2024 IEEE Real-Time Systems Symposium (RTSS) | 979-8-3315-4026-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/RTSS62706.2024.00016

BOXR: Body and head motion Optimization framework for eXtended Reality

Ziliang Zhang, Zexin Li, Hyoseung Kim, Cong Liu
University of California, Riverside
{zzhan357, zli536, hyoseung, congl}@ucr.edu

***Abstract*—The emergence of standalone Extended Reality (XR) systems has enhanced user mobility, accommodating both subtle, frequent head motions and substantial, less frequent body motions. However, the pervasively used Motion-to-Display (M2D) latency metric, which measures the delay between the most recent motion and its corresponding display update, only accounts for head motions. This oversight can leave users prone to motion sickness if significant body motion is involved. Although existing methods optimize M2D latency through asynchronous task scheduling and reprojection methods, they introduce challenges like resource contention between tasks and outdated pose data. These challenges are further complicated by user motion dynamics and scene changes during runtime.**

To address these issues, we for the first time introduce the Camera-to-Display (C2D) latency metric, which captures the delay caused by body motions, and present BOXR, a framework designed to co-optimize both body and head motion delays within an XR system. BOXR enhances the coordination between M2D and C2D latencies by efficiently scheduling tasks to avoid contentions while maintaining an up-to-date pose in the output frame. Moreover, BOXR incorporates a motion-driven visual inertial odometer to adjust to user motion dynamics and employs scene-dependent foveated rendering to manage changes in the scene effectively. Our evaluations show that BOXR significantly outperforms state-of-the-art solutions in 11 EuRoC MAV datasets across 4 XR applications across 3 hardware platforms. In controlled motion and scene settings, BOXR reduces M2D and C2D latencies by up to 63% and 27%, respectively and increases frame rate by up to 43%. In practical deployments, BOXR achieves substantial reductions in real-world scenarios—up to 42% in M2D latency and 31% in C2D latency—while maintaining remarkably low miss rates of only 1.6% for M2D requirements and 1.0% for C2D requirements.

I. Introduction

Extended Reality (XR), which encompasses virtual reality, augmented reality, and mixed reality, offers an immersive virtual-physical experience to the user [1, 2]. An XR system captures the user's motion through various sensors such as an inertial measurement unit (IMU) and a camera (CAM), processes the sensor data to generate a new image frame according to the user's motion and surrounding scene changes, and displays the generated frame on a head-mounted display.

As evident by prior research, the delay between a motion's capture and the motion's display in the output frame can cause perception misalignment [3, 4]. Existing systems address this by focusing on optimizing the Motion-to-Display latency (M2D) metric, which measures the time it takes to display the latest motion. Despite extensive optimization of M2D, as illustrated in the left subfigure of Fig. 1, the virtual user interface of Apple Vision Pro [2] drifts towards the opposite direction of movement when the user makes significant movements during snowboarding [5]. This significantly hinders the quality of the immersive experience and can cause motion sickness.

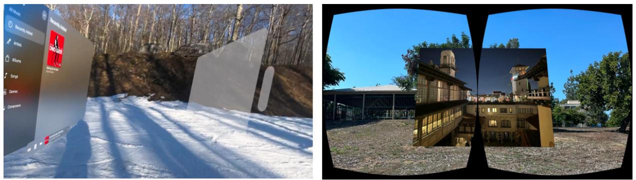

(A) Vision Pro UI drifted during snowboarding
(B) Our Image follows user during running

Fig. 1: Comparison between the Vision Pro [5] and BOXR when large body motion exists

We find that the fundamental reason behind this problem lies in the imprecise distinction of "motion". The user's motion recognized by an XR system can be categorized into two types based on the required sensors. The first type of motion includes frequent rotational movements or small-magnitude translational movements that do not involve position changes within the environment and thus can be reliably captured by IMU. Since this motion primarily involves head movement, we refer to it as *head motion*. The second type of motion involves less frequent but more extensive displacement, specifically movements that require camera-based localization in the environment to capture absolute position changes, such as walking in a room. Because this motion results in changes in the user's body posture, we refer to it as *body motion*. The M2D metric widely used in current studies represents *only the head motion* captured by the IMU sensor running at a high sampling rate (e.g., 500 Hz). Consequently, body motion, which requires slower camera updates (e.g., 20 Hz), remains unaccounted for in M2D assessments. Therefore, even with significant optimization of M2D by Vision Pro [6] and state-of-the-art research [7–9], the interface continues to encounter drift caused by the substantial delay of body motion in the output frame.

After a thorough analysis of the end-to-end XR execution pipeline, we identified the root cause of why the M2D metric alone cannot capture the delay between body motion detection and its display. To address this, we define and introduce another key metric: Camera-to-Display latency (C2D) which is essential for accurately capturing this delay. However, co-optimizing M2D and C2D in an XR system involves several challenges. First, state-of-the-art XR frame-

979-8-3315-4026-5/24/$31.00 ©2024 IEEE
DOI 10.1109/RTSS62706.2024.00016

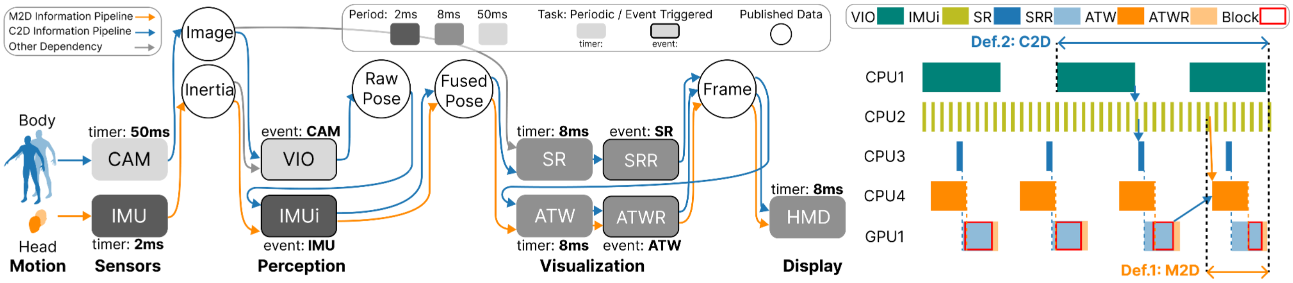


Fig. 2: Left: ILLIXR, an open-sourced state-of-the-art XR system, involves two types of motion processing sequences for the final display. Right: Example schedule of XR tasks. Blue arrows show C2D and orange arrows show M2D sequences.

works like ILLIXR [10] schedule tasks asynchronously and use reprojection-based methods such as asynchronous time-warp [7, 8] to optimize M2D. However, there is contention between the reprojection task required by M2D and the scene rendering task essential to C2D, as both need to compete for the shared GPU. Straightforward solutions include adjusting the periodicity of tasks to simultaneously improve M2D and C2D cannot eliminate contention, resulting in greater fluctuations in both metrics and exacerbating motion sickness. Additionally, the dynamic nature of user motion and scene changes introduces significant variations in task execution times, further complicating this issue.

