HoloAR: On-the-fly Optimization of 3D Holographic Processing for Augmented Reality

Shulin Zhao
suz53@psu.edu
The Pennsylvania State University
State College, PA, USA

Haibo Zhang*
huz123@psu.edu
The Pennsylvania State University
State College, PA, USA

Cyan S. Mishra
cyan@psu.edu
The Pennsylvania State University
State College, PA, USA

Sandeepa Bhuyan
sxb392@psu.edu
The Pennsylvania State University
State College, PA, USA

Ziyu Ying
ziy5087@psu.edu
The Pennsylvania State University
State College, PA, USA

Mahmut T. Kandemir
mtk2@psu.edu
The Pennsylvania State University
State College, PA, USA

Anand Sivasubramaniam
axs53@psu.edu
The Pennsylvania State University
State College, PA, USA

Chita R. Das
cxd12@psu.edu
The Pennsylvania State University
State College, PA, USA

ABSTRACT

Hologram processing is the primary bottleneck and contributes to more than 50% of energy consumption in battery-operated augmented reality (AR) headsets. Thus, improving the computational efficiency of the holographic pipeline is critical. The objective of this paper is to maximize its energy efficiency without jeopardizing the hologram quality for AR applications. Towards this, we take the approach of analyzing the workloads to identify approximation opportunities. We show that, by considering various parameters like region of interest and depth of view, we can approximate the rendering of the virtual object to minimize the amount of computation without affecting the user experience. Furthermore, by optimizing the software design flow, we propose *HoloAR*, which intelligently renders the most important object in sight to the clearest detail, while approximating the computations for the others, thereby significantly reducing the amount of computation, saving energy, and gaining performance at the same time. We implement our design in an edge GPU platform to demonstrate the real-world applicability of our research. Our experimental results show that, compared to the baseline, *HoloAR* achieves, on average, 2.7× speedup and 73% energy savings.

CCS CONCEPTS

• **Computing methodologies** → *Ray tracing*; • **Computer systems organization** → *Embedded software*; • **Human-centered computing** → *Visual analytics.*

KEYWORDS

Augmented Reality, Holographic Processing, Approximation, Energy-efficiency

ACM Reference Format:
Shulin Zhao, Haibo Zhang, Cyan S. Mishra, Sandeepa Bhuyan, Ziyu Ying, Mahmut T. Kandemir, Anand Sivasubramaniam, and Chita R. Das. 2021. HoloAR: On-the-fly Optimization of 3D Holographic Processing for Augmented Reality. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21), October 18–22, 2021, Virtual Event, Greece.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3466752.3480056

1 INTRODUCTION

Augmented reality (AR) has gained recent traction in both the consumer and research communities, thanks to the advances in efficient and low power computing technologies, high-speed communication, and specialized hardware platforms. These technologies have become an important part of our daily life, in the form of creative photography, content creation, gaming, online shopping, virtual touring, and educational and non-educational training, etc. For example, one of the earliest AR games, Pokémon GO (launched in July 2016), had a cumulative download of over 1 Billion, and generated about $900 Million in revenue by late 2019[1]. Moreover, these AR infotainment applications have helped many of us through the recent global pandemic by bringing us the liveliness of the virtual outdoors, while we were confined to our homes, and more AR capable mobile devices penetrating the market with cheaper price tags have made AR applications pervasive and made the virtual world easily accessible for users on the tip of their fingers.

However, even the state-of-the-art mobile devices with high bandwidth cannot meet the heavy compute and real-time demands of the AR applications, leading to very low quality of service (QoS) – in some cases as low as 1 frame per second (fps) [19, 54]. Further,

*Work was done as a student at Penn State.

[1]To give a quantitative estimation of the popularity of the game, a Pokémon GO event at Safari Zone New Taipei City, Taiwan in October 2019 had a total of 327,000 attendees and they walked around 4.5 million kilometers to catch 50 Million Pokémons [5].

MICRO '21, October 18–22, 2021, Virtual Event, Greece

ACM ISBN 978-1-4503-8557-2/21/10…$15.00
https://doi.org/10.1145/3466752.3480056

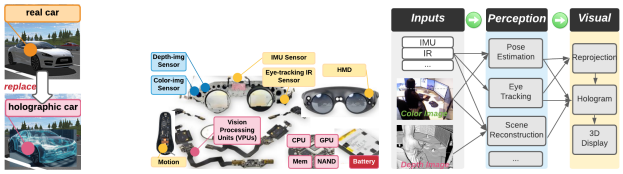
the limited battery capacity prevents users from enjoying their AR devices for extended periods of time.

To meet the heavy compute demands of these applications, most of AR applications are run using high-end desktop/server-class GPUs [18, 55], or specialized hardware accelerators [35] on cloud platforms [16, 27]. However, since most of these applications are now running on low-power mobile devices, and frequent communication of data to and from cloud via wireless medium is inefficient, optimization of an AR pipeline to maximize the compute and energy efficiency, while providing adequate QoS, at an edge device is an architectural challenge. Furthermore, existing AR headsets are typically equipped with multiple sensors for head orientation, eye tracking, motion detection, etc., to provide an interactive and life-like experience. These sensor inputs play a major role in deciding which portions of the 3D voxels need to be rendered for the user to view. However, even selective rendering of the portion of a scene, which is in the field of view (FoV) of the user, on a mobile device with limited compute and power budget is challenging [19, 52]. This calls for finding further opportunities for optimization. To understand the computing requirements in a typical AR pipeline consists of many stages (refer Sec. 2), we profiled a set of applications and found that the *hologram* processing is the primary bottleneck in terms of computation, energy consumption, and execution latency.

The heavy compute demand of the hologram (re)construction has made this a promising candidate for acceleration, and prior works have tried to offload it to cloud [16, 27, 67] and specialized accelerators [35] to achieve high throughput, but doing so has led the communication with the edge device to be a major bottleneck. Others have proposed to design efficient and lightweight deep neural networks (DNNs) to achieve high quality scene rendering at the edge device itself, but this requires model retraining/tuning for a particular user [33, 54]. Apart from the above works targeting all areas in a scene, *foveated rendering* techniques have been proposed to reduce image resolution in the peripheral area (typically beyond 135° vertically and 160° horizontally in human visual system (HVS)), while maintaining a normal/high quality only for 5° foveal vision [2, 22, 25, 47, 62]. Such differential resolution within an image can reduce computational costs without significantly impacting user experience [25, 47, 62]. Further optimizations such as eye-dominance (i.e., HVS prefers scene perception from one eye over the other) and learning-based foveated rendering are orthogonal to this core idea and beyond the scope of this paper [24, 30].

Despite providing significant performance and energy-efficiency benefits, these prior works still miss out on even more selective rendering of viewed hologram images - beyond just the FoV and/or regions of the user's focus. This is the primary motivation of this paper, where we explore trade-offs between hologram quality and processing costs. These trade-offs are not very straightforward due to the following **challenges**: First, *among all of the inputs to the AR headset (shown later in Fig. 1b), which one(s) are critical for holographic processing?* Second, *which features of these inputs are salient and need more fine-grained computation, and which of them could be approximated without impacting the QoS?* Third, *how do we make dynamic decisions of approximation based on the runtime conditions (e.g., user's current pose and eye movements)?*

Towards this, we propose *HoloAR*, an opportunistic and edge-friendly framework to speed up the AR holographic computation



(a) An app. (b) HW components [20, 29]. (c) SW pipeline [19, 50].

Figure 1: Main hardware components and software pipeline on a typical AR device.

and improve its energy efficiency, with "approximation" as the core idea. Starting from investigating and evaluating the existing foveated rendering techniques, this work further explores the entire design space for potential opportunities and optimizations unique in AR applications, for speedup as well as energy savings. The major **contributions** of this work can be summarized as follows:

- We first conduct a detailed characterization of a generic AR processing pipeline to identify the major bottlenecks in current state-of-the-art AR headsets, and set our optimization target as the *hologram computation*. (Sec. 2.1)
- From two open-source AR datasets [1, 58], we identify two properties in AR hologram applications: *spatio diversity for objects*, and *temporal locality for the user (viewer) interests (i.e., user typically focuses on one region within a short period of time)*. Such properties are leveraged as approximation opportunities to skip the "unimportant" portions of the hologram computation, based on user's region of focus (known as foveated rendering), and object's distance/size from the user. (Sec. 2.2)
- To capture these two approximation opportunities from both the user and object perspectives, first, the prior *foveated rendering* idea (denoted as *Inter-Holo* design) has been implemented (in Sec. 4.3) and found to work well (in Sec. 5) as in prior works [22, 25, 30, 47]. In this paper, we have gone beyond foveated rendering (*Inter-Holo*), by proposing an optimization/approximation called *Intra-Holo*, that complements the former in boosting performance/energy efficiency. Such *Intra-Holo* enhancement is ideally suited for holographic processing at the edge, without requiring additional hardware, cloud assistance, or machine learning.
- We implement both the designs on an edge GPU platform [36], without the need for any hardware modification. We evaluate these designs using the NVPROF tool [37] and hardware power management unit on the edge GPU platform [36]. Our experimental results reveal that, *HoloAR* provides 29% reduction in power consumption and 2.7× speedup, which collectively translate to 73% total energy savings compared to the baseline setup (Sec. 5.3). Finally, based on our findings, we discuss future directions that may help one design custom hardware accelerators for AR holograms (Sec. 5.5).

2 BACKGROUND AND MOTIVATION

Before diving deep into the problems and possible solutions associated with holographic processing, we first present the hardware

and software pipelines of a typical holographic AR application (in Fig. 1). We further describe the existing holographic execution inefficiencies in the AR pipeline and potential opportunities for computation reduction.

