform several consecutive unit sphere-based point cloud sampling within a small time span for each estimation position, as will be described in section 4.2.1. The unit sphere-based point cloud for each estimation position will likely only be partially initialized, due to incomplete environment observation.

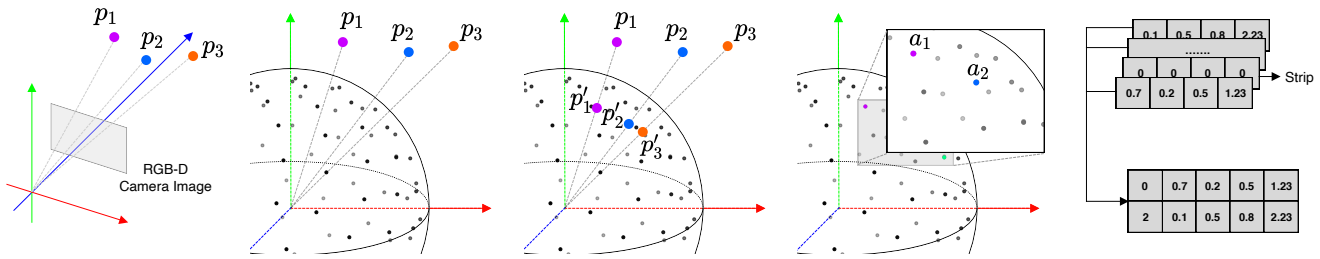
We will store the unit sphere-based point cloud in a custom designed data structure, represented as a 4D vector (the RGB values and the 3D Euclidean distance between the projected points and the sphere origin). This data structure design has two major advantages. First, sphere anchor positions can be pre-computed ahead of time. As such, our data structure only needs to store anchors in an ordered array with each index corresponds to an anchor position. This design also presents an opportunity to speedup Xihe's triggering strategy using pre-computed neighbors for each anchor, as will be described in Section 4.2.1. Second, storing unit sphere-based point cloud using this data structure allows Xihe to extract both 3D space positions and colors at viewing directions for estimation positions.

3.1.3 Downsampled Point Clouds for Virtual Objects. Xihe supports estimation positions that are specified via Xihe's APIs and can perform unit sphere-based point cloud sampling at each estimation position. Additionally, Xihe also provides a simplified workflow that automatically assigns estimation positions when a virtual object is placed to the scene. Specifically, given a virtual object's placement position (e.g., specified by the user), Xihe will first designate the placement position as one estimation position and subsequently generate multiple estimation positions based on the object size. In other words, Xihe can support multiple lighting estimation requests, the response of each is represented as SH coefficients, for a given object. However, due to current mobile rendering engine limitations, e.g., in the case of mobile Unity3D, Xihe will only render each object with one set of SH coefficients. To circumvent this limitation, AR developers using Xihe can either decompose the large object into smaller ones or customize rendering shaders to take advantage of multiple sets of SH coefficients.

3.2 3D Vision-based Estimation Pipeline

Another key design to support spatially-variant lighting comes down to an efficient algorithm that can extract and estimate lighting information from an incomplete environment point cloud. As depth sensors such as Lidar start to be equipped with mobile devices, it is now possible to leverage 3D vision-based algorithms for Mobile AR applications.

However, given that mobile 3D vision is in its infancy, there have been very few works in mobile-friendly 3D vision-based lighting estimation techniques [26, 33]. We choose the state-of-the-art 3D vision based lighting estimation model PointAR [33] as the building block for designing a new lighting estimation neural network that works well with unit sphere-based point cloud. Briefly, PointAR is



(a) Camera input. **(b)** Raw points. **(c)** Point projection. **(d)** Anchor initialization. **(e)** Encoding.

Figure 4: *An example of unit sphere-based point cloud sampling and encoding.*

a two-staged neural network pipeline that consists of: *(i)* a point cloud transformation to simulate the camera movement from the observation position to the rendering position; *(ii)* a point cloud-based compact deep learning model. Our practical XiheNet is designed by integrating our novel unit sphere-based point cloud sampling technique, as described in Section 3.1.2, into the first stage.

To train the XiheNet model, we first generate six training/test datasets—with the following number of anchors: [512, 768, 1024, 1280, 1536, 2048]—each consists of 608k/2037 unit sphere-based point cloud from the PointAR RGB-D dataset. Then, we extract the ground truth lighting information, represented as SH coefficients from both LDR and HDR environment maps. The resulting data item is in the form of a downsampled point cloud and the corresponding SH coefficients. For LDR-based and HDR-based datasets, we train six instances of XiheNet to study the performance and estimation accuracy trade-offs. As we later observe that XiheNet models trained with LDR-based datasets lead to better visual effects than that of HDR-based datasets; for the remainder of the paper, we will report results using XiheNet trained on LDR-based datasets.

Xihe outputs SH coefficients as an omnidirectional representation of environment lighting at a single world position for rendering. If directly using image-based lighting estimation models [8, 28], one needs to post-process to correctly orient estimated SH coefficients since the 3D world orientation cannot be represented on the image input. Our XiheNet guarantees the orientation constant [33] by explicitly considers the world space point cloud and estimates SH coefficients at the same orientation.

3.3 Edge-assisted Resource Sharing

3.3.1 Point Cloud Management. Naively preserving and managing time-series downsampled point clouds on mobile devices can be harmful to overall system performance as it can consume too much device memory. Therefore, Xihe proposes to use a stateful edge-based point cloud management design. Moreover, we abstract Xihe client as a point cloud provider to handle mobile heterogeneity, like camera parameters, through adapters.

We design the Xihe server to manage the unit sphere-based point cloud for each estimation position. Edge data will be updated throughout the AR session as new lighting estimation requests come in. Currently our XiheNet can produce highly accurate estimations even with per-frame data that have a number of uninitialized anchors. Based on our empirical analysis between observation completeness and the estimation accuracy, one practical way to further improve the estimation accuracy is to leverage multiple

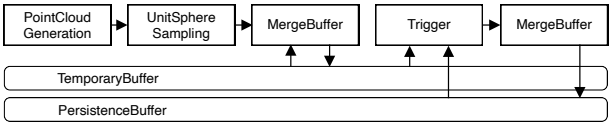


Figure 5: *Xihe mobile GPU processing pipeline.*

frames to progressively complete the unit sphere-based point cloud, i.e., with more colored anchors. Thus, Xihe server manages the collected environment point cloud in an accumulative fashion and merges the point cloud with data associated with newly triggered requests. As such, we can then augment the unit sphere-based point cloud sent with each estimation request with historical data for improving observation completeness. Such augmentation is particularly useful when environment observation data is shared among multiple estimation positions or clients in the same AR session.