Contributions. To effectively co-optimize M2D and C2D, we present **BOXR**, a **B**ody and head motion delay **O**ptimization framework for e**X**tended **R**eality systems. BOXR incorporates our discovery of C2D based on the delay of body motion and addresses the unique challenges due to co-optimization. First, BOXR proposes a contention-preventive scheduling policy to eliminate contention between rendering and reprojection tasks. Additionally, it designs an on-demand IMUi to minimize IMU wasted work during execution. This approach ensures consistently low M2D and C2D latencies, through efficiently managing task execution sequences to prevent unnecessary delays and conflicts. Built upon the scheduling policy, BOXR designs a *motion-driven* visual inertial odometer (VIO) which dynamically adapts the feature extraction frontend to the motion dynamics, correcting additional position errors with an error-bounding method. The intriguing aspect of this design is how it intelligently adapts in real-time to the user's motions, maintaining VIO execution times within the set budget without significantly increasing position errors. Additionally, BOXR implements Scene-Dependent Foveated Rendering, a technique that dynamically adjusts the foveation area according to the number of objects in the scene. By centering foveation on the centroid of objects, BOXR manages to maintain high frame quality while controlling render times. Our intuition is to prioritize rendering resources where they matter most, based on real-time scene analysis.

We implement BOXR based on the ILLIXR framework [10] and test four popular XR applications across three hardware settings with eleven trajectories[1]. As shown in Fig. 1, BOXR consistently maintains the interface at the center of the viewport that closely aligns with user movement when significant body motion occurs. We conduct evaluations using both controlled trajectory datasets with diverse XR application scenes and real-world scenarios featuring varied motion levels. Compared to the state-of-the-art, BOXR achieves:

- **Effectiveness:** BOXR reduces up to 63% of M2D and 27% of C2D while increasing frame rate by up to 43% across three hardware platforms under different applications and trajectories.
- **Robustness:** In extremely dynamic motion and scene scenarios, BOXR experiences at most 10% M2D increase and 6% C2D increase, leading to stable usability.
- **Applicability:** When deploying in real-world scenarios, BOXR achieves up to 42% M2D reduction and 31% C2D reduction, resulting in only 1.6% M2D miss rate and 1.0% C2D miss rate across three applications.

II. Background

A. XR System Model

Recent XR systems involve various sensing and algorithmic tasks. In this subsection, we give a brief description of these tasks and their execution workflow in ILLIXR [10], which is a representative state-of-the-art open-source XR framework. Fig. 2 illustrates the overview of the ILLIXR system.

To maximize utilization and increase throughput, ILLIXR adopts a publisher-subscriber model, where each task runs asynchronously as shown in Fig. 2 right [7, 10]. Sensor readings from the IMU and CAM are published onto image and inertia topics and pipelined into the *perception phase* consisting of VIO and IMUi tasks:

- Visual Inertial Odometer (VIO): The VIO task subscribes to both CAM image and IMU inertia topics, but executes only when the CAM topic arrives, consequently getting the same period as CAM. VIO extracts features and conducts pose estimation based on Multi-State Constraint Kalman Filter (MSCKF) algorithm [11]. VIO publishes as output the estimated pose capturing the latest body movement.
- IMU integration (IMUi): The IMUi task is triggered whenever a new IMU sensor reading arrives [12]. IMUi extrapolates the last pose from VIO to the arrival time of the IMU data, in order to compensate for the slow update rate of CAM and the long processing time of VIO. We call the output of IMUi a fused pose since it captures body motion

[1] Our implementation can be found at: https://github.com/rtenlab/BOXR.git

from the raw pose as well as head motion using the latest inertia data from IMU.

Next, the *visualization phase* involves the following tasks:

- Scene Reconstruction (SR): The SR task employs the fused pose from the perception phase and calculates the viewport that determines the display area at the default 120 FPS target frame rate, which is equivalent to an 8 ms period.
- Scene Reconstruction Rendering (SRR): The completion of the SR task activates the SRR task. SRR renders a 2D frame within the viewport provided by SR.
- Asynchronous Timewarp (ATW): The ATW task leverages the up-to-date fused pose published after SRR to generate the user's view matrix, which is the inverse of the camera transformation matrix that contains the complete 3D information. Determined by the default target frame rate, ATW is also scheduled with the 8 ms period and runs asynchronously to SR and SRR.
- Asynchronous Timewarp Reprojection (ATWR): Upon completion of ATW, the ATWR task reprojects the 2D frame based on the view matrix from ATW and completes the final 3D frame.

During the execution, VIO, IMUi, SR, and ATW only use the CPU for computation, whereas SRR and ATWR primarily use the GPU for rendering and projection. Each of these tasks is a separate thread and they run concurrently unless there is any explicit input dependency explained above. In particular, SRR and ATWR are designed to share a single GPU stream because they access the same memory region of the frame buffer; hence, they are executed sequentially on the GPU in a first-in-first-out order [13, 14].

B. Characterizing latency metrics

Since humans are highly perceptive of motion delay, any delay between the latest user motion and the displayed output that exceeds 20 ms leads to perception misalignment in an XR system [3, 4]. Within the context of XR, the latest 'head' motion is consistently captured by IMU due to its high sample rate compared to CAM. In the existing work [3, 7, 8, 10], this delay is quantified using the following metric:

Def. 1 (Motion-to-Display Latency)**.** The motion-to-display latency (M2D) is defined as the time interval between capturing of the latest motion by the IMU and its corresponding display on the HMD. In other words, M2D is the time to complete the IMUi→ATW→ATWR sequence, e.g., orange arrows in both sub-diagrams of Fig. 2.

However, as M2D does not distinguish between different types of motion, it always begins from the most recent IMU sensor input that captures only head motion. Consequently, even if the state-of-the-art XR system achieves satisfactory M2D performance [2, 6], users still experience noticeable perception misalignment when engaging in significant body motion, as we showed with Fig. 1(A). To better address the perception misalignment and effectively differentiate between delays caused by IMU-captured head motion and CAM-captured body motion, we introduce the following:

Def. 2 (Camera-to-Display Latency)**.** The camera-to-display latency (C2D) is defined as the time interval between capturing the latest body motion by CAM and its corresponding display on the HMD.[2] C2D is therefore the time to complete the sequence of VIO→IMUi→SR→SRR→ATW→ATWR, e.g., blue arrows in both sub-diagrams of Fig. 2.

C2D begins with VIO which generates a raw pose. IMUi then calculates to produce a fused pose that encompasses both head and body motion. Subsequently, SR and SRR utilize this fused pose to render the 2D frame, which ATW and ATWR further reproject to produce the final 3D frame output. Given that users typically perceive displacement through changes of objects in the scene, we set the C2D requirement to 80 ms, a threshold commonly applied in video games to ensure steady updates of virtual objects [1, 16].

III. Challenges

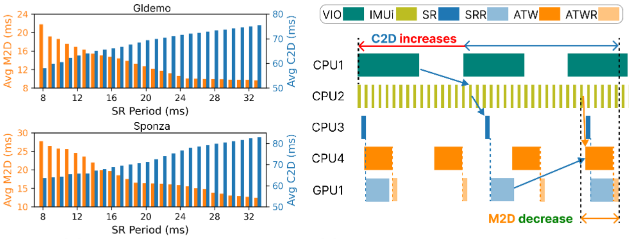
A. Challenges for M2D and C2D optimization

To understand the correlation between M2D and C2D, we will discuss notable performance bottlenecks and why simple methods that are seemingly positive in reducing latencies do not work in XR systems. We conducted experiments using two prevalent XR applications: Gldemo [10], which has lighter rendering demands, and Sponza [17], known for its intensive rendering requirements. Both applications are implemented on ILLIXR [10]. Only a PC platform is used in this section but our evaluation in Sec. V includes two embedded platforms as well. We used the default target frame rate of 120 FPS set by ILLIXR, which gives the task periods shown in Fig. 2.