2.1 AR Holographic Applications and Pipeline

The holographic display technique enables a large body of augmented applications in real life [14]. One such application is illustrated in Fig. 1a, where a physical car being driven on a highway is replaced by the corresponding virtual/augmented holographic car in a real-time fashion such that, instead of viewing the real cars, the AR user views the virtual ones. To implement such applications, today's AR headsets are usually equipped with various hardware components for sensing and processing, as depicted in Fig. 1b. Specifically, the AR hardware has three major components:
Sensor Inputs: The AR headset receives the real-time information from both the surrounding environment and the user (viewer), with two types of sensing: ① *"world sensors"* to sense the physical surrounding the user is currently in, such as cameras for the RGB image and LiDAR/depth sensor for the depth or distance of the objects in front of the user, and ② *"user sensors"* to sample the behavior/status of the user, such as inertial measurement unit (IMU) sensors for head rotation, IR sensors for eye tracking, and controller for hand gesture. After sensing, the input samples are then buffered in the video buffer, waiting to be processed timely at the frame-rate.
Processing Engines: To efficiently handle the above two types of inputs, various computational resources have been integrated into AR SoCs, as shown in Fig. 1b, e.g., CPUs for generic processing, GPUs for graphics computing, vision processing units (VPUs) for rendering, and tensor processing units (TPUs) for learning inferences. Recently, state-of-the-art AR headsets such as HoloLens [31] have even been planning to integrate the holographic processing units (HPUs) for processing the information coming from all of the on-board sensors (currently under development) [32].
On-board Battery: It is to be noted that all of the sensors and the processing engines mentioned above are *battery-backed*, as shown in Fig. 1b. This is for enabling users to freely move around in a large area without the need of connecting with a power cable constantly. Hence, the power/energy efficiency is critical metrics in many AR use cases so that the battery lifetime can be sufficiently long.

With these sensors and compute resources in place, an AR headset executes a set of software tasks, either entirely or selectively based on the applications' requirements [19]. Without loss of generality, a typical AR pipeline [19] is shown in Fig. 1c. At a high level, this AR pipeline has three major stages: ❶ *Inputs* stage first collects the real-time information from all the on-board sensors such as IMU, IR, camera and depth image sensors. With these inputs, ❷ *Perception* stage understands the current surrounding environment such as pose estimation for head rotations/directions, eye tracking for pupil centers, and scene reconstruction for the current view analysis. Finally, ❸ *Visual* stage combines the physical world with the virtual information (which is generated in real-time) together, and renders the final images (both the physical scene as well as the virtual frame augmented with it) for the user to view.

We want to emphasize that, compared to virtual reality (VR), the AR video processing typically incurs additional computational tasks and interacts with more hardware resources [61]. Based on our measurements collected from a smartphone [60] running a simple AR application [3], the processing performance can be lower than 0.5 fps, and the battery life can be as short as just 1 hour. This motivates us to investigate which component is the major performance and energy bottleneck, charging most of the "performance-and/or energy-taxes" from the battery-backed AR headsets.

Table 1: Ideal latency requirements [19].

Task	*Ideal Latency*	*Algo.*
Pose Estimate	33 ms	Kimera [53]
Eye Track	33 ms	NVGaze [26]
Scene Reconstruct	100 ms	InfiniTAM [50]
Hologram	33 ms	GSW [49, 63]

2.2 Motivation

2.2.1 What is the Major Bottleneck?

To identify the major performance bottlenecks in the current AR headsets, we characterized the execution latency of the software pipeline (discussed above in Fig. 1c) on a typical edge prototype [36] running a set of state-of-the-art AR-related tasks [19, 26, 49, 50, 53], and compared the collected results against ideal execution latencies for the same set of tasks (i.e., the maximum latency within which the task needs to finish before its next invocation). The ideal latencies and our collected latencies are given in Table 1 and Fig. 2, respectively, for an ILLIXR playground scenario [15, 19].

Comparing the ideal latencies with practical latencies, we make the following conclusions: In our practical setting, *Pose Estimation* tracks user's motion and viewing scene to estimate the current body pose [53], and it takes around 13.8ms. Furthermore, estimating the user's eye gaze, *Eye Track*, requires the execution of a light-weight neural network that takes 4.4ms and achieves 2.06° of accuracy [26]. Thus, both of these two tasks are able to meet the performance requirements shown in Table 1. On the other hand, *Scene Reconstruct* captures comprehensive consistent maps of environments from an RGB-D image, and consumes 120ms in the practical setting. Note that such maps are not necessarily required to be generated for each frame (typically computed once per two or three frames [28, 50], thus $67-100ms$ in Table 1); hence, we argue that the state-of-the-art InfiniTAM technique, which implements a framework for real-time depth fusion and learning of 3D scenes [50], is already close to the ideal case. However, *Hologram*, which takes depthmap, point-cloud, or light field as its input [18][2] to create arbitrary 3D configurations of optical traps useful for capturing, moving and transforming mesoscopic objects freely in the world [4], takes as long as 341.7ms on an edge GPU[3]. This 10× performance gap between the practical scenario and the ideal case (and the large amount of power/energy consumption this task makes) motivates us to focus on holographic processing in this paper, and explore the opportunities for improving the hologram computational efficiency to speed up the overall AR application execution and reduce its energy consumption.

[2]In this paper, we mainly use the popular depthmap input method.
[3]Five iterations of the GSW algorithm [63] are profiled.

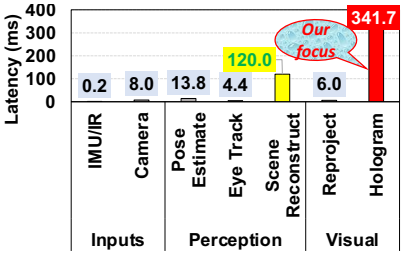
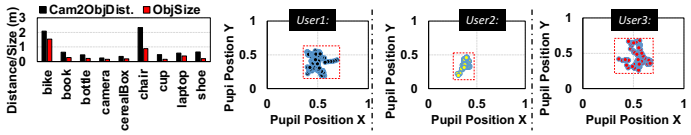
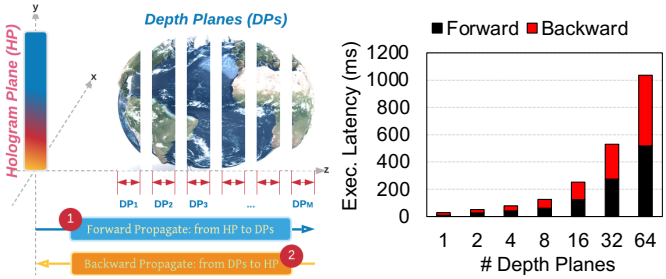


Figure 2: A comparison of latency requirements results collected from our practical setting and ideal cases shown in Table 1.



(a) Object study. (b) User eye tracking study.

Figure 3: Dataset study.



(a) Depthmap hologram algorithm. (b) Latency w/ num of depth planes.

Figure 4: Depthmap hologram algorithm details.

2.2.2 *What are the Prior Optimization Efforts?*

Targeting the compute-intensive holographic processing discussed above, *foveated rendering* techniques have been previously proposed to approximate selective regions (i.e., *peripheral vision*). In fact, prior research on HVS has shown that human eyes are able to observe beyond 135° vertically and 160° horizontally, but see fine details within an only around 5° central circle (i.e., *foveal vision*). Motivated by such degradation of *peripheral visual acuity*, *foveated rendering* reduces computational costs for the peripheral region, and maintains high/normal resolution only for the foveal region [2, 22, 24, 25, 30, 47, 62]. For instance, a real-time gaze-tracked foveated rendering system is proposed to yield performance and memory savings by avoiding shading up to 70% of the pixels for VR headsets [47]. Similarly, a prototype AR display also takes advantage of foveated rendering by tracking the user's gaze and providing low-resolution images to the peripheral area to reduce computation and improve display resolution [25]. More recently, another foveated rendering based CGH reconstruction technique has been proposed to accelerate calculations with negligible effect for the viewer [22]. We implemented such foveated rendering idea (denoted as *Inter-Holo* design in Sec. 4.3) and found to reduce around 23% execution latency (in Sec. 5) for AR holograms. However, such performance gain from foveated rendering is still insufficient to close the 10× gap discussed above. Thus, in this paper, we want to go beyond the prior foveated rendering for further optimizations, by investigating the potential opportunities which are unique to the AR use cases and may have been missed out before.

2.2.3 *What are the Potential Opportunities?*

Towards addressing this hologram bottleneck, various approaches from both the software [33, 52, 54] and hardware [32, 35] sides have been proposed. These prior approaches either incorporate additional memory for maintaining a lookup table for computation reduction, or build an application-specific integrated circuit (ASIC) chip specifically for holographic processing, which is more power-efficient than generic processors. While such approaches improve the hologram execution to some extent, they do not consider the unique features of the AR applications. Recall that, in the AR holographic application discussed above in Fig. 1a and Section 2.1, there are two types of inputs to the holographic pipeline – *world sensors* for the physical objects (real cars in this case) in the world, and *user sensors* for the user behavior/state such as pose and eye movements (discussed in details in Sec. 3). Therefore, in principle, more intuitive opportunities could exist in the AR application domain, from *both* the object and user perspectives. To identify them, we studied two published AR datasets (Objectron [1] for objects shown in Fig. 3a, and MPIIDEye [58] for users shown in Fig. 3b), and observed the following two properties in the AR holographic applications:

Spatio Diversity for Objects: Intuitively, objects which are far from the user and with small-sized shapes require less information to generate the virtual hologram than others (more details are provided in Sec. 3). Hence, the distance between the user and the objects (`Cam2ObjDist` shown in black color in Fig. 3a), as well as the size of how the object seems/appears to the user (`ObjSize` shown in red color in Fig. 3a) affect the amount of computations actually required to provide just enough yet necessary virtual holograms. For example, compared to the `chair` object in Fig. 3a, the `bike` object is closer to the user, and also has a larger range/size (`size=farmost-nearest`); thus, more information is required to create the hologram for the `bike` for maintaining fairly good QoS than the `chair`. Therefore, one opportunity to reduce the amount of computation is to *approximate* the hologram processing based on the objects' distances and sizes.