3.3.2 Lighting Estimation Management. Lighting estimation is inherently latency sensitive due to the strict latency requirement. In the context of mobile AR, rendering engine typically targets 30fps refresh rate, which corresponds to refresh approximately 33.33ms per frame. Ideally lighting estimation should be performed at each frame to achieve best accuracy. However, for deep learning-based estimation algorithms, fulfilling such requirement on mobile devices can be very challenging given the resource constraints and the likelihood of sharing on-device computation resources with tasks like rendering.

To address the low latency requirement and better support our overall vision of supporting multi-users AR sessions and multi-objects rendering, we design Xihe to run the inference execution on a GPU-accelerated edge server. A strategically provisioned edge can provide a low-latency and high-bandwidth connection between the mobile devices and the server [20], as well as the potential to batch inference requests. By using edge-based deep learning inference, Xihe has the potential to support more complex model and achieve much lower computation latency, e.g., by leveraging the powerful edge resources. The Xihe server will use the merged point cloud as the input to the 3D vision-based estimator and send back the estimation result in SH coefficients to Xihe client.

4 FAST AND ACCURATE ESTIMATION

Continuously processing point cloud data directly on mobile CPU can be time consuming, thus might violating the real-time goal. For example, a RGB-D image captured on a iPad Pro with the max resolution of 256x192 corresponds to 49k raw points. Fortunately,

operations on point clouds can naturally be parallelized at per-point level. We explicitly assemble as many operations as possible to run on mobile GPU to reduce the CPU-GPU communication overhead for achieving the best performance; the processing pipeline is shown in Figure 5. Further, as constantly sending all the RGB-D data over the network to the edge can be costly and might not be necessary, we devise an adaptive triggering strategy that skips inference requests if the environment lighting conditions do not change substantially. The triggering component (see Section 4.2.1) decides whether Xihe should offload the point cloud sample to the edge sever for inference or store it into a temporary buffer.

4.1 Per-frame Real-time Optimization

4.1.1 Accelerating Point Cloud Sampling. Our processing pipeline, as shown in Figure 5, starts with generating point cloud data from a camera captured RGB-D image feed, and performs generation at a pre-configured refresh rate. By leveraging GPU computing, our point cloud generation code can be fully parallelized for common RGB-D image resolutions, e.g., 256x192 in the case of Lidar-equipped iPad. Each generation outputs a point cloud of the world environment that is captured in the current camera view.

Then, we apply unit sphere-based point cloud sampling to the generated point cloud. However, naively using this sampling technique is computationally intensive, as it runs in $\Theta(\text{size}(P)*\text{size}(O))$ time. Although it is possible to parallelize both point processing and anchor searching, the number of required GPU threads can exceed the maximum support. For example, if we were to fully parallelize the sampling process, i.e., using one GPU thread for searching each point-anchor pair, on a point cloud with 49k points and 1280 anchors, we will need about 62M GPU threads, while the maximum supported GPU thread group size on mobile Unity3D platform is 65535. Although partially parallelizing the sampling, e.g., running the neareast anchor search for each point on a GPU thread, may comply with the current mobile GPU requirement, the execution time of each thread will be elongated. Therefore, we propose an optimization method for reducing the computation resource requirement of this sampling technique.

Specifically, we propose to build a 2D *acceleration grid* that uses densely pre-sampled points and pre-computed results to reduce the neareast anchor searching time. Specifically, we first densely sample a set of points on the unit sphere based on quantizing the spherical coordinates polar angels θ and ϕ. Then, we pre-compute the nearest anchor for all the densely sampled points and store the corresponding anchor index in the acceleration grid. At runtime, for each projected point, we first convert its cartesian coordinate to spherical coordinate and then apply the same quantization to match the point to a densely sampled point in the grid. The key advantage of doing so is that pre-sampled points can be stored as a 2D array, and indexed with spherical coordinates at runtime cheaply.

Note that using our proposed acceleration grid may introduce sampling errors as in essence this approach presents the entire sphere surface with discrete sampled points. Intuitively, the more points the grid has, the better the approximation. We empirically show that given a unit sphere-based point cloud with 1280 anchors, using a pre-computed grid of 1024x512 points allow 97% projected points match to their neareast anchors and only incur a negligible estimation error. More details will be presented in Section 6.3. After a new unit sphere-based point cloud data is sampled, Xihe client will continue the GPU pipeline execution by merging the newly generated data with historical data in the temporary buffer using an extrapolation operation. The merging operation is an anchor-wise copy operation between two buffers, and will always overwrite old data with new one.

An alternative approach to accelerate unit sphere-based point cloud sampling is to build a bounding volume hierarchy—a technique to accelerate ray tracing in real-time rendering [11]—by properly subdivising the sphere surface to reduce the search space. However, leveraging such subdivision methods is nontrivial as one has to balance a number of factors such as the sphere surface dividing time, division access time, and total search time. We leave this exploration as part of future work.

4.1.2 Unit Sphere-based Point Cloud Encoding. Xihe promises the low-latency network communication by leveraging above-mentioned compact point cloud data structure with byte-optimized encoding method. When an estimation request is triggered, Xihe client sends the corresponding encoded unit sphere-based point cloud as a byte-encoded HTTP packet to the Xihe server. Our encoding consists of two steps: the striping step which removes all uninitialized anchors from the unit sphere-based point cloud and the byte representation step which stores each point with fewer bits.

Specifically, instead of storing each point of the downsampled point cloud with four 32bits single precision floats, we use 8bits unsigned int and 16bits half-precision float to represent each point. Though the original format is more precise to calculate and performs better as it aligns to the GPU bus transaction size, such data format uses redundant data bytes. For example, when dealing with low dynamic range (LDR) camera images, 8bits data is usually sufficient to preserve the useful information. Also, due to limitations such as depth sensor precision and camera visible area size, the distance information can be presented with 16bits half precision float. Lastly, for each colored point, we use an extra 16bits to store their indices.

Our encoding scheme can lead to significantly savings both in terms of per-request and per-pixel data size. For example, for a unit sphere-based point cloud generated from a LDR camera image, using Xihe to encode the request data only needs 7 bytes, about 43.75% of the size if we encode with the original four single precision floats. Comparing to directly sending the raw RGB-D image (5 bytes per pixel), Xihe reduces approximately 98.3% data usage by only needing to send on average 4249 bytes for an RGB-D image with 256x192 resolution. This results in both less network data transfer and potentially less network time.