We first analyze the effect of ATW and ATWR tasks. In existing XR systems, ATW and ATRW are known to be effective in reducing M2D, thanks to their asynchronous execution to SR and SRR [7, 8, 10]. The right side of Fig. 2 gives an example schedule. The execution time of SR is much smaller than that of ATW because SR calculates only one viewport position but ATW performs the entire matrix transformation based on the latest fused pose. Conversely, SRR takes longer to execute than ATWR because SRR renders a whole 2D frame based on the viewport from SR but ATWR only reprojects the scene with the depth information from ATW. Also, we can observe from the figure that the execution of SRR and ATRW does not overlap (highlighted by red blocks). This is because they share the frame buffer memory, requiring them to contend for the same GPU stream and execute sequentially.

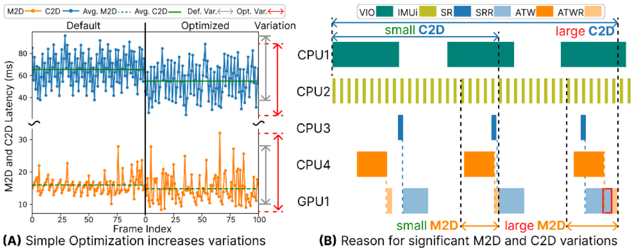
To mitigate such GPU contention for ATWR and improve M2D, one may consider increasing the period of SR, thereby decreasing the number of SRR jobs and reducing the blocking time caused by contention. However, Fig. 3(A) demonstrates that optimizing M2D by increasing the SR period leads to a larger C2D. This phenomenon is further elucidated in Fig. 3(B). As the SR period is increased, the last ATWR job receives the raw pose from the first VIO job, which is outdated

[2] It is worth noting that C2D differs from the photon-to-photon latency [15] used in some literature since it only measures the time for see-through display of CAM images on the HDM, without involving the pose estimation of VIO.



(A) Tradeoff for M2D and C2D **(B)** Increasing SR period leads to worse C2D

Fig. 3: Negative impact of improving M2D on C2D



(A) Simple Optimization increases variations **(B)** Reason for significant M2D and C2D variations

Fig. 4: Effects of simple co-optimization using task timers

since the second VIO has already finished execution. The use of this outdated raw pose causes a significant C2D increase, the amount of which is illustrated by the red arrow in Fig. 3(B).

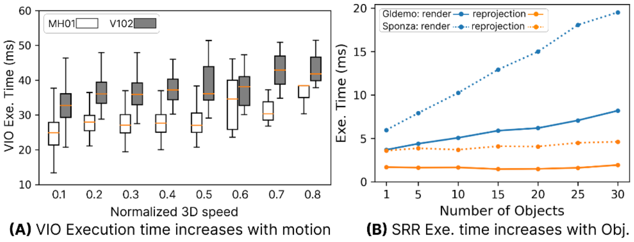
Obs. 1. Solely optimizing M2D latency by minimizing contention may lead to the deterioration of C2D because of the outdated raw pose caused by reduced SR workload. This necessitates the consideration of both M2D and C2D during optimization.

A better approach might be adjusting the periods of both SR and ATW. However, such an approach could have side effects as detailed in Fig. 4(A). It shows the profiling of M2D and C2D in 100 frames output when running Gldemo with default scheduling and with the optimized scheduling that achieved the smallest average-case M2D and C2D among various combinations of SR and ATR periods. Although the optimization can achieve a lower M2D and C2D on average than the default setting, their variations are significantly increased. Compare Fig. 4(B) to the right side of Fig. 2. While changing both SR and ATW periods fixes the GPU contention and the outdated raw pose issues for the first and second ATWR jobs, the third ATWR job still suffers from the same issues. This leads to an extremely large C2D that even surpasses the M2D and C2D in Fig. 3(B). Such significant fluctuations of M2D and C2D can cause great discrepancies between the actual and perceived motions, which is the root cause of motion sickness [4].

Obs. 2. Simply adjusting the periodicities of XR tasks can cause significant fluctuations in M2D and C2D. A scheduling technique that considers the freshness of sensor data and GPU contention is necessary.

B. XR characteristics at runtime

In real-world scenarios, user motion dynamics and scene diversities contribute to significant variations in the execution time of certain tasks. We identify two characteristics (C1, C2) unique to XR systems given the runtime dynamics.



(A) VIO Execution time increases with motion **(B)** SRR Exe. time increases with Obj.

Fig. 5: XR characteristics under motion and scene dynamics.

C1: Motion-dependent Tracking. During the perception phase, the XR system processes captured user motion through the VIO task, which exhibits significant variations in execution time when exposed to a broad range of motion. To analyze how motion dynamics affect VIO execution times, we test OpenVINS [11], a widely used VIO algorithm that is standard in the ILLIXR framework, running on an embedded platform with two distinct trajectories from the EuRoc MAV dataset [18], specifically recorded in the Machine Hall (MH01) and Vicon Room (V102) environments. The execution time of VIO is recorded across a spectrum of normalized 3D speeds up to the maximum speed observed in the dataset. As shown in Fig. 5(A), VIO execution time increases with speed in both recorded trajectories. This escalation is due to the feature extraction front-end which detects more new features at higher speeds. Since VIO employs the MSCKF for the pose estimation part that calculates based on these new features, a larger number of features updated during extraction prolongs the execution time of the pose estimation, resulting in the overall increase of VIO execution time.

C2: Scene-dependent Rendering. During the visualization phase, the complete 3D frame needs SRR to render the 2D frame, which the ATWR reprojects. Therefore, we denote SRR execution time as render time and ATWR execution time as reprojection time. We investigate the correlation between the render and reprojection time when the objects in the viewport increase. We set up the experiment with only SRR, ATW, and ATWR running sequentially on the same two XR applications (Sponza and Gldemo) used in Sec. III-A. Throughout the experiment, we manipulate the number of objects by directly altering the objects in the viewport. As shown in Fig. 5(B), the render time increases linearly with the number of objects, whereas the reprojection time remains relatively constant. This discrepancy can be attributed to the workload heterogeneity of SRR and ATWR. SRR necessitates the rendering of every vertex, thereby requiring the calculation of each vertex's coordinates based on the given viewport. In contrast, ATWR reprojects only the updated vertices using the depth information from ATW without re-rendering the entire frame. Therefore, despite object changes in the scene, the impact on ATWR's execution time is minimal, whereas SRR's execution time increases linearly with the number of objects.

Obs. 3. Even if the co-optimization of M2D and C2D is

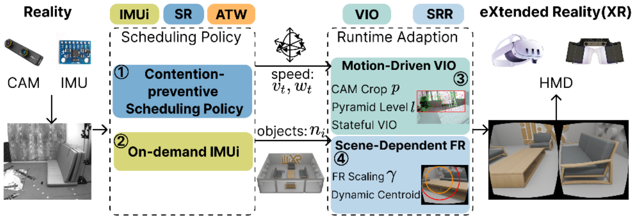


Fig. 6: BOXR Overview.

achieved, it can be easily nullified by motion dynamics (C1) and scene changes (C2). Therefore, XR systems need techniques to control the execution time fluctuations caused by these factors.