Temporal Locality for the User Interests: As also established by prior foveated rendering proposals, the foveal vision (or Region of Focus, RoF) is only a small region in the current scene and can be traced by eye tracking techniques [26]. As can be observed from three users' eye tracking shown in Fig. 3b, all focus only on a portion of the entire viewing window within a short period of time (10 seconds in this case). On the other hand, even when viewing the exact same scene, the RoF varies across users. For example, `User1` has similar interest as `User3`, whereas `User2` focuses more on the bottom left corner. Clearly, such temporal similarity for a particular user's interests exposes another opportunity for leveraging prior

foveated rendering in AR holograms, by reducing the amount of computation needed for the objects which are outside the RoF, thus only emphasizing on the processing of the objects which the user is currently focusing on.

Driven by these observations, we next want to study the details of hologram with the goal of addressing two critical questions: *What are the problems in the current state-of-the-art hologram software and hardware?*, and *How can we leverage "approximation opportunities" (based on the two observations above) to speed up hologram processing and save energy, while still maintaining a high QoS?*

3 HOLOGRAPHIC PROCESSING STUDY

To leverage the opportunities in the holographic processing from a RGB-D (i.e., RGB and depth) image, we need to first understand the detailed execution of the entire hologram processing from both the algorithm and hardware perspectives. We illustrate the details of depthmap hologram processing in Fig. 4a and Algo. 1 as two major steps (more details on the depthmap hologram algorithm can be found elsewhere [4, 18, 55, 63]). As shown in Fig. 4a, the depthmap input is first sliced into several planes (M depth planes in this case). With these depth planes, the first step, *Forward Propagation* (denoted ❶ in Fig. 4a), is to overlay the i^{th} plane on the propagation result of the previous 1^{st} to $(i-1)^{th}$ planes, and then propagate to the next $(i+1)^{th}$ plane. Note from `Line#3` to `Line#5` in Algo. 1 that, such forward propagation is massively parallel at the depth plane level (across planes) as well as at the pixel level (within one plane). Each depth plane processes the forward-propagation from the hologram plane independently, and each pixel on a particular depth plane goes through the exact processing sequence (`HP2DP` in `Line#5`; more details can be found in [4, 18]). This makes hardware parallelization and pipelining easier on a block/tensor type of architecture such as GPUs. Note, however, that, this step also requires sequential barriers within each plane (`Line#6` synchronizes the threads in a warp/block for one depth plane) and across planes (`Line#7` synchronizes the results from all the depth planes, before moving forward to the second step). Hence, as we will show later in this section, such barriers sliced into the massive parallel execution can cause load imbalance and instruction stalls, which slow down the entire execution and impact performance. The second step, *Backward-Propagate* (denoted ❷ in Fig. 4a), accumulates the results of each depth plane, backpropagates it to the hologram plane via the `DP2HP` procedure (in `Line#11`), and generates the final hologram for this depthmap input. Like the first step, this step also involves synchronizations between planes (in `Line#12`), which can again impact parallelization and slow down the entire execution.

Intuitively, the execution performance is mainly determined by the number of depth planes (the outer for-loop in the algorithm) as well as the number of pixels in each depth plane (the inner for-loop in the algorithm). To study how the number of depth planes affects the hologram performance, we profile the execution latency from a typical edge GPU device [36], generating holograms with different number of depth planes (assuming the same number of pixels in each plane), and the results are plotted in Fig. 4b. From this figure, one can observe the following: First, in general, these two steps take similar times to execute, due to the similar procedures they employ, as shown in Algo. 1. Second, by increasing the number of depth planes, it takes around 2× latency to generate a hologram with 2× number of depth planes. As also mentioned in Sec. 2.1, the 16 depth planes required by most of the AR applications (typically 10 to 100 depth planes are sufficient) [19, 49] consume more than $300ms$, which is 10× larger than the real-time (QoS) requirement. Thus, it can be concluded that, without any optimization, a state-of-the-art edge GPU is only able to compute for < 4 depth planes in real-time [36]. These observations motivate us to investigate the reasons behind such low performance on GPU: is it because of the intrinsic software/algorithm characteristics, or is it primarily a hardware mapping issue?

Algorithm 1: Depthmap Hologram Algorithm [4, 18].

```
Input  : M: Number of depth planes
Input  : DP[i]: Pixels in the i^th depth plane
Output : Hologram: Generated hologram
1  procedure Depthmap_Hologram(M, DP)                    // main
2      // Step-1: Forward-propagate
3      for i in [1, M] do                                // planes in parallel
4          for p in DP[i] do                             // pixels in parallel
5              IntraPlane_i = HP2DP(i, p)
6          IntraBlockSync(IntraPlane[i])
7      InterBlockSync()
8      // Step-2: Backward-propagate
9      for i in [1, M] do                                // in parallel
10         for p' in IntraPlane[i] do                    // in parallel
11             Hologram[p'] += DP2HP(i, p')
12     InterBlockSync()
13     return {Hologram}
```

Towards this, we profiled the hologram processing on the edge GPU [36] using the NVPROF tool [37], and observed the following: First, the SM utilization for both the steps is very high, i.e., 74% for *Forward-Propagation* and 90% for *Backward-Propagation*. This is because the execution is massively parallel at the depth plane level as well as at the pixel level. Moreover, the L1 hit rate for both these steps is as high as 99%. Thus, GPU seems to be one of the reasonable hardware candidates for mapping the hologram application. Second, the four major reasons for instruction stalls in the *Forward-Propagation* step are: Data Request (21%), Execution Dependency (19%), Instruction Fetch (15%), and Sync (10%), whereas in the *Backward-Propagation* step they are Read-only Loads (42%), Sync (24%), Data Request (16%), and Execution Dependency (6%). These stalls originate mainly from the inter-block and intra-block synchronizations required by the application, as discussed above when explaining Algo. 1. Because of this, recently, alternate hardware-based solutions have been proposed to improve the computational efficiency by replacing the expensive transcendental calculations with lookup table (LUT) based memoization [35], or mitigating the data movement overheads by employing a customized buffer on-chip [32], or simply offloading computations to cloud then streaming back [16, 27, 67]. While such an approach improved the computational efficiency and reduced power consumption to some extent, rethinking the design of hologram software/hardware considering the unique features of the AR holographic applications (as discussed in Sec. 2.2) as well as the characteristics of the underlying hardware can potentially open up further opportunities.

Motivated by this, we next explore the entire design space for the AR holographic applications running on edge GPUs, and try to exploit potential opportunities for reducing computations to improve both performance and energy efficiency in hologram processing.

4 PROPOSED STRATEGIES

As discussed in Sec. 2.2, holographic processing dominates the latency and energy consumption in the AR video pipeline. Further, we also observed in Sec. 3 that, the main reason behind this is that the number of depth planes affects the number of synchronizations between parallel executions, and determines the amount of computation required to generate the holograms. Unlike prior works targeting at optimizing the efficiency of the hologram programming itself by proposing alternative hardware [32, 35], we primarily focus on exploring the intrinsic approximation opportunities (discussed in Sec. 2.2.3) ignored in the current implementation of the AR applications, but can be embedded into the existing hardware such as GPUs, to speedup the holographic execution and improve power/energy efficiency with negligible quality loss.

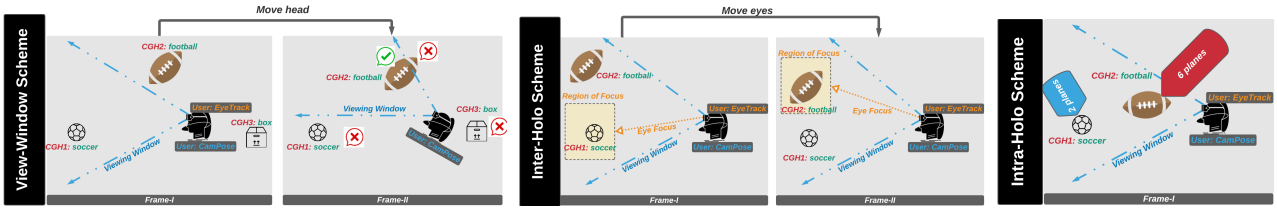
4.1 Exploring the Entire Design Space in AR Hologram Processing

Exploring the entire design space for the AR hologram processing is a non-trivial task. First of all, a large number of sensor inputs are fed into the hologram pipeline (as shown in Fig. 1b), such as IMU sensors, eye tracking or IR sensors, hand motion sensors, RGB-D image sensors, etc. To improve the hologram approximation, we need to first identify the set of inputs that affect the hologram computing the most. As discussed above in Sec. 3, both the user's pose and the gaze position, as well as the targeted objects (intended to be replaced by the virtual holograms) shape the hologram computation. Further, in many cases, these inputs are dynamically changing at the same frequency (e.g., the image sensors) as the frame-rate, which needs to be captured and updated at runtime, or even at a faster rate (e.g., the IMU and IR sensors). Thus, to systematically explore the potential opportunities of approximation in the AR hologram applications, we start by distinguishing between three fundamental scenarios, where the *objects*, *head pose*, and *eye tracking* provide different opportunities, as depicted in Fig. 5.