4.2 Cross-frame Optimization

4.2.1 Adaptive Estimation Triggering. Most modern camera systems provide high refresh rate, e.g. 30fps or higher. Estimating scene lighting at the same frequency for each frame can be beneficial for achieving good visual results, but also consume significant computation and energy resources. Additionally, it might not be necessary to update lighting information this frequently as environment lighting conditions might not change at this rate. To avoid sending unnecessary estimation requests to the edge, we design a

triggering strategy that allows Xihe client to efficiently compare lighting changes on mobile devices.

Designing effective triggering strategies involve addressing two major challenges: *(i)* potential camera movements between consecutive frames; *(ii)* low latency requirement. The first challenge indicates that image difference-based triggering methods are less robust as camera movements can lead to mismatched camera images between frames. Although it is possible to leverage techniques that stitch consecutive frames, such techniques are likely to violate the latency requirement outlined in the second challenge. As such, we design a unit sphere-based point cloud-based triggering strategy which is less sensitive to observation point cloud changes and can be integrated as a part of our mobile GPU processing pipeline to satisfy the real-time goal.

The triggering decision making, in essence, evaluates the unit sphere-based point cloud differences between frames and decides triggering based on the amount of difference. Specifically, Xihe makes the triggering decision by: *(i)* calculating the anchor-wise color difference (i.e., MSE of two RGB values) between two unit sphere-based point clouds that are stored in the temporary buffer and the persistent buffer, respectively; *(ii)* obtaining the pooling averages using a sliding window of size N (i.e., N anchors) on the sphere for each anchor; *(iii)* triggering estimation when any pooled value exceeds a threshold value θ. Both the threshold and the number of nearest neighbors can be configured empirically and we set the threshold value $\theta = 0.6$ and $N = 4$ based on our analysis in Section 4.2.1. If Xihe client decides to trigger the lighting estimation, we will continue the pipeline execution by merging the temporary buffer into the persistent buffer. Otherwise, we will early exit the GPU pipeline execution.

4.2.2 Providing Temporal-coherent Rendering. Lighting estimation can fall short in reflecting the physical world lighting at the exact moment. As we described in Section 4 and will show in Section 6, Xihe can achieve as fast as 20.67ms per lighting estimation request. This property well positions us to use a simple yet effective approach to achieve temporal-coherent rendering. To compensate the estimation delay, we utilize mobile ambient light sensor, which is cheap to use and provides low-latency ambient average color and intensity lighting data, to continuously adjust the rendered environment lighting per frame. Once the SH coefficients response is available on the mobile device, Xihe will apply it to re-lit the virtual object. As such, our compensation technique can improve the visual coherence when the environment lighting is changing very rapidly and handle use cases with less ideal network conditions.

To account for user movement during a single AR session, Xihe leverages mobile device's built-in accelerometer and gyroscope to obtain the camera position and orientation information. This allows Xihe to track estimation positions (i.e., represented as world coordinates relative to the origin coordinate) and distinguish them as active and inactive positions; an inactive position is one that is outside the current camera view. If all estimation positions are inactive, Xihe will not engage the GPU pipeline[1]; as estimation positions become active, Xihe will resume its normal operations. If Xihe client triggers the estimation and subsequently sends the unit sphere-based point cloud corresponding to the active estimation position to the edge, Xihe server can opportunistically augment any managed unit sphere-based point cloud at the edge.

[1] An alternative is to leverage the triggering algorithm to proactively send unit sphere-based point cloud to the edge for point cloud augmentation.

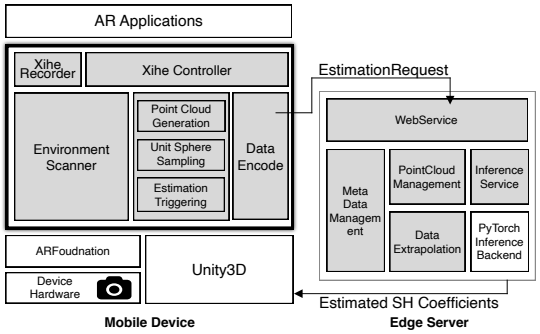


Figure 6: *Xihe architecture.*

Table 1: *Xihe key classes and functions.*

API	Meaning
`XiheController`	Main system controller
`EnvironmentScanner`	Supporting environment scan
`GPUDataProcessor`	Supporting point cloud processing
`InferenceBackend`	Provide lighting estimation inference
`XiheRecorder`	Provide AR session recording
`EstimateAt(p)`	Function to estimate lighting at given position
`PlaceAndEstimateAt(p)`	Function to simply place virtual object and estimate placement position lighting
`StartRecording`	Function to start AR session recording
`StopRecording`	Function to stop AR session recording

5 IMPLEMENTATION

We implement Xihe in two logical components: one runs on the mobile client side and the other as a managed edge service. Figure 6 depicts the architecture of Xihe. Our implementation consists of around 5K lines of code written in C#, Python, and CUDA C++, and works with commodity hardware and software frameworks. Specifically, Xihe client is built on top of AR Foundation 4.2.0 [30] which provides basic AR functionalities and works with rendering engine including Unity3D 2020. We design Xihe client to run on a wide range of mobile hardware. Xihe is developed as a Python-based web API server and uses PyTorch [22] backend with just-in-time model optimization to host our deep learning model XiheNet. The server is packaged as a Docker image to facilitate deployment.

Client. Xihe client is implemented as a C# library that runs on Unity3D. Table 1 summarizes the provided APIs. The mobile GPU pipeline, including *Point Cloud Generation*, *UnitSphere Sampling*, and *Estimation Triggering* components, is implemented as Unity3D compute shaders in HLSL. Developers can start using Xihe with both new and existing AR projects by initializing the *XiheController* at the start of application life cycle. This allows the applications to either create a new AR session or join an existing one. Developers may ignore the implementation of other internal components or can extend Xihe to support new features, e.g., using new camera hardware through overwriting the *AcquireEnvironmentScan* function in the *EnvironmentScanner* class. We provide *EstimateAt* and *PlaceAndEstimateAt*, two key functions to provide spatially-variant lighting estimation through the *XiheController*. The first estimation function

directly returns the estimated SH coefficients as an ordered array, and therefore can be used in any customized rendering pipelines or shaders. The second function works directly with Unity3D engine by supporting a simplified workflow of placing virtual objects in the format of Unity3D Prefabs into the physical surrounding. If sufficient lighting change is detected, both functions will trigger HTTP requests to send the encoded unit sphere-based point cloud to the server. Xihe also has a built-in AR session recorder that captures the essential AR session information, including lighting estimation position, RGB-D camera feed, camera pose and ambient light sensor data. This recorder is built on top of the same `EnvironmentScanner` used in Xihe's estimation workflow. We provide two APIs, `StartRecording` and `StopRecording`, to control the recording process.