IV. Methodology

A. BOXR Overview

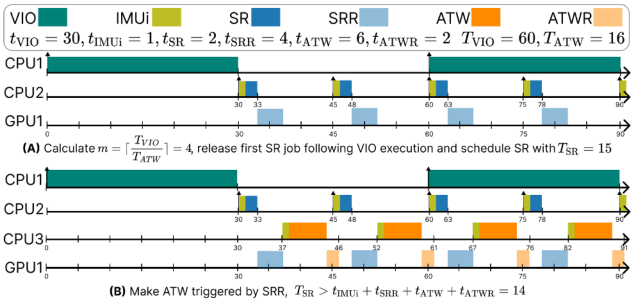
We present BOXR, a body and head motion optimization framework for XR based on the three observations from Sec. III and show its system design in Fig. 6.

Motivated by Obs. 1 and Obs. 2, the contention and outdated raw pose result in large M2D and C2D, which can be addressed with a better scheduling policy design. BOXR proposes a contention-preventive scheduling policy (① in Fig. 6) that manages task execution sequences to maintain the up-to-date raw pose while preventing contentions. It further designs an on-demand IMUi (② in Fig. 6) to mitigate the problem of dropped IMU sensor information during the execution. Through the BOXR scheduling policy, we obtain an execution sweet spot given static execution time for all tasks.

Based on Obs. 3, the execution sweet spot derived from the BOXR scheduling policy can be nullified by the motion dynamics (C1) and scene changes (C2). Therefore, BOXR introduces the following runtime adaptions that each targets at an XR characteristic. To negate the effect of C1, BOXR designs a motion-driven visual inertial odometer (MVIO) (③ in Fig. 6) that crops the input CAM image and controls the image pyramid level of the feature extraction based on the current motion magnitude. Also, to limit the extent of any resulting positional error, it includes an error bounding mechanism. BOXR then designs a scene-dependant foveated rendering (SFR) (④ in Fig. 6) that replaces the original SRR to address C2. SFR dynamically changes the foveation area according to the number of objects in the viewport and centers the foveation area to the objects centroid.

B. BOXR Scheduling Policy

In this subsection, we assume the execution time of all tasks stays constant when designing the scheduling policy. This assumption will be lifted later in Sec. IV-C and Sec. IV-D. BOXR profiles the execution time for tasks VIO, IMUi, SR, SRR, ATW, and ATWR as t_{VIO}, t_{IMUi}, t_{SR}, t_{SRR}, t_{ATW}, and t_{ATWR}, respectively, and this needs to be done only once for a target hardware platform. To maintain the given sensor sample rate and target frame rate, BOXR assumes the period of CAM (T_{CAM}) which equals the period of VIO (T_{VIO}), the period of IMU (T_{IMU}), and the period of ATW (T_{ATW}) are given in advance. Without loss of generality, they follow $T_{\text{CAM}} > T_{\text{ATW}} \gg T_{\text{IMU}}$. The scheduling policy mentioned below is executed prior to each VIO execution.



(A) Calculate $m = \lceil \frac{T_{VIO}}{T_{ATW}} \rceil = 4$, release first SR job following VIO execution and schedule SR with $T_{\text{SR}} = 15$

(B) Make ATW triggered by SRR, $T_{\text{SR}} > t_{\text{IMUi}} + t_{\text{SRR}} + t_{\text{ATW}} + t_{\text{ATWR}} = 14$

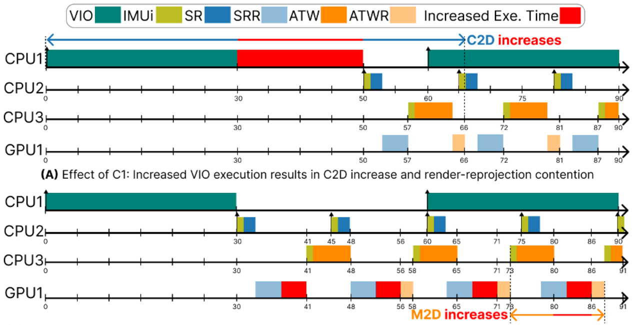
Fig. 7: BOXR scheduling example

Contention-preventive Scheduling. To address the challenges from Obs. 1 and Obs. 2, we aim to eliminate the contention between SRR and ATWR while providing an up-to-date raw pose to ATWR.

At first, to prevent unnecessary contention between ATWR and SRR in any C2D sequence, we schedule the first SR job to start its execution immediately after the completion of the VIO job. This ensures the first SR job always receives the raw pose result from the latest VIO. In addition, within each T_{VIO}, the asynchronous execution of SRR and ATWR contributes to their contention. Therefore, we schedule ATW and ATWR jobs synchronously to SR and SRR jobs such that they begin only after the completion of the SRR job.

Next, to ensure the freshness of the raw pose data in the final output frame of ATWR, we co-optimize the SR & SRR and ATW & ATWR periods with the following approach. Since $T_{\text{VIO}} > T_{\text{ATW}}$, multiple output frames from ATWR exist in each T_{VIO}. If a different time interval from SR & SRR is applied to ATW & ATWR to process each frame, the SRR job from the previous C2D instance (VIO→IMUi→SR→SRR→ATW→ATWR) may overlap with the ATWR job in the current C2D instance. This contributes to the outdated raw pose in Obs. 2, which inevitably results in large C2D variation. If an equal interval to execute each sequence is given, the ATWR and SRR overlapping can be safely avoided. Therefore, we first calculate the number of C2D instances within T_{VIO} by $m = \lceil \frac{T_{\text{VIO}}}{T_{\text{ATW}}} \rceil$. We then equally divide T_{VIO} by m to determine the period of SR, T_{SR}, by $T_{\text{SR}} = \frac{T_{\text{VIO}}}{m}$. Although we do not explicitly change the period of ATW, T_{ATW}, it is implicitly determined since ATW runs synchronously to SR under our scheduling approach. To avoid additional contention caused by the delay of some ATW jobs, we further set a lower bound on T_{SR}, with $T_{\text{SR}} \geq t_{\text{IMUi}} + t_{\text{SRR}} + t_{\text{ATW}} + t_{\text{ATWR}}$. This eliminates the possibility of choosing a too small T_{SR} since there will always be t_{ATWR} time between two SRR jobs.

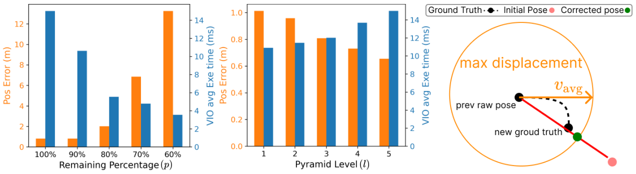
On-demand IMUi. IMUi processes every IMU result to maximize the availability of the fused pose, which leads to wasted work of IMUi since the output fused pose is only used by SR and ATW, which satisfies $T_{\text{SR}} \simeq T_{\text{ATW}} \gg T_{\text{IMU}}$. Although decreasing the IMU sample rate proves effective in



(A) Effect of C1: Increased VIO execution results in C2D increase and render-reprojection contention

(B) Effect of C2: longer rendering time results in more render-reprojection contentions that cause M2D increases

Fig. 8: Impact of C1 and C2 on scheduling



(A) Pose quality and exe. time tradeoff when remaining image(p) changes

(B) Pose quality and exe. time tradeoff when pyramid level(l) changes

(C) Example of VIO Error bound

Fig. 9: Motion-Driven Visual Inertial Odometer

reducing wasted work, it delays the update of the fused pose, resulting in a significant increase in M2D and C2D.

To address this problem, we propose an on-demand IMUi method that samples the IMU reading and computes the fused pose at the beginning of each SR and ATW job. Since the fused pose is only needed before SR and ATW, this method keeps the same M2D and C2D with a timely fused pose update.