- In the *Viewing-Window* scenario shown in Fig. 5a, only the `soccer ball` object is located inside the viewing window in the current frame, `Frame-I`, while `football` and `box` are not. Thus, only the `soccer ball` hologram is required to be computed for this frame, and other two can be skipped. Similarly, for the next frame, `Frame-II`, now the user lifts her head a bit, hence the corresponding viewing window changes from the previous one. Because of this, now the `football` is partially located in the viewing window, and requires computing (only for the bottom right part that is inside the viewing window). Note also that, since the `soccer ball` hologram has been already generated in `Frame-I`, we can skip its computation. Again, the `box` object is still outside of the viewing window and thus, we do not need to compute its hologram. We use such a viewing-window based "sub-hologram" technique which has already been proposed in prior works (such as Sub-Hologram [52]) as the *Baseline* design.
- Apart from the viewing window, the dense or sparse hologram computing is also RoF-dependent, which is the main idea behind foveated rendering [25, 47, 62], as discussed in Sec. 1 and Sec. 2.2.3. In the *Inter-Holo* scenario shown in Fig. 5b, such a region of focus is just a subset of the entire viewing window, and thus contains a small number of objects that need to be computed with rich information (as it needs 16 depth planes). However, for the objects outside of the current RoF, since the user is not currently focusing on them, a reasonable approximation would not affect the user experience that much (which implies we do not need 16 depth planes for all of them). For example, in `Frame-I` in Fig. 5b, the user is currently focusing on the `soccer ball`; meanwhile the `football` is located outside of the RoF, hence, becomes a candidate for approximation. On the other hand, in `Frame-II`, the user moves her eyes and changes the region of focus. Now, the `football` needs the full depth planes' information, while the `soccer ball` can be approximated. To take advantage of this opportunity, the `football` object (which is inside the RoF in this example scenario) requires all of the 16 depth planes to compute its dense hologram, whereas the other objects (the `soccer ball` in this case) can be approximated with a pre-defined sparse sampling factor (e.g., $\frac{1}{2}$; more details provided later in Algo. 2 and Sec. 4.3). We leverage such foveated rendering idea in *Inter-Holo* as our *Reference* design in this paper.
- The above *Viewing-Window* and *Inter-Holo* proposals target at reducing the amount of computation for the holograms from the head orientation (`rotation`) and eye tracking (`up-down`) perspectives. As discussed in Fig. 3a, another enabler for computation reduction is the relative distance between the camera/user and the objects, i.e., `left-right`. As one can observe from the *Intra-Holo* scenario shown in Fig. 5c, the `football` is larger than the `soccer ball`, and is located much closer to the user. Hence, even though both of them are located inside the RoF (and, of course, inside the viewing window), intuitively, the `soccer ball` hologram does not need as much information as the `football` hologram to compute. Inspired by this observation, another level of approximation can be explored based on the relative camera-to-object distance as well as the object range/size.

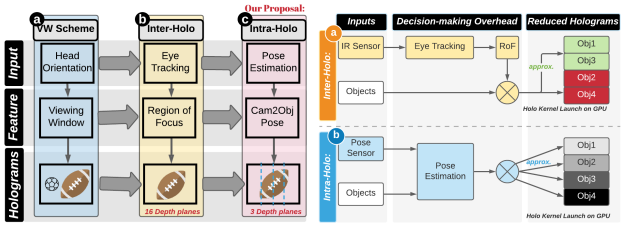
4.2 HoloAR Overview

Driven by the above discussion and the potential approximation opportunities presented by the *Inter-Holo* and *Intra-Holo* scenarios, we propose *HoloAR*, a novel framework for holographic processing in AR applications to improve *both* the performance and energy consumption of the hologram processing, without affecting user experience. *HoloAR* aims to reduce the amount of hologram computations as much as possible by carefully approximating the hologram computing for select objects, while maintaining an acceptable video quality. The overall design of our proposed *HoloAR* framework is illustrated in Fig. 6a. First, *HoloAR* utilizes the existing viewing-window based technique [52] (denoted ⓐ) to skip the hologram computations for the objects which are outside of the current viewing window, in a "just-in-time" fashion. Next, *HoloAR* employs the *Inter-Holo* scheme (denoted ⓑ), to take advantage of the region of focus from analyzing the current eye tracking inputs and sparsely compute the objects outside the RoF. Finally, *HoloAR*



(a) Viewing-Window scenario [52]. (b) Inter-Holo scenario. (c) Intra-Holo scenario.

Figure 5: Three opportunities for reducing hologram computation in an AR application.



(a) HoloAR overview. (b) The *Inter-Holo* and *Intra-Holo*.

Figure 6: The proposed *HoloAR* which includes *Inter-Holo* leveraging foveated rendering, and *Intr-Holo* further approximating holograms for far objects.

Algorithm 2: Inter-Holo algorithm.

Input : IRs: eye tracking sensors
Input : $Objs$: set of virtual objects
Input : α: inter-holo approximation factor, $\alpha \in (0, 1]$
Output: $Holograms$: Generated holograms

```
1 procedure Inter_Holo(IRs, Objs, α)                    // main
2     RoF = EyeTracking(IRs)
3     for obj in Objs do                                // View-Window only
4         if obj in RoF then                            // inside of RoF
5             Holograms[obj] = Algorithm1(16, obj)
6         else                                          // outside of RoF, thus approximate
7             Holograms[obj] = Algorithm1(16 × α, obj)
8     return {Holograms}
```

uses the *Intra-Holo* scheme (denoted ⓒ), to identify the number of depth planes required for a particular object by analyzing the relative camera-to-object distance as well as the shape/size of the target object. Note that, both the *Inter-Holo* and the *Intra-Holo* schemes are complementary to each other, when both the eye tracking and pose estimation inputs are available at the same time. Therefore, we also investigate a combined *Inter-Intra-Holo* scheme which combines both the schemes to further reduce the amount of hologram computations.

We would like to emphasize that the proposed *HoloAR* framework can, in principle, work with any hardware platform. As discussed later in Sec. 5, in this paper, we evaluate the performance and energy benefits of *HoloAR* by using an embedded GPU prototype for the edge AR headsets [36], and leave the hardware-software co-design based on FPGA-based acceleration for future work. However, the architectural insights on how to co-design a next-generation accelerator that can accommodate our proposed *HoloAR* framework are discussed later in Sec. 5.5.

4.3 Inter-Holo Computation Optimization

We first answer how to deploy the previously proposed foveated rendering technique on AR holograms, by investigating how to leverage the temporal similarity when the user's region of focus is only a part of the entire viewing window, as mentioned earlier in Sec. 2.2 (Fig. 3b). To capture the current RoF, an additional eye tracking step is introduced before the hologram computations, as shown in Fig. 6b ⓐ. This eye tracking step takes the current IR sensor images as its input, and analyzes the user's current gaze area as well as the viewing direction. Note that this additional eye tracking procedure needs to be invoked for each frame, in order to capture/reflect the current eye movements without causing nausea for the user. As a result, eye tracking needs to incur minimum overhead, while providing a fairly good accuracy. Fortunately, there already exist a large body of techniques which can track the eye movements efficiently (e.g., see [26] and [12] and the references therein). In this work, we chose to use the NVGaze technique [26] to perform eye tracking for the *Inter-Holo* design due to two main reasons. First, it provides sufficient accuracy for the AR applications – as high as 2.06° accuracy for gaze shape/direction estimation across a wide field of view [26]. Second, its execution latency when running on our edge GPU prototype [36] is within 4.5ms, which contributes to less than 1% of the entire hologram processing pipeline latency.

With the RoF attained from the eye tracking, the next question we need to answer is how to deploy the approximation opportunities discussed above in Sec. 2.2.3 on top of the existing hologram pipeline. As shown by `Line#5` and `Line#7` in Algo. 2, our proposal can actually reuse the original hologram execution engine without any architectural modifications or reprogramming. In fact, only one input argument, i.e., the number of depth planes, requires to be changed based on the approximation factor α, when the object is outside of RoF. Here, we set α to 0.5, as our detailed profiling (discussed later in Sec. 5) indicates that setting α to this specific value brings significant energy savings while maintaining good hologram quality. We also present a sensitivity study on how energy savings and performance vary with different approximation factors in Sec. 5.4.

4.4 Intra-Holo Computation Optimization

Algorithm 3: Intra-Holo proposal algorithm.

```
Input  : Poses: pose sensors
Input  : Objs: set of virtual objects
Output : Holograms: Generated holograms
1 procedure Intra_Holo(Poses, Objs)                    // main
2     Cam2ObjDists = PoseEstimation(Poses)
3     for obj in Objs do                    // approx. based on dist.
4         β = approxFactors(cam2ObjDists[obj])
5         Holograms[obj] = Algorithm1(16 × β, obj)
6     return {Holograms}
```

In the *Inter-Holo* design, the hologram computation can be approximated by identifying the region of focus from eye tracking. However, the scope of this approximation opportunity might be limited due to the strict 16 depth planes requirement for all objects inside the RoF, regardless of their distance from the user. In fact, there may still be another level of opportunity for approximating the objects in long distance (*Intra-Holo*, shown in Fig. 5c). To leverage this opportunity, we need to know where the user is located in the world and what the objects in the world look like [13, 19, 53, 59]. Next, we use one of the popular SLAM techniques, Kimera-VIO [53], to estimate the user's pose and understand the relative positions of the objects and the user. As shown in Fig. 6b ❻, similar to the *Inter-Holo* pipeline, the additional pose estimation step also sits between the inputs and the original hologram processing, and thus has to be efficient without introducing much overhead. Our profiling on the edge GPU prototype [36] shows that Kimera-VIO takes, on average, 13.75*ms* latency to execute, which is less than 1% of the total hologram processing time. Therefore, the overhead introduced due to the additional pose estimation step is negligible compared to the baseline latency, thereby opening up opportunities for significant energy savings and performance speedup as demonstrated later in Sec. 5.

With the help of the pose estimation, now the AR hologram pipeline has the knowledge about the range/size of each object as well as its relative distance from the user. Next, as shown in Algo. 3, for each of the objects, a corresponding approximation factor (β) can be determined based on these insights. Similarly, the original hologram engine can still be reused without any reprogramming, except for the first argument, i.e., the number of depth planes for this particular object, as shown in `Line#5` of Algo. 3.

Inter-Intra-Holo: It is to be noted that, when the user eye tracking and pose estimation are available simultaneously for hologram processing, the *Inter-Holo* and *Intra-Holo* schemes can be both applied to achieve maximum amount of energy savings and performance benefits. In this paper, we refer to this combined scheme as *Inter-Intra-Holo*. In this scheme, we first identify the objects inside/outside the RoF (*Inter-Holo*), and then approximate each of them based on its shape and distance (*Intra-Holo*). Note that since the other option – first *Intra-Holo*, then *Inter-Holo* – is theoretically identical to the proposed *Inter-Intra-Holo*, we skip its detailed discussion due to space limitation.