Server. Xihe server is built on top the *Tornado* web framework and the PyTorch inference backend with just-in-time model optimization. Xihe server provides two key features, namely the *AR session management* and the *lighting estimation request processing*. The AR session management service is provided by three components, i.e., *MetaData Management*, *PointCloud Management*, and *Data Extrapolation*. Specifically, the *MetaData Management* is responsible for storing AR session's basic information, such as unique session ID issued by the server, client-specific identifier, and timestamp. To connect to the Xihe server, each Xihe client will either create a new AR session by posting requests or join an existing AR session by providing the session ID. Additionally, we use the *NumPy* library to perform tensor operations on HTTP payload in the byte format to achieve historical data extrapolation. A pretrained XiheNet (number of anchors = 1280) is included and managed by the PyTorch inference backend. Xihe server is packaged as a Docker image and can be setup with minimal configuration effort.

A reference AR application. We include an example AR application implemented with Xihe APIs and the ARKit V4.0 plugin from the ARFoundation. The resulting application can be compiled to run on iOS and macOS. This reference application allows a mobile user to place 3D virtual objects by selecting rendering positions in the current camera view. For each virtual object, the mobile application will generate one logical lighting estimation request per frame. Depending on the detected lighting conditions, Xihe client will trigger one or more physical lighting estimation requests which will send the encoded unit sphere-based point cloud to the Xihe server for inference. The number of physical lighting estimation requests per frame is by default depending on the object size but can also be configured by the mobile AR developers for performance and quality trade-offs. The 3D objects will be rendered with spatially-variant lighting information provided by Xihe. Lastly, users can easily record the AR session with Xihe's session recorder, facilitating real-world record-and-replay experiments.

6 EVALUATION

We conducted our experiments by using an example AR application which uses Xihe's APIs for obtaining spatially-variant lighting estimation. We used three different devices, a MacBook Pro 15", a 2nd generation iPad Pro 11" with a built-in Lidar sensor, and an iPhone 11 Pro, to evaluate the on-device performance. Our edge service is on a desktop running Ubuntu 20.04 with a 16 core AMD Ryzen Threadripper 2950X CPU, 128GB memory, and a Nvidia RTX 2080Ti GPU. We quantified Xihe's performance in terms of end-to-end lighting estimation time, accuracy, and visual effects and compared it to the commercial baseline ARKit [13], an academic framework GLEAM [21], and a 3D vision estimation pipeline [33] where appropriate. Xihe can deliver spatially-variant lighting estimation as fast as 20.67ms and achieves visually-coherent rendering effects. We further evaluate how each proposed technique and configuration contributes to Xihe's performance with a detailed breakdown study, e.g., with different sampling strategy, anchor size, and mobile network condition, and a lab-based real-world evaluation.

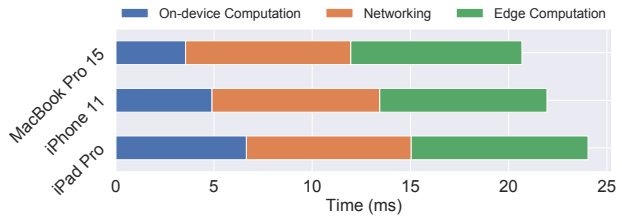


Figure 7: *Xihe end-to-end time.* Xihe lighting estimation via the university WiFi, can complete in as fast as 20.67ms.

Figure 8: *AR scenes rendered with lighting information provided by ARKit and our framework Xihe.*

6.1 End-to-end Performance

We demonstrate the end-to-end performance achieved by Xihe; Xihe takes less than 24.04ms to complete in all three devices with the university WiFi as shown in Figure 7. As such, Xihe not only can support 30fps refresh rate but also takes 19.9% less time than GLEAM [21]. The on-device GPU computation takes less than 6.65ms to finish running on all three devices. This indicates that Xihe can support a wide range of mobile devices. Figure 8 compares the visual effects of three 3D objects. Given the same virtual object and the same environment lighting condition, Xihe's reference AR application is able to display the virtual object in a photorealistic manner. However, when using the ARKit's ambient lighting estimation APIs, objects will be rendered with less desirable effect as only ambient lighting intensity and color information are available.

Table 2: *Mobile computation breakdown.* All components except the first and the last run in the GPU pipeline.

On-device Component	MacBook Pro Avg. Time (ms)	iPhone Avg. Time (ms)	iPad Pro Avg. Time (ms)
AcquireEnvironmentScan	N/A	N/A	1.458
GeneratePointCloud	0.063	0.193	0.221
UniSphereSampling	0.015	0.061	0.063
MergeTemporaryBuffer	0.007	0.024	0.029
MakeTriggeringDecision	1.34	2.185	3.040
MergePersistentBuffer	0.013	0.057	0.062
EncodeBuffer	2.16	2.332	2.832

Table 3: *Edge computation breakdown.*

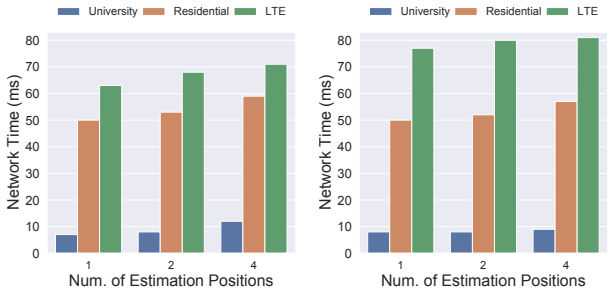
Edge Component	512 Anchors (ms)	1280 Anchors (ms)	2048 Anchors (ms)
Decoding	0.23	0.30	0.32
Extrapolation	0.01	0.01	0.01
Inference	3.99	5.91	10.80

6.2 Performance Breakdown

6.2.1 On-device performance. To quantify the performance of Xihe client, we measure the time breakdown of each component that runs on the mobile device. Table 2 shows the average time across five runs measured with the Unity3D built-in profiling tool [31]. For the MacBook Pro and iPhone measurement, since they do not have built-in Lidar sensors, we randomly selected five test data. First, we observe that the total on-device time excluding the *AcquireEnvironmentScan* step using MacBook Pro, iPhone and iPad Pro are 3.57ms, 4.90ms and 6.65ms, respectively. This is expected as the results matches the devices' GPU computation capabilities. Second, the long GPU time of two dominating components (i.e., `MakeTriggeringDecision` and `EncodeBuffer`) are likely due to the callback functions needed for communicating between CPU and GPU and non-continuous memory access during encoding.