We present the complete scheduling example in Fig. 7. Fig. 7(A) shows that given $T_{\text{CAM}} = T_{\text{VIO}} = 60$ and $T_{\text{ATW}} = 16$, we calculate $m = 4$ in the first T_{VIO}. It then depicts the scheduling decisions for each VIO, IMUi, SR, and SRR job based on the contention-preventive scheduling and on-demand IMUi, where the first job of VIO is released at 0 and immediately followed by the sequential execution of IMUi, SR, and SRR. Contention-preventive scheduling further calculates the $T_{\text{SR}} = 15$ and executes IMUi, ATW, and ATWR upon the finish of SRR, as shown in Fig. 7(B). The T_{SR} is then compared with the designed lower bound to prevent contention.

C. Motion-Driven Visual Inertial Odometer (MVIO)

As explained with C1 from Obs. 3, a larger magnitude of motion inflates the VIO execution time due to the increase of new features extracted. Fig. 8(A) illustrates its impact on the schedule. As the VIO execution time increases, the release of the first SR job for the current T_{VIO} is delayed, leading to a significant increase in C2D. Therefore, we control the increase of VIO execution time by limiting the number of features through cropping the input image to VIO as well as decreasing the level of the pyramid for its feature pyramid network [11]. We denote the remaining input image percentage as p and the pyramid level as l. To further analyze the effect of image crop and pyramid level adjustment, we run the VIO task on a PC with V102 trajectory used in Fig. 5 and record the pose quality measured in position error and VIO execution time by only changing p in Fig. 9(A) and by only changing l in Fig. 9(B). We observe that as p decreases, indicating a larger cropped area, the execution time decreases exponentially while the position error increases exponentially. As l increases, indicating a higher pyramid level, the execution time increases linearly as the position error decreases linearly.

Algorithm 1: MVIO Algorithm

```
Input: IMU $a_t, v_t, w_t$, profiling data $v_B, w_B, f(e, l_{\min})$, $e_{\text{req}}, S_{\max}$;
/* Motion Score */;
$t_{prev} \leftarrow$ Timenow();
$S \leftarrow$ CalculateMotionScore($v_t, w_t, v_B, w_B$) /*Eq. 1*/;
$p \leftarrow 1$;
$l \leftarrow l_{\max}$;
if $S > 0$ then
    /* Image Crop and Pyramid Level Adjustments */;
    $p_{\min} \leftarrow$ SetMinimumCrop($f(e_{\text{req}}, l_{\min})$);
    $p \leftarrow$ CalculateP($S, S_{\max}, p_{\min}$) /*Eq. 2*/ ;
    $l \leftarrow$ ChangeL($S, S_{\max}$) /*Eq. 3*/ ;
end
$image \leftarrow$ CropImage(p, $a_t$) /*Crop image based on $a_t$ direction*/;
/* Start of VIO execution */;
$features \leftarrow$ FeatureExtraction($image, l$);
$pose \leftarrow$ PoseEstimation($features$);
/* VIO Error Bounding */;
$t_{\text{VIO}} \leftarrow$ Timenow() - $t_{prev}$;
$raw\_pose \leftarrow$ readRawPose() /*Read latest $raw\_pose$*/;
$max\_dis \leftarrow$ CalculateMaxDisplacement($v_t, a_t, t_{\text{VIO}}, raw\_pose$);
if $pose > max\_dis$ then
    $pose \leftarrow$ LinearFit($raw\_pose, max\_dis$);
end
$raw\_pose \leftarrow pose$ /*Publish updated $raw\_pose$*/;
```

Due to the tradeoff presented in Fig. 9, we design the Motion-Driven Visual Inertial Odometer (MVIO) to control the growth of execution time while limiting the decrease of pose quality. MVIO first sets a budget of VIO execution time $B_{\text{VIO}} = t_{\text{VIO}}$, where t_{VIO} is the VIO execution time used in Sec IV-B. It changes the remaining input image percentage p and pyramid level l and corrects any obvious position errors with its error bounding method. MVIO executes prior to each VIO job, with a complete algorithm provided in Alg. 1.

Motion Score. To capture the motion changes from the default speed and rotation, MVIO first computes a motion score S which quantifies the deviation of the current motion to the representative motion from profiling. During the profiling of VIO execution time, we obtain the scalar value of velocity v_B and scalar value of rotation w_B when VIO completes execution in a given budget B_{VIO} with the original image ($p = 100\%$) and the max pyramid level $l_{\max}$. At runtime, MVIO invokes IMU to get the acceleration and rotation at current time t, and calculates the first integral of a_t to get the current scalar velocity v_t and the current scalar rotation w_t. MVIO then constructs S that describes the motion deviation from the v_B and w_B, which is shown in Eq. 1. During the profiling, we record the maximum observed velocity and rotation and use them to obtain the maximum possible motion score $S_{\max}$, which is given as input to the MVIO algorithm.

$$S = \frac{v_t - v_B}{v_B} + \frac{w_t - w_B}{w_B} \tag{1}$$

The MVIO algorithm given by Alg. 1 calculates S at the beginning (line 4) as it is used in subsequent calculations. Initially, MVIO sets $p = 1$ and $l = l_{\text{max}}$, which means no image cropping or pyramid level reduction from the maximum level (line 5 and line 6). $S > 0$ indicates a larger motion compared to the default v_B and w_B is detected, necessitating image cropping and pyramid level adjustment (line 8 to line 11) to keep the execution time below B_{VIO}. Otherwise, MVIO maintains the default full image and the maximum pyramid level (line 5 to line 6).

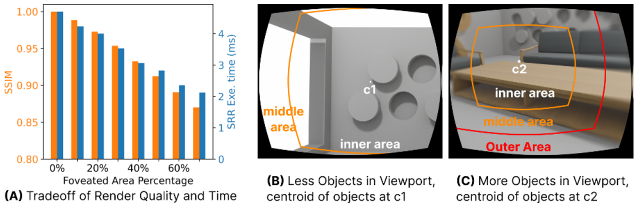
Image Crop Adjustment. To limit the pose quality degradation, MVIO bounds the position error with the minimum remaining percentage p_{min} that is calculated from a profiling function (line 9). Let us denote the position error as e. We construct a function $p = f(e, l_{\text{min}})$ through profiling, which describes the relationship of p to e when applying the minimum pyramid level l_{min}, as shown in Fig. 9(B). By design, we allow the user to determine the maximum position error tolerable in the system, denoted by e_{req}. Therefore, by $f(e_{\text{req}}, l_{\text{min}})$, we get the minimum remaining percentage p_{min}, which is used during image cropping to limit the error to e_{req}. Note that, since the function $f(e, l_{\text{min}})$ has been profiled with l_{min}, the use of $p \geq p_{min}$ ensures at most e_{req} positional error when adjusting l later as long as $l \geq l_{min}$.

As shown by Fig. 9(B), VIO execution time and remaining image percentage p have an exponential relationship. In addition, as illustrated in Fig. 5(A), VIO execution time has an approximately linear relationship to motion speed. Hence, we formulate the calculation of p with an exponential relationship to S, given by Eq. 2.

$$p = \max(p_{\text{min}}, p_{\text{min}}^{\frac{S}{S_{\text{max}}}}) \tag{2}$$

The max function in Eq. 2 ensures if any abnormally large speed and rotation result in an $S > S_{\text{max}}$, MVIO always uses p_{min} to control the position error below e_{req}. The term $\frac{S}{S_{\text{max}}}$ returns the percentage difference between current S to S_{max}. Through the exponential calculation, MVIO returns the p value (line 10) and crops the image based on p and direction of motion indicated by the vector value of acceleration a_t from IMU (line 13). This occurs because new features will always emerge from the direction of motion as new image content appears from that direction.