4.5 Design and Implementation

Optimization Choices: Our main goal in this paper is to reduce the amount of hologram computation by appropriate approximation, in order to speed up hologram processing, to satisfy the real-time requirement as well as to reduce the energy consumption and prolong the battery life of the AR device, while maintaining the QoS. Our proposal is fundamentally different from prior optimizations targeting various architectures or execution environments, such as customized hardware accelerators [35], cloud assistance [16, 27, 67], or neural network training/inferencing [33, 54]. Note that, each of these prior efforts has its own limitations, e.g., expensive in-house implementation and fixed functionality without proper power gating in accelerators [35]; requiring reliable network connections and expensive round-trip latency in cloud offloading [16, 27, 67]; and re-training of a new model for each application scenario and potentially for each user in neural networks [33, 54]. Thus, our proposal does not rely on any assistance from hardware accelerators, cloud platforms, or neural networks. Instead, we focus exclusively on a typical edge GPU to execute the hologram, and present ~~our~~ three techniques, namely, *Inter-Holo* (as *Reference*), *Intra-Holo* and *Inter-Intra-Holo*, which capture various approximation opportunities in the AR hologram applications to improve both performance and energy efficiency.

Framework Prototype: To prototype a real-life AR headset, a proper codebase and a hardware platform are essential. For our codebase, we build our proposals on top of ILLIXR [19], which is the first open-source full-system extended reality testbed. ILLIXR already contains several AR software components (some of them are shown in Fig. 1c), including head tracking, IMU integration, reprojection, and sound processing. On top of the ILLIXR codebase, we implemented three new components – eye tracking, pose estimation, and hologram processing. Further, we mapped these AR software components to an edge GPU prototype [36], from which the power breakdown across different components such as SoC, memory, CPU, and GPU are measured through the on-board Texas Instruments INA 3221 voltage monitor IC hardware, and the performance of execution status is sampled by the Nvidia NVPROF [37] profiling tool, which enables the collection of a timeline of CUDA-related activities on both the CPU and GPU, including kernel execution, memory transfer, CUDA API calls and events/metrics for CUDA kernels.

5 EVALUATION

We evaluate our proposed *HoloAR* design by comparing the execution latency and total energy consumption with four different AR hologram setups. In this section, we first describe our evaluation methodology, experimental platform, datasets, and measurement tools. Next, we analyze the results measured using these platforms. After that, we show the general applicability of the proposed design, and also present results from a sensitivity study that focuses on the quality-loss vs. energy-savings trade-offs. We conclude this section by outlining some research directions for implementing approximation-based accelerators for AR holograms.

Table 2: Salient features of the six videos used in this study.

No.	Video	#Frames	#Obj/Frame	Distance	ObjSize
1	bike[38]	150k	1.1	2.08m	1.54m
2	book[39]	576k	1.5	0.64m	0.28m
3	bottle[40]	476k	1.1	0.47m	0.22m
4	cup[41]	546k	1.6	0.47m	0.16m
5	laptop[42]	485k	1.3	0.58m	0.38m
6	shoe[43]	557k	2.3	0.65m	0.21m

5.1 AR Hologram Configurations

We evaluate the following five configurations of AR hologram processing to demonstrate the effectiveness of our proposed *HoloAR*:

- **Baseline (Viewing-Window):** Similar to the recent viewing-window based sub-hologram optimization [52], we first obtain the field of view or the current viewing window from the user's head orientation, and then skip the computations of the objects, which are outside the viewing window (i.e., only compute for the objects located inside) to save computations and energy. This software-based viewing window optimization is considered to be the state-of-the-art at an algorithm level, and we refer to it as *Baseline* in this study. We evaluate this baseline by profiling its performance and energy consumption from a mobile GPU [36].
- **Inter-Holo:** We evaluate the *Inter-Holo* design on a mobile GPU [36] using a framework similar to the state-of-the-art ILLIXR framework [19], with one additional eye tracking task (as shown in Fig. 6b(a)) integrated into the existing pipeline to partially bypass the computations of holograms that are outside the focus area. Note that, this implementation is purely done in software, without any hardware modification.
- **Intra-Holo:** We evaluate our *Intra-Holo* design again on a mobile GPU as shown in Fig. 6b(b). This approach tries to reduce the amount of hologram computation by approximating each of the holograms based on the distance between the user and the object.
- **Inter-Intra-Holo**: The above two designs can be integrated together into the original hologram pipeline, in either Inter-then-Intra or Intra-then-Inter fashion. In this paper, we chose the first one and denote this design as *Inter-Intra-Holo*.
- **HORN-8:** While hardware acceleration of hologram is not a goal of this work, to qualitatively compare our GPU-based design with hardware specific accelerators, we also discuss one of the most recent ASIC implementations, HORN-8 [35]. Due to unavailability of its hardware implementation or datasheet, we estimate its power efficiency compared to the equivalent GPU SoC based on a published data [51]. With this estimation, we briefly discuss the performance and computational efficiency variations between this accelerator and our approach, and discuss takeaways that can help one to co-design a hardware accelerator targeting hologram processing.

5.2 Experimental Platform and Datasets

The edge GPU platform used in this work consists of a 512-core Volta GPU, a 4Kp60 HEVC codec, 16GB LPDDR4x memory, 32GB eMMC storage, and a power management unit (PMU) that exposes the real-time power traces to users [36]. To ensemble the AR pipeline with generic state-of-the-art components shown in Fig. 1c, we implemented an open-source full-system extended reality testbed, ILLIXR [19], on the edge GPU platform [36], and built our *HoloAR* design on top of it. To collect performance metrics such as the streaming multiprocessor (SM) utilization, memory traffic, and CUDA kernel execution latency, we utilized the open-source Nvidia NVPROF tool [37] on the GPU platform.

We use the published short object-centric Objectron [1] video dataset, which is accompanied by AR session metadata such as camera poses, as well as the object annotations such as position, orientation and dimension for nine categories of object videos[4]. The salient characteristics these videos are given in Tab. 2. To replace these real objects, we choose six virtual holograms (Sniper, Rock, Tree, Planet, Rabbit, and Dice holograms) from the OpenHolo depthmap database [45]. Note that the real-object and the corresponding virtual-hologram are randomly mapped because, theoretically, different mappings have no impact on the performance speedup and energy saving results (shown in Sec. 5.3).

5.3 Experimental Results

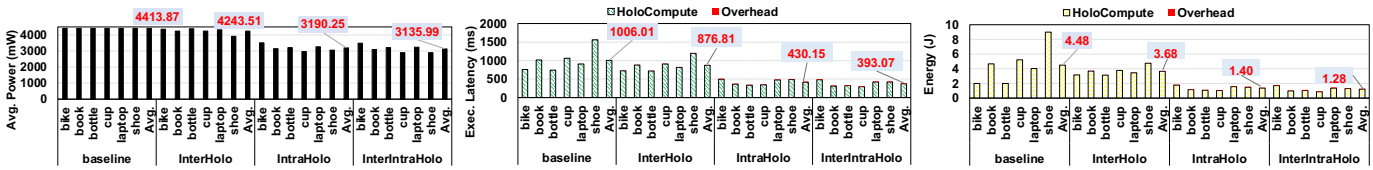
We present the power and energy consumption, as well as the execution latency of the hologram computation in Fig. 7, when processing the six videos listed in Tab. 2, with the first four configurations described earlier in Sec. 5.1. We discuss the impact of our proposal on output/result quality, compared to the baseline design, later in Sec. 5.4.

Power Consumption: Overall, the *Inter-Holo* scheme consumes around $4.24Watts$, on average, when running on the edge GPU, which translates to 3.86% power reduction, compared to the baseline. In addition, our *Intra-Holo* scheme is more power efficient than *Inter-Holo*, translating to 27.72% power reduction with respect to the baseline. This indicates that the optimization scope of the distance-based *Intra-Holo* is larger than that of the RoF-based *Inter-Holo*, which provides more sparsity in the hologram computing. Finally, combination of the two schemes (*Inter-Intra-Holo*), results in 28.95% power reduction compared to the baseline.

To better explain where the power benefits come from, we further breakdown the power consumption for the hologram processing into four parts: CPU (to handle sensor inputs, scheduling, kernel launch, etc.), GPU (to execute hologram), Mem (for data accesses), and the SoC (the remaining hardware components, e.g., codec, network), with different number of depth planes (ranging from 2 to 16), as shown in Fig. 8a. One can observe from this figure that, when the number of depth planes is increased, the power consumptions of *SoC* and *CPU* do not change much, while, in contrast, both the *GPU* and *Mem* consume more power. This is due to the fact that, to process a denser hologram with more depth planes, additional GPU cores are scheduled to launch the per-plane CUDA kernel (as discussed in Algo. 1) with more holographic data accesses (fetched from the host-side memory).

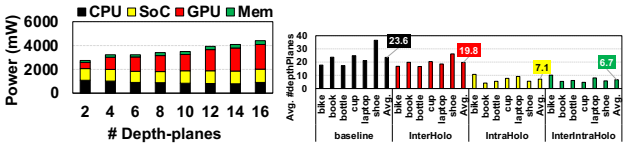
We next quantify how many depth planes can be reduced by our approximation scheme. Towards this, we plot, in Fig. 8b, the average number of depth planes (across our six videos), required by the four design alternatives (configurations). We see that, the number

[4]Due to space limitation, we chose six representative categories that cover diversity across multiple video parameters.



(a) Avg. power (mW). (b) Exec. latency (ms). (c) Energy consumption (J).

Figure 7: (a) Average power consumption, (b) execution latency, and (c) energy consumption with different configurations and video inputs.