6.2.2 Edge performance. To quantify the performance of Xihe server, we measure the time breakdown of each component that runs on a GPU-equipped desktop. Table 3 shows the average time across the entire test dataset. First, we observe that both the `decoding` and `inference` time increase with the number of anchors. This is expected as unit sphere-based point cloud with more anchors are likely to have more encoded points that need to be decoded and transformed. Second, the point cloud extrapolation only takes 0.01ms but provides the opportunity to improve the lighting estimation accuracy, i.e., by boosting the input point cloud's entropy with historical data.

6.2.3 Network performance. Figure 9 shows the network time under different network conditions and user interactions. If the user places larger objects, i.e., more positions to estimate, the network time increases sublinearly under all network conditions. For example, for iPad Pro that uses the university WiFi, the network time to handle four estimation positions is about 1.28X that of one estimation position. Both residential WiFi and LTE take several times longer than using the university WiFi, indicating the need to properly deploy the Xihe server. Even under undesirable network condition, e.g., iPhone with LTE, Xihe can still generate one lighting estimation in about 79.2ms which is lower than the 400ms needed by GLEAM to generate high-fidelity estimation [21].



(a) iPhone. **(b)** iPad Pro (LTE via hotspot).

Figure 9: *Xihe network time.* We measure the time needed to transfer the encoded point cloud of 1280 anchors and to receive the lighting estimation SH coefficients.

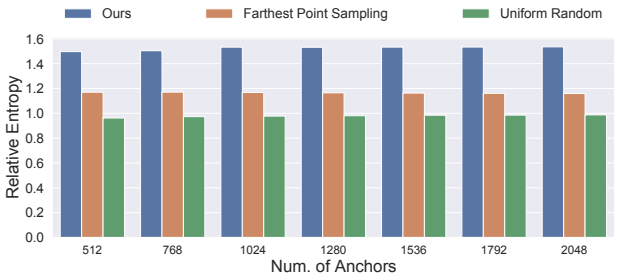
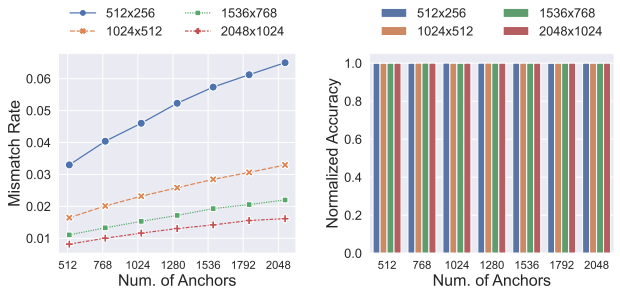


Figure 10: *Relative entropy comparison of different point cloud sampling techniques.* Our unit sphere-based point cloud sampling achieves 55.74% and 30.80% better entropy compared to uniform random sampling [33] and farthest point sampling techniques [24], respectively.

6.3 Impact of Point Cloud Sampling

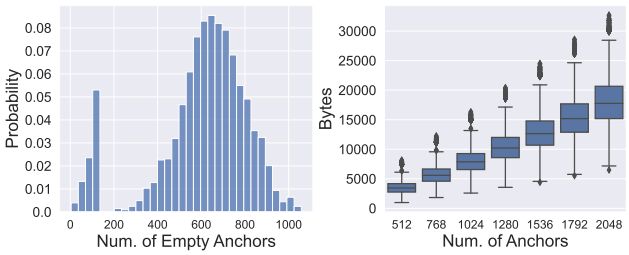
Entropy comparison. We use the point cloud test dataset from PointAR [33] and generate three variants using the *uniform random sampling* [33], *farthest point sampling* [24], and our proposed unit sphere-based point cloud sampling techniques. For each point cloud (and its downsampled versions), we first calculate the entropy using Equation (1) and divide it by the raw point cloud entropy. Figure 10 compares the relative entropy. First, our unit sphere-based point cloud sampling approach is more effective in preserving the entropy, with on average 0.545 higher relative entropy than using the uniform random sampling, and 0.359 higher than using the farthest point sampling. Second, using more anchors can improve the entropy but the improvement plateaus after 1280. This observation suggests that using 1280 anchors can be effective. Later in Section 6.5 we will compare and show that unit sphere-based point cloud technique also achieves better estimation accuracy.

Impact of acceleration grids. In this section, we analyze the error associated with the acceleration grid with the *mismatch rate* metric, calculated by comparing the colored anchors with and without acceleration. We first randomly generate a set of 1M points in a cubic 3D space with an edge length of 10 meters to simulate common AR application scenarios in real-world environments. Figure 11(a) shows the mismatch rate using different acceleration grid sizes. First, as the grid size increases (i.e., corresponding to pre-sampled more points), the mismatch rate decreases. For example, with an acceleration grid of 1024x512 and an anchor size of 1280, we observe that 97.36% of points were matched to the same anchor without



(a) Acceleration mismatch **(b)** Impact on estimation accuracy

Figure 11: *Impact of acceleration grids.* Even though acceleration grids might erroneously match a small percent of points, it has minimal impact on the estimation accuracy.



(a) Number of Anchors = 1280. **(b)** Non-empty anchors.

Figure 12: *Empirical characterization of unit sphere-based point cloud generated from the test dataset [33].*

Table 4: *Comparison of different encoding methods.*

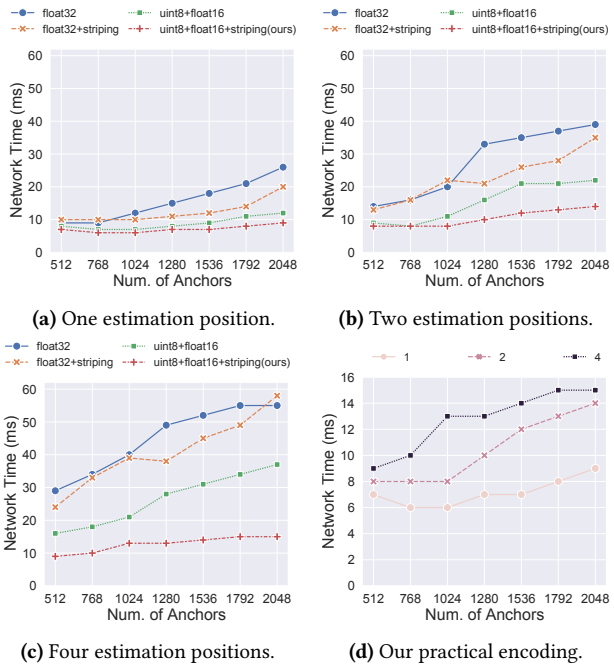
Encoding Method	Unit Size (bytes)	Avg. Size (bytes)	Avg. Time (ms)
`float32`	16	20480	0.003
`float32` + striping	18	10926	1.793
`uint8` + `float16`	5	8960	1.675
`uint8` + `float16` + striping (Ours)	7	4249	1.003

using acceleration. Second, for a given grid size, the mismatch rate also depends on and grows with the number of anchors. However, the impact is relatively small compared to the choice of acceleration grid size and is within 1% for the range of anchor numbers.