Pyramid Level Adjustment. As shown in Fig. 9(A), VIO execution time and pyramid level follow a linear decrease relationship that is discrete since the pyramid level is an integer value. Because we profile $p = f(e, l_{\text{min}})$ with a minimum pyramid level l_{min}, the parameter e after image crop should still satisfy $e \leq e_{\text{req}}$, which gives the room for l adjustment. Therefore, we decrease l from l_{max} with a step-wise function in Eq. 3 and compute it after the calculation of p (line 11). During the computation, S_{max} is evenly distributed to $l_{\text{max}} - l_{\text{min}} + 1$ segments. We calculate the current S and then compare it with each segment to determine the corresponding l using the step-wise function. Through the process, MVIO can equally leverage the image crop and pyramid level reduction to control the growth of VIO execution time within the designed B_{VIO} while maintaining the minimum position error e_{req}.



(A) Tradeoff of Render Quality and Time

(B) Less Objects in Viewport, centroid of objects at c1

(C) More Objects in Viewport, centroid of objects at c2

Fig. 10: Scene-Dependent Foveated Rendering

$$l = \begin{cases} l_{\text{max}}, & \text{if } S \leq S_{\text{max}}/l_{\text{max}} \\ l_{\text{max}} - 1, & \text{if } S_{\text{max}}/l_{\text{max}} < S \leq (l_{\text{min}} + 1)S_{\text{max}}/l_{\text{max}} \\ \dots \\ l_{\text{min}}, & \text{if } S > S_{\text{max}}/l_{\text{min}} \end{cases} \tag{3}$$

VIO Error Bounding. Due to the introduction of e_{req}, MVIO further employs a VIO error bounding method to limit the position error by leveraging the raw pose published by the previous VIO. After the execution of the VIO algorithm (line 14 to line 16), MVIO calculates the current VIO execution time t_{VIO} that can be used to determine the maximum possible displacement from the previous raw pose result (line 19). MVIO chooses to use the raw pose because the fused pose introduces further noise from the head motion during the IMUi calculation. The maximum displacement max_dis, indicated as the orange circle in Fig. 9(C), is computed by getting the average velocity v_{avg} during t_{VIO} with $v_{\text{avg}} = v_t + a_t \times t_{\text{VIO}}$ and then apply this speed to the previous raw pose by $max\_dis = raw\_pose + v_{\text{avg}} \times t_{\text{VIO}}$. If the $pose$ from the MVIO execution exceeds the max_dis, the $pose$ is fitted to the raw_pose by finding the intersection between the line that connects $pose$ and raw_pose and the max_dis.

D. Scene-Dependent Foveated Rendering (SFR)

Based on C2 from Obs. 3, we observe the increase of objects in the viewport causes the execution time of SRR (render time) to grow linearly, leading to the contention between SRR and ATWR again shown in Fig. 8(B). Existing work has developed the foveated rendering method that can reduce the rendering time by using low-resolution texture and meshes in the peripheral area [19–24]. Foveated rendering typically involves three levels of rendering resolution: an inner area with the original resolution, a middle area with half the original resolution, and an outer area with one-quarter of the original resolution. We denote the inner area as the foveation area. While existing work uses a fixed foveation area that is not adaptive to scene changes, we change the foveation area and plot the tradeoff of frame quality measured in the Structural Similarity (SSIM) Index and render time in Fig. 10(A) with a fixed objects number. We choose to use the SSIM index because it measures the human-perceived difference in image, which is widely used in existing research [10, 25, 26]. Fig. 10(A) indicates both

Algorithm 2: SFR Algorithm

```
1  Input: SR $vp\_objects$, profiling data $K, M, C, r_{max}, n_B$;
2  $n$ ←Lengthof($vp\_objects$) /*get number of objects in viewport*/;
3  $\alpha \leftarrow 1$;
4  /* Foveation Scaling Factor */;
5  if $n > n_B$ then
6      $\gamma \leftarrow$ CalculateGamma($\alpha, K, M, C, n_B$) /*Eq. 4*/;
7      /* Optimization Loop */;
8      while $C\gamma > B_{SRR}$ do
9          if $\alpha > 0$ then
10             $\alpha \leftarrow \alpha - 0.1$;
11             $\gamma \leftarrow$ CalculateGamma($\alpha, K, M, C, n_B$) /*Eq. 4*/;
12         end
13     end
14 end
15 /* Foveated Rendering */;
16 $inner\_area \leftarrow \gamma \cdot (height \times width)$;
17 $middle\_area \leftarrow$
       $(\gamma \cdot height + offset) \times (\gamma \cdot width + offset) - inner\_area$;
18 $outer\_area \leftarrow outer\_area - middle\_area - inner\_area$;
19 $c$ ←CalculateCentroid($vp\_objects$) /*get centroid of objects*/;
20 $2D\_frame \leftarrow$
       FoveatedRender($c, inner\_area, middle\_area, outer\_area$);
21 $frame \leftarrow 2D\_frame$ /*publish rendered 2D frame*/;
```

the quality of rendering and rendering time follow a linear decrease relationship to the area of foveation.

Motivated by this tradeoff, we design the Scene-Dependent Foveated Rendering (SFR) that aims to control the increase of rendering time with minimum degradation of rendering quality. SFR first adopts a budget of render time $B_{\text{SRR}} = t_{\text{SRR}}$. We then profile the number of objects that result in B_{SRR} render time with no usage of foveated rendering and denote it as n_B. SFR replaces the default SRR and follows Alg. 2.

Foveation Scaling Factor. Due to the linearity of render quality and render time to the foveation area in Fig. 10(A), we set up a foveation scaling factor, denoted as γ, to scale the area of foveation which limits the growth of render time within the designed B_{SRR}. Through offline profiling, we are able to get the linear coefficients of each linear relationship. In the following equations, C denotes the linear coefficient of render time to γ, M denotes the linear coefficient of rendering quality to γ, and K denotes the linear coefficient of render time to number of objects in the viewport. To start up, the objects in the current frame's viewport, denoted as $vp\_objects$, are provided alongside C, K, and M. SFR first gets the number of $vp\_objects$ (line 2) n and calculates the γ according to Eq. 4 when $n > n_B$ (line 6). In the equation, α is used as a control value that is initially set to 1 to max out the frame quality (line 3). The first term indicates that when $\gamma = 1$, no foveated rendering is used. The next term $\frac{(1-\alpha)M}{\alpha C}$ indicates the contributing factor from render quality. The last term $\frac{(n-n_B)}{\alpha KC}$ indicates the contributing factor from object number changes.

$$\gamma = 1 - \frac{(1-\alpha)M}{\alpha C} - \frac{(n-n_B)}{\alpha KC} \tag{4}$$

Optimization Loop. With the initial γ changes, it is still possible that the rendering time exceeds the B_{SRR}, so we opt to further reduce the rendering time by making concessions in rendering quality with a decreased α in Eq. 4. This process is called the Optimization loop and is presented in line 7 to line 11. Initially, SFR projects the rendering time with the calculation of $C\gamma$ and compares it with the B_{SRR}. If the projected rendering time exceeds B_{SRR}, SFR lowers the value of α by 0.1 and recalculates γ using Eq. 4. This loop ends when $\alpha = 0$ or $C\gamma \leq B_{\text{SRR}}$.