(a) Power breakdown. (b) Avg. number of depth planes.

Figure 8: (a): Profiling the power breakdown on the edge GPU prototype [36]; and (b): Average number of depth planes required for four design configurations.

of depth planes required by the *Inter-Holo* scheme is reduced from 23.6 to 19.8, and even further to 7.1 and 6.7, by the *Intra-Holo* and *Inter-Intra-Holo* schemes, respectively. The above observations from these two figures explain the power benefits of our proposed designs.

Execution Latency: Clearly, the reduction in the number of depth planes when using our approximation schemes can reduce the hologram execution latency as well. As shown in Fig. 7b, overall, the *Inter-Holo* scheme provides a 1.15× speedup compared to the baseline. Further, a 2.42× speedup is achieved when employing *Intra-Holo* (with only 0.44% overhead), and 2.68× when employing *Inter-Intra-Holo* (with only 0.14% overhead). Recall that the number of depth planes for each hologram object affects the execution latency dramatically as shown in Fig. 4b; thus, these performance benefits come from the speedup brought by the reduced depth planes by approximation in our schemes. Another interesting observation is that, *Intra-Holo* saves more execution time than *Inter-Holo*. This is because the latter only approximates the objects outside of the current region of focus (still requiring full compute for the objects inside), whereas the scope of the former is much larger, i.e., including all the objects in the current viewing window and approximating each of them based on its location. In addition, from an individual video's perspective, we further observe that the `shoe` video achieves the maximum performance benefits from our schemes (specifically, 23%, 69% and 73% latency reduction with *Inter-Holo*, *Intra-Holo* and *Inter-Intra-Holo*, respectively, compared to the baseline). In contrast, the `bike` video achieves the minimum speedup (4%, 34% and 36% in the same order). The reason behind this is that, as shown earlier in Tab. 2, the `bike` video usually has only one object per frame (1.1 on average), and also the ranges/sizes of the bikes are larger, compared to others. Thus, chances for approximating the objects outside the RoF (in *Inter-Holo*) and the objects which are relatively far-away from the user (in *Intra-Holo*) are limited. On the other hand, the `shoe` video frames typically contain more objects (2.3 on average, as shown in Tab. 2), thereby gaining more opportunities to reduce the amount of computations for all the objects in the current frame.

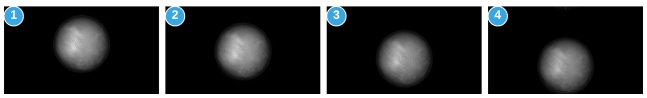
Energy Savings: The above power and latency reductions provided by *HoloAR* eventually translates to energy savings for the hologram processing. As shown in Fig. 7c, on average, the *Inter-Holo* scheme saves 18% energy compared to the baseline, and the *Intra-Holo* scheme saves 70% energy. Finally, the energy saving achieved by the *Inter-Intra-Holo* scheme is about 73%, meaning that it only consumes 27% of the baseline energy.

To put the energy-efficiency of our designs into perspective, we compared their energy consumption against the state-of-the-art HORN-8 hardware accelerator [35]. Due to the unavailability of the hardware RTL, we estimated its energy consumption based on the published characterization numbers from the Jetson GPU platform with the ZCU102 FPGA[64] (which is similar to the HORN-8 prototype) [51]. Because of the LUT memoization and power efficiency optimizations [35], HORN-8 saves around 48% power[5]. However, HORN-8 does not explore the approximation opportunities to speedup the hologram execution. Hence, as shown in Fig. 7c, our *HoloAR* design running on the edge GPU [36] still saves 25% more energy than the custom HORN-8 accelerator.

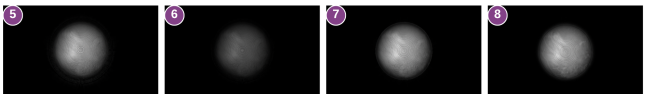
5.4 Sensitivity Study

Impact on Quality: The prior *Inter-Holo* scheme captures the small-region of eye focus to approximate the hologram outside of RoF, and the proposed *Intra-Holo* takes advantage of the sparse computation required for the far objects to reduce the number of depth planes. To study how these approximation decisions affect the hologram video quality, we next want to reconstruct/render the hologram from our design based on the real-time eye movements and head orientations, and compare the quality of the reconstructed images against the baseline using the peak signal-to-noise ratio (PSNR) [21, 44] metric. Given the lack of the physical optical holographic displays (e.g., the prototype built in Tensor Holography project [54]), we numerically generate the reconstructed holographic images on top of the OpenHolo library [18]. Three demo examples of reconstructed images by OpenHolo are shown in Fig. 9: viewing a whole-hologram from different pupil positions in Fig. 9a; viewing an entire hologram (in Fig. 9b); and a partial

[5]The data is from our estimation based on [51], rather than real-hardware measurements.



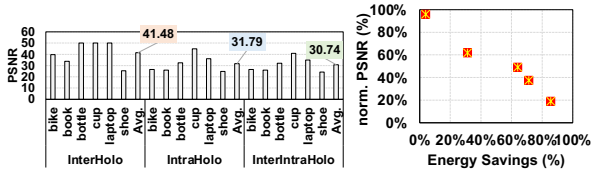
(a) Viewing W-CGH from different eye-center positions. Eye-center coordinates from left to right: `(0, 11mm)`, `(0, 12mm)`, `(0, 13mm)`, and `(0, 14mm)`.



(b) Viewing W-CGH from different focal distances. Focal distance from left to right: `0.3m`, `0.4m`, `0.5m`, and `0.6m`.

(c) Viewing S-CGH from different focal distances. Focal distance from left to right: `0.3m`, `0.4m`, `0.5m`, and `0.6m`

Figure 9: A demo of viewing/rendering the virtual planet whole-hologram (W-CGH, generated from all of the depth planes, i.e., from `1-st` to `16-th`) or sub-hologram (S-CGH, generated from only a subset of the depth planes, from `9-th` to `12-th` in this case) with different configurations. (a): Viewing the W-CGH from different eye-center positions. (b) Viewing the W-CGH from different focal distances. (c): Viewing the S-CGH from different focal distances.



(a) PSNR. **(b) Trade-offs.**

Figure 10: Sensitivity studies.

hologram (in Fig. 9c) from different distances. Compared to the baseline, we then report the averaged PSNR [21, 44] of the reconstructed images from the six videos in Fig. 10a. It can be observed from this figure that, even with the most aggressive approximation introduced by *Inter-Intra-Holo*, the video quality is still sufficient for most of the AR applications (30.7 on average) [57].

Further, to study how the tuned approximation (in Algo. 2 and Algo. 3) affect the energy savings achieved, we report five design points in Fig. 10b. This figure shows a clear pattern of trade-offs between more-energy-savings vs. more-quality-drop.

Generality of *HoloAR*: Although the core idea of approximation seems to be general across many video domains, our proposal is not expected to work very well for all AR applications. Specifically, there are two classes of applications that would probably achieve only limited benefits from our approach. First, for the quality-critical applications such as AR surgery [56], ultra-high resolution/quality of holograms are typically required. In this case, offloading computations to a resource-rich cluster/cloud system would be a more reasonable design choice (instead of approximating on the edge). Second, for applications, which are motion-sensitive such as the spaceship simulation [34], the hologram computation process is required to complete faster, in order to correctly reflect the current user's eye movement and head pose in real-time. The proposed *HoloAR* on the edge GPU cannot achieve such strict latency requirement, and can cause lagging, e.g., the eye could move to another area, while the hologram is still being computed for the previous focus region. We postpone optimizations for such applications to a future work.

5.5 Future Work

Despite the hardware-agnostic nature of *HoloAR*, it is still interesting to study how to deploy our idea on an ASIC hardware, and co-design the next-generation accelerator on edge for the AR hologram. Towards this, we plan to explore three critical questions in our future work: First, how many processing units (PUs) are required and just sufficient for most of the cases in a typical AR holographic application? To answer this, we plan to characterize the number of depth planes needed in various AR applications, and guide the optimal design choices (i.e., number of PUs, frequency, input and output buffer size, etc.) based on application requirements, and evaluate both PSNR and user-experience metrics such as satisfaction and dizziness [66]. Second, How do we maintain high power efficiency of PUs during runtime? In some cases where a small amount of hologram computation required, not all of the PUs on-board are needed to be active. We plan to design and implement a clock/power gating technology to switch off the un-utilized PUs and save power/energy. Third, how do we handle the corner cases, where more computational resources than that provided by the accelerator are required? Towards this, we plan to design a system-level scheduler which can efficiently partition the hologram tasks between the heterogeneous accelerator and original execution engines such as CPUs or GPUs.

6 RELATED WORK

In this section, we summarize prior work related to different aspects of holographic processing.

Optimizations in Holographic Processing: Holographic processing has been optimized in various domains [33, 35, 52, 54], to improve power efficiency or execution performance. For example, HORN-8 [35] has proposed a special-purpose computer for electro-holography to reduce the power consumption and still deliver a high frame rate (similar to that of a cloud GPU). From the software/algorithm perspective, a sub-hologram technique is proposed with a tracked viewing-window technology to tailor the holographic computation only for the necessary information inside of the window [52]. More recent efforts have attempted to combine holographic processing with neural network techniques. For instance, DeepHolo [33] proposes a binary-weighted computer-generated hologram model to recognize 3D objects. Furthermore, another convolution neural network (CNN) model is trained and deployed on mobile devices to synthesize a photorealistic colour 3D hologram from a single RGB-depth image in real time [54]. Apart from neural network techniques, foveated rendering is another promising performance optimization for reducing computational costs [2, 22, 24, 25, 30, 47, 62], as summarized in Sec. 2.2.2. In this paper, the foveated rendering idea (denoted as *Inter-Holo* design) has been implemented (in Sec. 4.3) and found to work well (in

Sec. 5) as in prior works. Further, in this paper, we have gone beyond foveated rendering (*Inter-Holo*), by proposing an optimization/approximation called *Intra-Holo*, that complements the former in boosting performance/energy efficiency. This enhancement is ideally suited for holographic processing at the edge, without requiring additional hardware, cloud assistance, or machine learning framework.