We further investigate the impact of our acceleration mechanism on the lighting estimation accuracy. Figure 11(b) shows the normalized accuracy, calculated by comparing the accuracy achieved using acceleration and the ground truth accuracy using XiheNet. We see that neither the acceleration grid size and the anchor number impact the lighting estimation accuracy. This also suggests that our XiheNet has a good generalization.

6.4 Performance of Encoding

Next, we evaluate the effectiveness of different encoding methods. Figure 12 presents the empirical characterization by performing the unit sphere-based point cloud sampling technique to the raw point clouds from the test dataset. We find that unit sphere-based point cloud generated from a single RGB-D camera image usually contains many uninitialized anchors (i.e., empty anchors), as shown in Figure 12(a). For example, when setting the number of anchors to be 1280, we observe that more than half anchors are empty.



(a) One estimation position. **(b)** Two estimation positions.

(c) Four estimation positions. **(d)** Our practical encoding.

Figure 13: *Network performance of different encoding methods.* The mobile AR application connects to the Xihe server via the university WiFi.

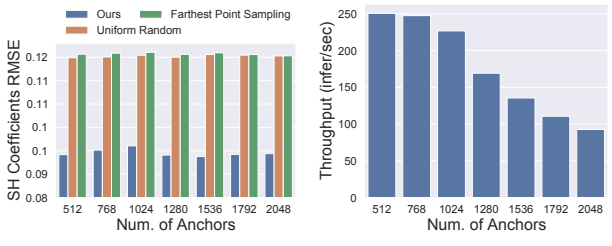
Figure 12(b) shows the required bytes for encoding only non-empty anchors using `float32`.

Table 4 compares the required bytes and time to encode our unit sphere-based point cloud using different methods. Even though directly encoding using `uint8` for RGB values and `float16` for depth information only requires 5 bytes per anchor, it takes more than twice as many bytes to transfer the entire unit sphere-based point cloud than our encoding approach. Further, as the striping operation is cheaper, taking about 0.09ms on the iPad Pro, our encoding method improves the total encoding time by 1.67X compared to directly encoding with `unit8+float16`.

Figure 13 shows the median network time to transfer the unit sphere-based point cloud from the MacBook Pro to the Xihe server via the university WiFi. Results for other devices and network conditions (residential WiFi and T-mobile LTE) exhibit similar trends and are omitted. First, we observe that our encoding method takes as little as 26.3% network time compared to three baselines under all combinations of anchor numbers and estimation positions. Second, the time taken by our encoding method grows slower with the number of anchors compared to other encoding approaches.

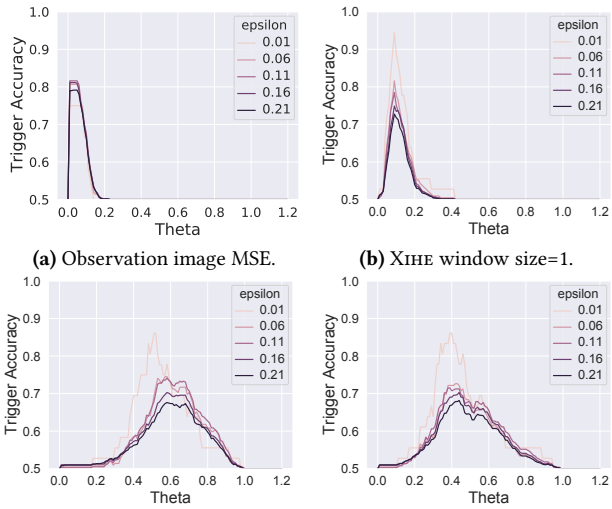
6.5 3D Vision-based Estimator Performance

We evaluate the performance of XiheNet and compare its accuracy to a state-of-the-art lighting estimator PointAR [33]. We use the same 3D indoor dataset (about 608k training and 2037 test data) as PointAR and preprocess each image to generate the unit sphere-based point cloud and SH coefficients pairs. We extract SH coefficients from the LDR format given its best visual rendering effects. We repeat the same process using the uniform random sampling to generate the training dataset for PointAR. We train



(a) Lighting estimation error. **(b)** Inference throughput.

Figure 14: *3D vision-based lighting estimator performance.*



(a) Observation image MSE. **(b)** Xihe window size=1.

(c) Xihe window size=4. **(d)** Xihe window size=16.

Figure 15: *Triggering strategy analysis.* Each strategy is evaluated on five ϵ-percentile split datasets.

both XiheNet and PointAR using the same hyperparameters, i.e., 2 PointConv blocks, each with multilayer perceptron setup of (64, 128) and (128, 256) [33]. Additionally, we train a third model (with the same backbone as XiheNet) with farthest point sampling [24].

Figure 14 shows the performance of the 3D vision-based estimators. We use SH coefficients RMSE, which is defined as the numerical difference between the predicted and ground truth SH coefficients, as the metric to evaluate the lighting estimation accuracy [8]. Our XiheNet achieves better SH coefficients RMSE (the lower the better) for all unit sphere-based point cloud sizes as shown in Figure 14(a). Further, we observe that XiheNet on a Nvidia RTX 2080Ti GPU can generate lighting estimations between 3.99ms to 10.80ms. As shown in Figure 14(b), we can support up to 250 inferences per second with batch size=1.