Dynamic Objects Centroid. Since the centroid of objects does not align with the viewport center, we need to calculate the objects' centroid and fix the center of foveation there to effectively reduce the rendering time. Therefore, after setting the inner area, middle area, and outer area (line 16 to line 18), SFR calculates the centroid of objects in the viewport by taking the average of all objects geometry center x coordinates x_{avg} and y coordinates y_{avg} and returns the centroid coordinates as $(x_{\text{avg}}, y_{\text{avg}})$. We show two examples of 2D frames rendered by SFR in Fig. 10(B).

E. Generalizability

Since BOXR only modifies the scheduler and three tasks (VIO, IMUi, and SRR) of the state-of-the-art XR framework, it can be directly applied to any XR systems that share similar tasks and pipelines. When deploying on a new platform, BOXR requires re-profiling task execution times for a target hardware setup. This step provides the execution times needed for the BOXR scheduling policy and sets the budget for MVIO and SFR. To eliminate the manual effort involved in the process, BOXR includes a setup application that collects all necessary execution times for each task. These data are then fed into our scheduler, MVIO, and SFR algorithm to determine the initial conditions for execution. Following the same process, we have successfully deployed BOXR onto three different hardware platforms in Sec. V-A.

One may have a concern that since the execution times of all tasks have to be re-profiled for each hardware setup, it may generate profiling errors that can interfere with the scheduling decisions and lead to performance degradation. However, since the MVIO and SFR use the same profiling data to make runtime adaption, they can offset the inaccuracy by reactively changing the magnitude of adjustment. For example, if a shorter VIO execution time is profiled, MVIO will crop the image more aggressively due to a larger difference between B_{VIO} and t_{VIO}. Hence, VIO is still controlled below B_{VIO} amidst motion dynamics.

V. Evaluation

A. Evaluation Setup

Hardware Setup. To encompass a broad range of hardware platforms commonly used to support XR systems, we evaluate BOXR on three hardware platforms, including a PC equipped with NVIDIA GTX 3060 GPU and two embedded devices, NVIDIA AGX Xavier [27] and NVIDIA Orion Nano [28], which are widely used in cutting-edge XR research [10, 29–31]. We fix the power of the PC to 180W, Xavier to 30W, and Nano to 15W to control the GPU frequency and lock the CPU frequency of each embedded platform to its maximum value. The output frame is displayed on Northstar Next HMD [32] with a max 90Hz refresh rate.

TABLE I: Evaluated XR Applications

	Sponza(Spon)	**Materials**(Mat)	**Gldemo**(Gl)	**Platformer**(Plat)
Object	32	81	7	3014
Vertex	192870	62826	54760	26168
Texture	33	24	8	4

TABLE II: EuRoc MAV Dataset Categorization

	No Motion	**Small Motion**	**Med. Motion**	**Large Motion**
$v_{3D}(m/s)$	$[0, 0.1)$	$[0.1, 1)$	$[1, 2)$	$[2, \infty)$
Perc(%)	10.62	66.75	20.30	2.33

XR Applications Setup. To encompass diverse scene complexities that contribute to different rendering pressures in the everyday use case of XR, we selected four OpenXR [1] applications initially built with the Godot engine [17]. Their details are shown in Table I and ordered by decreasing rendering demands. Sponza brings the user to a detailed palace for exploration. Materials renders a series of identical objects with various textures. Gldemo simulates an indoor scene resembling a household living room. Platformer creates a pixel-style world with numerous individual objects for gaming scenarios. We fixed the output resolution to 2560×1440 and set both horizontal and vertical fields of view to $90°$. To avoid additional overhead from application interfaces and focus on the effects of motion and scene dynamics, we loaded only the scene information, including meshes and textures, from these applications and used OpenGL as the rendering engine for all rendering and reprojection tasks.

Controlled Trajectories. We controlled the input visual-inertial information using the eleven trajectories from EuRoc MAV datasets [18]. This dataset covers the majority of indoor XR use cases with average speed from $0.33m/s$ to $0.99m/s$ and rotation from $0.21rad/s$ to $0.66rad/s$. We set the CAM period to 50ms and the IMU period to 5ms which is used by the dataset. To understand the effect of head and body motion on the system performance, we categorized all the visual-inertial information into four motion classes shown in Table II based on their ground-truth 3D speed (v_{3D}), which is the scalar value of the linear speed in 3D space composed of both velocity and rotation.

Baseline Configuration. We compared BOXR against two state-of-the-art baselines derived from the ILLIXR framework [29–31, 33]. ILLIXR [10] is a popular open-sourced XR software testbed that is being widely used in XR research. *ILLIXR* uses the default publisher-subscriber model in Sec. II-A to schedule tasks, with V-Sync disabled to maximize throughput. *ILLIXR-OP* uses the same optimized version of ILLIXR in Sec. III-A, which modifies the periods of SR and ATW to minimize M2D and C2D while still adhering to the same publisher-subscriber model. To assess the contribution of each component, we evaluate the scheduling policy and the complete framework separately by making them into two separate baselines. *BOXR-S* is a static version that only implements the scheduling policy described in Sec. IV-B. *BOXR* implements the complete framework that includes the scheduling policy, MVIO, and SFR in Sec. IV. We fixed the target frame rate to 90 FPS, which matches the max screen refresh rate, and recorded the results of M2D and C2D at the end of each frame.