Holographic Displays on AR: Another large body of prior works focus on optimizing the holographic displays for the next-generation AR headsets [6, 17, 23]. For example, Michelson proposes a holographic display technology that optimizes image quality for emerging near-eye displays using two SLMs and camera-in-the-loop calibration [7]. Neural-Holography proposes an algorithmic hologram generation framework that uses camera-in-the-loop training to achieve unprecedented image fidelity and real-time frame rates [48]. OLAS proposes an overlap-add stereogram algorithm, which uses overlapping hogels to encode the view-dependent lighting effects of a light field into a hologram, achieving better quality than other holographic stereograms [46]. These display quality optimizations are orthogonal to our approximation-based proposal, and our approach can be used along with such optimizations.

Volumetric Video Streaming, Compression, and Other Optimizations: Volumetric sensor inputs such as LiDAR have large volume and require significant computational power and bandwidth to process/transmit. Targeting them, prior efforts have proposed to optimize their compression ratio, processing performance, and energy efficiency [8–10, 16, 27, 67–70]. For example, ASV leverages characteristics unique to stereo vision and proposes algorithmic and computational optimizations to improve performance and energy-efficiency of "depth from stereo" [11]. Tigris proposes an algorithm-architecture co-design system specialized for point cloud registration, to improve real-time performance and energy efficiency for 3D perception applications [65]. To efficiently steam volumetric video to mobile devices, GROOT proposes a novel PD-Tree data structure and streams the volumetric videos at a 30fps frame rate with minimal memory usage and computation for decoding [27]. Note, however, that none of these existing schemes target at reducing the amount of "unnecessary" computations in the AR holographic applications. In addition to the *Inter-Holo* design, our proposed *Intra-Holo* technique focuses on computation approximation opportunities, and as such, it is orthogonal to these prior efforts.

7 CONCLUSION

The extremely heavy computation in hologram processing hinders the growth of the 3D display applications on AR headsets. Thus, prior efforts have proposed using accelerators or cloud for optimizing the hologram computation. In contrast, this paper attempts to exploit available approximation opportunities unique in AR holographic applications, and proposes a two-stage *HoloAR* scheme to speed up the execution and save energy. Specifically, we leverage the existing foveated rendering in *Inter-Holo* to track the user's eye movements and approximate the holograms of the objects that are outside the user interest. We also propose *Intra-Holo* to further approximate each of the object holograms, by analyzing its current distance from the user. Our experimental results show that, compared to the baseline, *HoloAR* achieves 2.7× speedup and 73% energy savings. We believe that the lessons learned from this work will help in designing next-generation hologram accelerators that can combine approximation as well as other optimizations such as tuning PU counts, frequency, and power gating for achieving the target performance and energy efficiency for edge devices.

ACKNOWLEDGMENTS

This research is supported in part by NSF grants #1763681, #1629915, #1629129, #1317560, #1526750, #1714389, #1912495, and #1909004. This work was also supported in part by CRISP, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. We would also like to thank Dr. Jack Sampson and Dr. Dinghao Wu for their feedback on this paper.

REFERENCES

[1] Adel Ahmadyan, Liangkai Zhang, Jianing Wei, Artsiom Ablavatski, and Matthias Grundmann. 2020. Objectron: A Large Scale Dataset of Object-Centric Videos in the Wild with Pose Annotations. *arXiv preprint arXiv:2012.09988* (2020).

[2] Rachel Albert, Anjul Patney, David Luebke, and Joohwan Kim. 2017. Latency Requirements for Foveated Rendering in Virtual Reality. *ACM Trans. Appl. Percept.* (2017).

[3] ARCore. 2020. Using Scene Viewer to Display Interactive 3D Models in AR from an Android App or Browser. "https://developers.google.com/ar/develop/java/scene-viewer".

[4] Stephen A Benton and V Michael Bove Jr. 2008. *Holographic Imaging*. John Wiley & Sons.

[5] BusinessofApps. 2020. Pokémon GO Revenue and Usage Statistics. "https://www.businessofapps.com/data/pokemon-go-statistics/".

[6] Chenliang Chang, Kiseung Bang, Gordon Wetzstein, Byoungho Lee, and Liang Gao. 2020. Toward the Next-generation VR/AR Optics: A Review of Holographic Near-eye Displays from a Human-centric Perspective. *Optica* (2020), 1563–1578.

[7] Suyeon Choi, Jonghyun Kim, Yifan Peng, and Gordon Wetzstein. 2021. Optimizing image quality for holographic near-eye displays with Michelson Holography. *Optica* (2021), 143–146.

[8] Yu Feng, Patrick Hansen, P. Whatmough, Guoyu Lu, and Yuhao Zhu. 2021. A LiDAR-Guided Framework for Video Enhancement. *ArXiv* (2021).

[9] Y. Feng, Shaoshan Liu, and Yuhao Zhu. 2020. Real-Time Spatio-Temporal LiDAR Point Cloud Compression. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), 10766–10773.

[10] Yu Feng, Boyuan Tian, Tiancheng Xu, Paul Whatmough, and Yuhao Zhu. 2020. Mesorasi: Architecture Support for Point Cloud Analytics via Delayed-aggregation. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 1037–1050.

[11] Yu Feng, Paul Whatmough, and Yuhao Zhu. 2019. ASV: Accelerated Stereo Vision System. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 643–656.

[12] gazepoint. 2020. Eye Tracking and Neuromarketing Research Made Easy. "https://www.gazept.com/".

[13] Patrick Geneva, Kevin Eckenhoff, Woosik Lee, Y. Yang, and Guoquan Huang. 2020. OpenVINS: A Research Platform for Visual-Inertial Estimation. *2020 IEEE International Conference on Robotics and Automation (ICRA)* (2020), 4666–4672.

[14] Giorgia Lombardo. 2020. Meet the Humans of the Future: Holograms, Digital Humans, and Deep Fakes. "https://medium.com/demagsign/meet-the-humans-of-the-future-holograms-digital-humans-and-deep-fakes-35024b881545".

[15] Stuart Golodetz, Michael Sapienza, Julien Valentin, Vibhav Vineet, Ming-Ming Cheng, Anurag Arnab, Victor Adrian Prisacariu, Olaf Kaehler, Carl Yuheng Ren, David W. Murray, Shahram Izadi, and Philip H.S. Torr. 2015. SemanticPaint: A Framework for the Interactive Segmentation of 3D Scenes. *arXiv* (2015).

[16] Bo Han, Yu Liu, and Feng Qian. 2020. ViVo: Visibility-aware Mobile Volumetric Video Streaming. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*. 1–13.

[17] Zehao He, Xiaomeng Sui, Guofan Jin, and Liangcai Cao. 2019. Progress in Virtual Reality and Augmented Reality Based on Holographic Display. *Appl. Opt.* (2019), A74–A81.

[18] Jisoo Hong, Youngmin Kim, Hyunjoo Bae, and Sunghee Hong. 2020. OpenHolo: Open Source Library for Hologram Generation, Reconstruction and Signal Processing. In *Imaging and Applied Optics Congress*. Optical Society of America, HF3G.1.

[19] Muhammad Huzaifa, Rishi Desai, Samuel Grayson, Xutao Jiang, Ying Jing, Jae Lee, Fang Lu, Yihan Pang, Joseph Ravichandran, Finn Sinclair, Boyuan Tian, Hengzhi Yuan, Jeffrey Zhang, and Sarita V. Adve. 2021. Exploring Extended Reality with ILLIXR: A new Playground for Architecture Research. arXiv:cs.DC/2004.04643

[20] IFIXIT. 2020. Magic Leap One Teardown. "https://www.ifixit.com/Teardown/Magic+Leap+One+Teardown/112245".

[21] NATIONAL INSTRUMENTS. 2019. Peak Signal-to-Noise Ratio as an Image Quality Metric. "https://www.ni.com/en-us/innovations/white-papers/11/peak-signal-to-noise-ratio-as-an-image-quality-metric.html".

[22] Yeon-Gyeong Ju and Jae-Hyeung Park. 2018. Fast Generation of Mesh Based CGH in Head-Mounted Displays using Foveated Rendering Technique, In Imaging and Applied Optics 2018 (3D, AO, AIO, COSI, DH, IS, LACSEA, LS&C, MATH, pcAOP). *Imaging and Applied Optics 2018 (3D, AO, AIO, COSI, DH, IS, LACSEA, LS&C, MATH, pcAOP)*, DTu5F.6.

[23] Daniel K Nikolov, Sifan Ye, Sydney Dlhopolsky, Zhen Bai, Yuhao Zhu, and Jannick P Rolland. 2020. Hyperion: A 3D Visualization Platform for Optical Design of Folded Systems. *Frameless* 2, 1 (2020), 21.

[24] Anton Kaplanyan, Anton Sochenov, Thomas Leimkühler, Mikhail Okunev, T. Goodall, and Gizem Rufo. 2019. DeepFovea: Neural Reconstruction for Foveated Rendering and Video Compression using Learned Statistics of Natural Videos. *ACM Trans. Graph.* (2019), 212:1–212:13.

[25] Jonghyun Kim, Youngmo Jeong, Michael Stengel, Kaan Akşit, Rachel Albert, Ben Boudaoud, Trey Greer, Joohwan Kim, Ward Lopes, Zander Majercik, Peter Shirley, Josef Spjut, Morgan McGuire, and David Luebke. 2019. Foveated AR: Dynamically-Foveated Augmented Reality Display. *ACM Trans. Graph.* (2019).

[26] Joohwan Kim, Michael Stengel, Alexander Majercik, Shalini De Mello, David Dunn, Samuli Laine, Morgan McGuire, and David Luebke. 2019. Nvgaze: An Anatomically-informed Dataset for Low-latency, Near-eye Gaze Estimation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–12.