6.6 Triggering Strategy Analysis

We quantify the effectiveness of different triggering strategies using the metric *triggering accuracy* that describes the percentage of correctly identified environment lighting changes between two consecutive frames. We create a new dataset that consists of 1754 image pairs and a binary label for each pair indicating the lighting change. The image pairs are formed with two methods: *(i)* select data items from the test dataset that share the same observation image but from different estimation locations; *(ii)* randomly select a pair of data items.The first pairing method covers the slower-changing scene scenarios while the second pairing method covers faster-changing ones. To assign the lighting change label to each pair, we use three common metrics, SH coefficients RMSE and two image-based metrics (reconstructed irradiance map PSNR and SSIM), that are used for describing lighting conditions. We then sort each image pair in ascending order based on the metric calculation and label the lower and higher ϵ percents as 0 and 1, respectively. A label of 0 indicates no lighting change while a label of 1 indicates lighting change. By ignoring the middle percentile, our label assignment method allows us to automatically distinguish the pairs that exhibit lighting changes from ones that do not with high confidence. We manually verify the labeling results.

Figure 15 compares the triggering accuracy with different dataset partition threshold ϵ and triggering threshold θ, using the SSIM metric. Results corresponding to other metrics show similar trends and are omitted. We first observe that the observation MSE-based triggering strategy has a good accuracy when the triggering threshold θ is properly configured. Outside a small range of θ, the triggering accuracy is significantly lower. Second, we see that Xihe's sliding window based strategy can achieve better triggering accuracy than the MSE-based strategy and it's less sensitive to the choice of θ. Using larger window sizes have little impact on the triggering accuracy. As different window sizes correspond to different computation complexity, we use a default of window size=4 in Xihe.

Table 5: *Real-world evaluation of Xihe.* We record and replay three AR sessions under different lighting and movement dynamics.

Variable	Threshold θ	SH coefficients RMSE Mean	SH coefficients RMSE Std	Triggering Percentage
R1: Light temperature	0.5	0.0180	0.0119	1.59%
	0.6	**0.0193**	**0.0175**	**0.27%**
	0.7	0.0169	0.0047	0.13%
R2: Light intensity	0.5	0.0113	0.0065	2.69%
	0.6	**0.0110**	**0.0038**	**0.24%**
	0.7	0.0180	0.0054	0.24%
R3: User movement	0.5	0.0040	0.0231	48.41%
	0.6	**0.0070**	**0.0262**	**23.67%**
	0.7	0.0102	0.0051	5.91%

6.7 Lab-based Real-world Evaluation

In this section, we present a real-world evaluation of Xihe in a lab environment to demonstrate its effectiveness of our triggering algorithm. We show that with optimal configuration ($\theta = 0.6, N = 4$, 1280 anchors), Xihe can skip sending at least 76.24% estimation requests to the edge while still achieves comparable accuracy to running inference every frame. We use Xihe's session recorder to capture recordings when the user is interacting with our reference AR application in an indoor environment. We create both lighting condition and movement dynamics by using a remotely controlled light source and having the user walk around the light source.

For each recording, we control one of the variables, i.e., light temperature, light intensity, and user movement. The light source allows us to vary the temperature from candle light (1500K) to daylight (6500K) with 500K increment and the intensity from 0% to 100% (800 lumens) with 1% increment. We record relevant AR session information per frame. In total, we create three recordings with an average length of 35 seconds. We replay each recorded AR

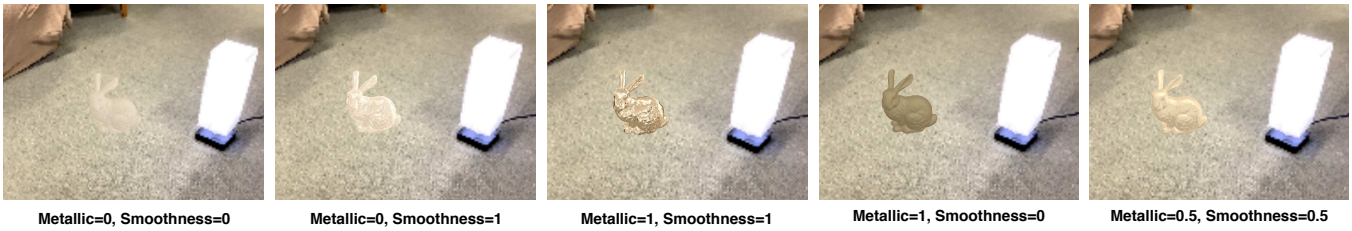


Figure 16: *AR scenes rendered with Xihe with different materials.* Both metallic and smoothness are Unity parameters.

session to Xihe and report both the SH coefficients RMSE and the percentage of triggered frames (i.e., being sent to the edge).

As shown in Table 5, Xihe ($\theta = 0.6$) only needs to send up to 23.67% inference requests to the XiheNet while only incurring an average RMSE of 0.011. We also inspect the visual effects of the rendered object during the replay and confirm minimal differences with and without the triggering algorithm enabled. Interestingly, for the third recording where the user is walking around the light source with the iPad Pro, Xihe triggers a lot more inference requests than the other two recordings. We suspect that more frequent triggering is likely due to increased observation completeness at estimation positions and enlarged viewing angles.

7 RELATED WORK

To provide mobile AR that is suitable for real-world deployment, researchers have been working on aspects including energy optimization, interactivity, and lifelike rendering [2, 3, 26]. The key to achieve lifelike rendering in AR is the ability to obtain accurate lighting information [8, 25, 29]. Although intuitively simple, there are a number of AR-specific challenges that distinguish this task from prior work in the graphics community [6, 11, 25].

There has been little work on providing real-time lighting estimation for mobile AR [5, 21, 33]. Commercial SDKs such as ARKit [13] and ARCore [9] only provide ambient lighting estimation which is often insufficient to capture the spatially-variant environment lighting. A recent work GLEAM leverages physical probes and improves the rendering effects over these commercial SDKs [21]. However, the use of physical probes hinders the user experiences. Our work is the first 3D vision-based framework that provides spatially-variant lighting estimation in real time.

The system design of Xihe is largely inspired by our empirical study and tackles problems that are common to edge-based AR systems [16, 18, 19]. Specifically, when designing Xihe, we focus on minimizing the reliance on mobile resources to avoid excessive power consumption [2, 3, 12]; we also minimize the network communication to the edge server by only issuing requests that are likely to improve the lighting estimation accuracy, e.g., when the lighting condition changes or when Xihe has more updated environment information [4, 17]. Our work differs from existing work on edge-based AR systems in addressing the lighting estimation-specific requirements and 3D vision-based opportunities.

8 DISCUSSION

We made the conscious decision to co-design some aspects of Xihe, including the unit sphere-based point cloud sampling and the triggering metric, with a state-of-the-art 3D lighting estimator [33]. We believe such application-specific optimizations are worthwhile trade-offs, allowing us to fully explore the performance potential of both algorithms and systems. In other words, rather than exposing the trade-offs of accuracy and performance to AR developers, we offload such responsibilities to the framework design phase.