[27] Kyungjin Lee, Juheon Yi, Youngki Lee, Sunghyun Choi, and Young Min Kim. 2020. GROOT: A Real-time Streaming System of High-fidelity Volumetric Videos. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*. 1–14.

[28] T. Lee and T. Hollerer. 2008. Hybrid Feature Tracking and User Interaction for Markerless Augmented Reality. In *2008 IEEE Virtual Reality Conference*. 145–152.

[29] Magic Leap. 2020. Magic Leap 1 is a Wearable Computer for Enterprise Productivity. "https://www.magicleap.com/en-us/magic-leap-1".

[30] Xiaoxu Meng, Ruofei Du, and Amitabh Varshney. 2020. Eye-dominance-guided Foveated Rendering. *IEEE Transactions on Visualization and Computer Graphics* (2020), 1972–1980.

[31] Microsoft. 2020. HoloLens 2 Tech Specs. "https://www.microsoft.com/en-us/p/hololens-2/91pnzzznzwcp/?activetab=pivot:techspecstab".

[32] Microsoft Research Blog. 2020. Second Version of HoloLens HPU will Incorporate AI Coprocessor for Implementing DNNs. "https://www.microsoft.com/en-us/research/blog/second-version-hololens-hpu-will-incorporate-ai-coprocessor-implementing-dnns/".

[33] Naoya Muramatsu, Chun Wei Ooi, Yuta Itoh, and Yoichi Ochiai. 2017. DeepHolo: Recognizing 3D Objects Using a Binary-Weighted Computer-Generated Hologram. In *SIGGRAPH Asia 2017 Posters*.

[34] NASA. 2020. NASA at Home – Virtual Tours and Apps. "https://www.nasa.gov/nasa-at-home-virtual-tours-and-augmented-reality".

[35] Takashi Nishitsuji, Yota Yamamoto, Takashige Sugie, Takanori Akamatsu, Ryuji Hirayama, Hirotaka Nakayama, Takashi Kakue, Tomoyoshi Shimobaba, and Tomoyoshi Ito. 2018. Special-purpose Computer HORN-8 for Phase-type Electroholography. *Opt. Express* (2018), 26722–26733.

[36] Nvidia. 2019. JETSON AGX XAVIER AND THE NEW ERA OF AUTONOMOUS MACHINES. "http://info.nvidia.com/rs/156-OFN-742/images/Jetson_AGX_Xavier_New_Era_Autonomous_Machines.pdf".

[37] Nvidia. 2020. CUDA Toolkit Documentation: Nvprof. "shorturl.at/zEFU5".

[38] Objectron. 2020. Objectron Dataset Annotation: bike. "https://github.com/google-research-datasets/Objectron/blob/master/index/bike_annotations".

[39] Objectron. 2020. Objectron Dataset Annotation: book. "https://github.com/google-research-datasets/Objectron/blob/master/index/book_annotations".

[40] Objectron. 2020. Objectron Dataset Annotation: bottle. "https://github.com/google-research-datasets/Objectron/blob/master/index/bottle_annotations".

[41] Objectron. 2020. Objectron Dataset Annotation: cup. "https://github.com/google-research-datasets/Objectron/blob/master/index/cup_annotations".

[42] Objectron. 2020. Objectron Dataset Annotation: laptop. "https://github.com/google-research-datasets/Objectron/blob/master/index/laptop_annotations".

[43] Objectron. 2020. Objectron Dataset Annotation: shoe. "https://github.com/google-research-datasets/Objectron/blob/master/index/shoe_annotations".

[44] OpenCV. 2019. Similarity check (PNSR and SSIM) on the GPU. "https://docs.opencv.org/2.4/doc/tutorials/gpu/gpu-basics-similarity/gpu-basics-similarity.html".

[45] OpenHolo. 2020. OpenHolo Database. "http://openholo.org/database/depth".

[46] Nitish Padmanaban, Yifan Peng, and Gordon Wetzstein. 2019. Holographic Near-Eye Displays Based on Overlap-Add Stereograms. *ACM Trans. Graph.* (2019).

[47] Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron Lefohn. 2016. Towards Foveated Rendering for Gaze-Tracked Virtual Reality. *ACM Trans. Graph.* (2016).

[48] Yifan Peng, Suyeon Choi, Nitish Padmanaban, Jonghyun Kim, and Gordon Wetzstein. 2020. Neural Holography. In *ACM SIGGRAPH 2020 Emerging Technologies (SIGGRAPH '20)*. Association for Computing Machinery.

[49] Martin Persson, David Engström, and Mattias Goksör. 2011. Real-time Generation of Fully Optimized Holograms for Optical Trapping Applications. In *Optical Trapping and Optical Micromanipulation VIII*, Vol. 8097. International Society for Optics and Photonics, 80971H.

[50] Victor Adrian Prisacariu, Olaf Kähler, Stuart Golodetz, Michael Sapienza, Tommaso Cavallari, Philip H. S. Torr, and David William Murray. 2017. InfiniTAM v3: A Framework for Large-Scale 3D Reconstruction with Loop Closure. *CoRR* (2017).

[51] Murad Qasaimeh, Kristof Denolf, Jack Lo, Kees A. Vissers, Joseph Zambreno, and Phillip H. Jones. 2019. Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels. In *15th IEEE International Conference on Embedded Software and Systems*. 1–8.

[52] Stephan Reichelt, Ralf Haussler, Norbert Leister, Gerald Futterer, Hagen Stolle, and Armin Schwerdtner. 2010. *Holographic 3-D Displays - Electro-holography Within the Grasp of Commercialization*. IntechOpen. shorturl.at/jmnpD

[53] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. 2020. Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

[54] Liang Shi, Beichen Li, Changil Kim, Petr Kellnhofer, and Wojciech Matusik. 2021. Towards Real-time Photorealistic 3D Holography with Deep Neural Networks. *Nature* 592 (2021).

[55] Tomoyoshi Shimobaba, Jiantong Weng, Takahiro Sakurai, Naohisa Okada, Takashi Nishitsuji, Naoki Takada, Atsushi Shiraki, Nobuyuki Masuda, and Tomoyoshi Ito. 2012. Computational Wave Optics Library for C++: CWO++ Library. *Computer Physics Communications* (2012), 1124–1138.

[56] Jeffrey H. Shuhaiber. 2004. Augmented Reality in Surgery. *Archives of Surgery* (2004), 170–174.

[57] Randall Shumaker and Lackey Stephanie. 2014. *Virtual, Augmented and Mixed Reality: Designing and Developing Augmented and Virtual Environments: 6th International Conference, VAMR 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014, Proceedings, Part I*. Vol. 8525. Springer.

[58] Julian Steil, Inken Hagestedt, Michael Xuelin Huang, and Andreas Bulling. 2019. Privacy-Aware Eye Tracking Using Differential Privacy. In *Proc. ACM International Symposium on Eye Tracking Research and Applications (ETRA)*. 1–9.

[59] Stereolabs. 2020. ZED Software Development Kit. "https://www.stereolabs.com/developers/release/".

[60] techradar. 2020. Google Pixel 2 Review. "https://www.techradar.com/reviews/google-pixel-2-review".

[61] Oren M Tepper, Hayeem L Rudy, Aaron Lefkowitz, Katie A Weimer, Shelby M Marks, Carrie S Stern, and Evan S Garfein. 2017. Mixed Reality with HoloLens: Where Virtual Reality Meets Augmented Reality in the Operating Room. *Plastic and reconstructive surgery* (2017), 1066–1070.

[62] Lingjie Wei and Yuji Sakamoto. 2019. Fast Calculation Method with Foveated Rendering for Computer-generated Holograms Using an Angle-changeable Ray-tracing Method. *Appl. Opt.* (2019), A258–A266.

[63] Yang Wu, Jun Wang, Chun Chen, Chan-Juan Liu, Feng-Ming Jin, and Ni Chen. 2021. Adaptive Weighted Gerchberg-Saxton Algorithm for Generation of Phase-only Hologram with Artifacts Suppression. *Opt. Express* (2021), 1412–1427.

[64] XILINX. 2020. Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit. "https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html".

[65] Tiancheng Xu, Boyuan Tian, and Yuhao Zhu. 2019. Tigris: Architecture and Algorithms for 3D Perception in Point Clouds. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 629–642.

[66] Hiroshi Yoshikawa, Takeshi Yamaguchi, and Hiroki Uetake. 2016. Image Quality Evaluation and Control of Computer-generated Holograms. In *Practical Holography XXX: Materials and Applications*, Hans I. Bjelkhagen and V. Michael Bove Jr. (Eds.). International Society for Optics and Photonics, SPIE, 144 – 152.

[67] Anlan Zhang, Chendong Wang, Bo Han, and Feng Qian. 2021. Efficient Volumetric Video Streaming Through Super Resolution. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*. 106–111.

[68] Haibo Zhang, Prasanna Venkatesh Rengasamy, Shulin Zhao, Nachiappan Chidambaram Nachiappan, Anand Sivasubramaniam, Mahmut T. Kandemir, Ravi Iyer, and Chita R. Das. 2017. Race-to-Sleep + Content Caching + Display Caching: A Recipe for Energy-Efficient Video Streaming on Handhelds. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 517–531.

[69] Haibo Zhang, Shulin Zhao, Ashutosh Pattnaik, Mahmut T. Kandemir, Anand Sivasubramaniam, and Chita R. Das. 2019. Distilling the Essence of Raw Video to Reduce Memory Usage and Energy at Edge Devices. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. 657–669.

[70] Shulin Zhao, Haibo Zhang, Sandeepa Bhuyan, Cyan Subhra Mishra, Ziyu Ying, Mahmut T. Kandemir, Anand Sivasubramaniam, and Chita R. Das. 2020. Déjà View: Spatio-Temporal Compute Reuse for Energy-Efficient 360° VR Video Streaming. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 241–253.