Nevertheless, the lighting estimation accuracy provided Xihe will be bounded by the supported lighting estimators. For example, as XiheNet currently only demonstrates good estimation accuracy for low-frequency lighting, virtual objects that require high-frequency lighting information (such as metallic finish) will have less photorealistic rendering effects. Figure 16 show the visual effects of Stanford bunny with different material settings. Lower metallic values will give more matte-looking finish while lower smoothness values will lead to higher diffused reflection.

Additionally, to *effectively* support any future models on Xihe, respective components have to be rethought and redesigned. However, given Xihe's modular design and that its major components are general enough, we do not anticipate substantial changes. Lastly, Xihe currently provides an end-to-end 3D vision-based lighting estimation service per AR session. To support multi-user shared AR sessions, we will at least need to redesign the Point Cloud Management module to carefully manage the lifecycles and states.

9 CONCLUSION

Our system Xihe is a 3D vision-based lighting estimation platform that provides fast and accurate spatially-variant lighting estimation for mobile AR systems. Specifically our unit sphere-based point cloud sampling allows us to effectively downsample the raw point cloud captured in real time without compromising the lighting estimation accuracy. To avoid unnecessary network communication between mobile and edge, we designed an adaptive triggering algorithm that only sends unit sphere-based point cloud to the edge when there are significant lighting condition changes. The good estimation accuracy is guaranteed by our 3D-based lighting model that is inspired by recent work [26, 33] and is redesigned to consider both network and storage cost. We implemented Xihe on top of Unity3D, ARFoundation, and Pytorch frameworks. Our controlled experiments with three devices including a Lidar-enabled iPad Pro demonstrated that Xihe can provide visually-better rendering than ARKit and GLEAM under various experiment settings.

ACKNOWLEDGMENTS

We thank all anonymous reviewers, our shepherd, and our artifact evaluator Tianxing Li for their insight feedback. This work was supported in part by NSF Grants #1755659 and #1815619.

REFERENCES

[1] [n.d.]. *Depth buffer - The gritty details.*

[2] Kittipat Apicharttrisorn, Xukan Ran, Jiasi Chen, Srikanth V Krishnamurthy, and Amit K Roy-Chowdhury. 2019. Frugal Following: Power Thrifty Object Detection and Tracking for Mobile Augmented Reality. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems* (New York, New York) *(SenSys '19)*. ACM, New York, NY, USA, 96–109.

[3] Kaifei Chen, Tong Li, Hyung-Sin Kim, David E Culler, and Randy H Katz. 2018. MARVEL: Enabling Mobile Augmented Reality with Low Energy and Low Latency. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems* (Shenzhen, China) *(SenSys '18)*. ACM, New York, NY, USA, 292–304.

[4] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. 2015. Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems* (Seoul, South Korea) *(SenSys '15)*. Association for Computing Machinery, New York, NY, USA, 155–168.

[5] 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

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

[7] 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).

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

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

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

[11] Johannes Gunther, Stefan Popov, Hans-Peter Seidel, and Philipp Slusallek. 2007. Realtime ray tracing on GPU with BVH-based packet traversal. In *2007 IEEE Symposium on Interactive Ray Tracing*. IEEE, 113–118.

[12] Jinhan Hu, Alexander Shearer, Saranya Rajagopalan, and Robert LiKamWa. 2019. Banner: An Image Sensor Reconfiguration Framework for Seamless Resolution-based Tradeoffs. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services* (Seoul, Republic of Korea) *(MobiSys '19)*. Association for Computing Machinery, New York, NY, USA, 236–248.

[13] Apple Inc. 2020. Introducing ARKit 4. https://developer.apple.com/augmented-reality/arkit/.

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

[15] Inter IKEA Systems B. V. 2017. IKEA Place. https://apps.apple.com/us/app/ikea-place/id1279244498. Accessed: 2020-7-2.

[16] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge Assisted Real-time Object Detection for Mobile Augmented Reality. In *The 25th Annual International Conference on Mobile Computing and Networking (MobiCom'19)* (Los Cabos, Mexico) *(MobiCom '19, Article 25)*. Association for Computing Machinery, New York, NY, USA, 1–16.

[17] Q Liu and T Han. 2018. DARE: Dynamic Adaptive Mobile Augmented Reality with Edge Computing. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. 1–11.

[18] Q Liu, S Huang, J Opadere, and T Han. 2018. An Edge Network Orchestrator for Mobile Augmented Reality. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. ieeexplore.ieee.org, 756–764.

[19] 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.

[20] Samuel S. Ogden and Tian Guo. 2018. MODI: Mobile Deep Inference Made Efficient by Edge Computing. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*.

[21] 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.

[22] PyTorch. [n.d.]. Tensors and Dynamic neural networks in Python with strong GPU acceleration. https://github.com/pytorch/pytorch. Accessed: 2020-8-4.

[23] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2016. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *arXiv preprint arXiv:1612.00593* (2016).

[24] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/d8bf84be3800d12f74d8b05e9b89836f-Paper.pdf

[25] Ravi Ramamoorthi and Pat Hanrahan. 2001. An efficient representation for irradiance environment maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*. ACM Press, Not Known, 497–500. https://doi.org/10.1145/383259.383317

[26] Kai Rohmer, Johannes Jendersie, and Thorsten Grosch. 2017. Natural environment illumination: Coherent interactive augmented reality for mobile and non-mobile devices. *IEEE transactions on visualization and computer graphics* 23, 11 (2017), 2474–2484.

[27] Ruwen Schnabel and Reinhard Klein. 2006. Octree-based point-cloud compression. In *Proceedings of the 3rd Eurographics / IEEE VGTC conference on Point-Based Graphics* (Boston, Massachusetts) *(SPBG'06)*. Eurographics Association, Goslar, DEU, 111–121.

[28] Shuran Song and Thomas Funkhouser. 2019. Neural Illumination: Lighting Prediction for Indoor Environments. *CVPR* (2019).

[29] 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*.

[30] Unity. 2020. AR Foundation 4.2.0-preview.5. https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.2/manual/index.html.

[31] Unity3D. [n.d.]. Unity Profiler. https://docs.unity3d.com/Manual/Profiler.html. Accessed: 2020-8-4.

[32] Wenxuan Wu, Zhongang Qi, and Li Fuxin. 2019. PointConv: Deep Convolutional Networks on 3D Point Clouds. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[33] 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.