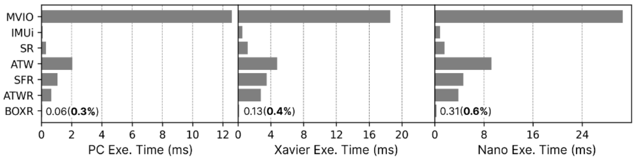


Fig. 11: System runtime breakdown

TABLE III: Dropped IMU sensor information

	MH01	**MH05**	**V101**	**V102**
ILLIXR	36%	26%	27%	35%
ILLIXR-OP	62%	46%	26%	73%
BOXR-S	0%	0%	0%	0%
BOXR	0%	0%	0%	0%

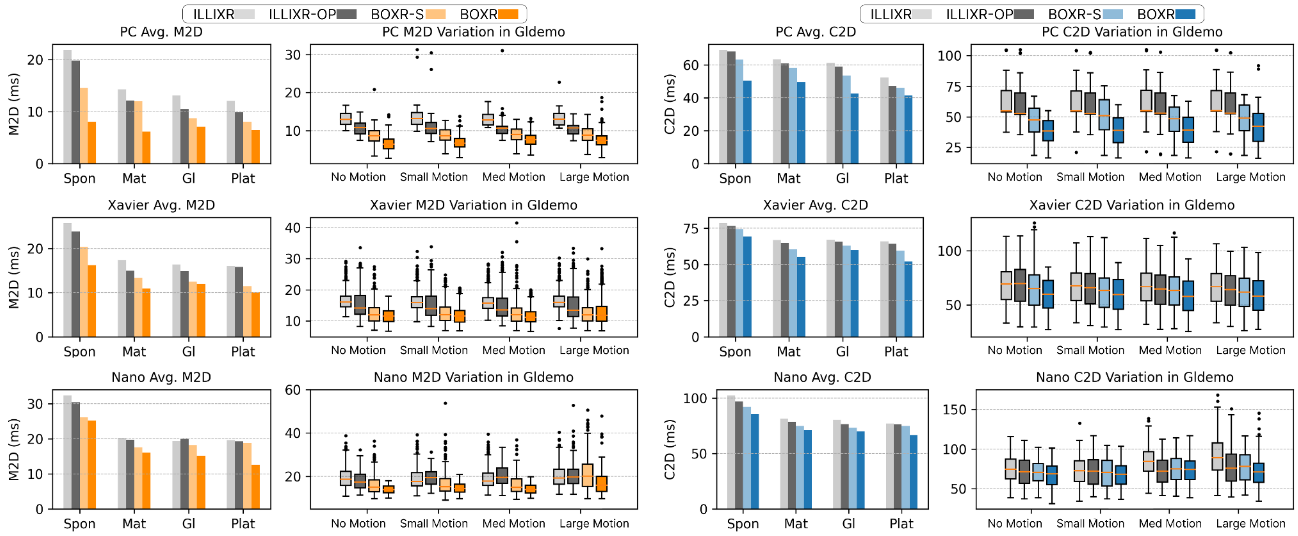
B. Controlled Settings Effectiveness

Latency and Overhead Breakdown. To evaluate the overhead of BOXR, we profiled the average execution time of each task in the XR framework. We use the MVIO to replace the VIO task and SFR to replace the SRR task. Within these tasks, we separately profile the overhead of running BOXR algorithms, including the scheduling policy, line 3 to line 13 and line 17 to line 22 in Alg. 1, as well as line 2 to line 19 in Alg. 2. As shown in Fig. 11, the average overhead of our method is around 0.06ms, 0.13ms, and 0.31ms compared to the entire average system runtime, which is more than 17ms on PC, 36ms on Xavier, and 48ms on Nano. The overhead of using our method is about 0.3%, 0.4%, and 0.6%, respectively.

Dropped IMU Sensor Information. We record the dropped IMU sensor information in two trajectories from the Machine Hall (MH01, MH05) and two from the Vicon Room (V101, V102) in EuRoC MAV. As shown in Table III, in ILLIXR, the IMU operates at a significantly shorter period than SR and ATW, leading to up to 36% dropped information. ILLIXR-OP further increases the SR period, causing up to 73% IMU sensor information to be dropped. BOXR-S, benefiting from On-demand IMUi, triggers IMU sensor readings only at the beginning of SR and ATW, resulting in no dropping of sensor information. Additionally, BOXR samples IMU readings at the beginning of VIO, which are used to calculate the raw pose leading to no drop of sensor information.

M2D Metric. To obtain the M2D value defined in Def. 1, we record the age of the latest IMU information at the time of frame output. The resulting average M2Ds are shown in the left subfigures of Fig. 12(A). Compared to ILLIXR, BOXR-S reduces M2D by up to 33% because the proposed scheduling policy effectively reduces contention between SRR and ATWR. Furthermore, BOXR achieves M2D reductions by up to 63% on PC, 37% on Xavier, and 36% on Nano, thanks to SFR which further controls the rendering time when rendering a large number of objects.

The right subfigures of Fig. 12(A) demonstrate the variation of M2D. BOXR-S increases M2D standard deviation by 15%



(A) Average M2D to XR applications and M2D variations to Motion Class

(B) Average M2D to XR applications and M2D variations to Motion Class

Fig. 12: M2D and C2D Evaluation. The plots from left to right show the average M2D, M2D variations, average C2D, and C2D variations. The M2D and C2D variations show each M2D and C2D given different motion classes in the Gldemo application.

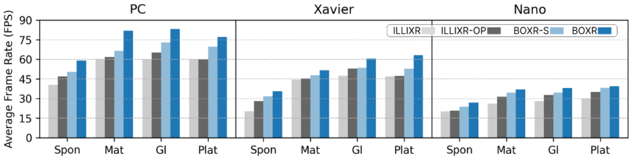


Fig. 13: Avg. output frame rate with 90FPS target frame rate

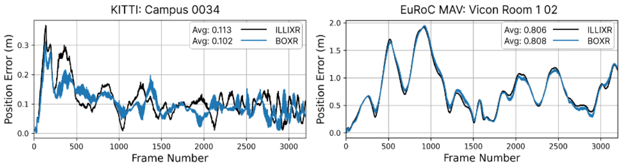


Fig. 14: Position error and render time-quality tradeoff

compared to ILLIXR on Nano. As shown in Fig. 8(B), BOXR-S assumes constant task execution time, which cannot eliminate contention due to the dynamics brought by objects in the scene (C2) and potentially increases the M2D variation during runtime. However, BOXR reduces M2D standard deviation by up to 57% compared to ILLIXR. This is due to the use of SFR, which bounds rendering time to avoid contention while maintaining similar quality by dynamically adjusting the foveation area and object centroid based on the number of objects in the scene.

C2D Metric. We add the CAM timestamp to each SRR-generated 2D frame and then record the age of this CAM information during the final 3D frame output as C2D. The left subfigures of Fig. 12(B) illustrate the average C2D. BOXR-S achieves up to 13% C2D reduction compared to ILLIXR since the scheduling policy effectively prevents contentions between SRR and ATWR. BOXR reduces C2D by up to 27% compared to ILLIXR, owing to MVIO which bounds the VIO execution time across from motion dynamics.

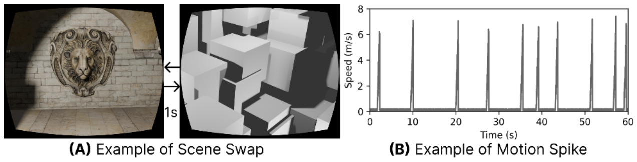
The right subfigures of Fig. 12(B) show the variation of C2D. BOXR-S experiences 6% increase of C2D standard deviation compared to ILLIXR on Xavier. As explained in Fig. 8(A), the dynamics of motion (C1) contribute to variation increase since BOXR-S neglects the motion effect on VIO execution time. However, BOXR achieves up to 23% less C2D standard deviation compared to ILLIXR. This significant improvement is made possible by MVIO, which controls the VIO execution time with image crop and pyramid level adjustment during large motion. Notably, the only extremely large C2D outliers for BOXR occur in the large motion class, which constitutes just 2% of the entire dataset. This is because when unexpectedly large motion dynamics happen (i.e. sudden drop or turn), MVIO may still use the previous velocity to calculate the first VIO job during such abrupt motion dynamics, creating a single large C2D outlier. We will provide more evaluation about this in Sec. V-C.

Average Output Frame Rate. Figure 13 shows the average output frame rate measured in frames per second (FPS) during a 60-second runtime, with the target frame rate fixed at 90 FPS. Due to reduced contention between SRR and ATWR through BOXR scheduling policy, BOXR-S achieves up to a 27% frame rate increase compared to ILLIXR and up to a 16% increase compared to ILLIXR-OP. BOXR achieves up to a 43% frame rate increase compared to ILLIXR and up to a 35% increase compared to ILLIXR-OP, thanks to the use of SFR that adapts to the objects change. With BOXR optimization, the PC achieves a consistent 60 FPS across all four applications, ensuring an ideal XR experience. Xavier achieves 60 FPS in Gldemo and Platformer, making it practical for low-render demand XR applications. Furthermore, BOXR enables usage of the XR system on Nano, which outputs 30 FPS for three applications.

Quality of Pose. To test the general effectiveness of the MVIO algorithm, we run both ILLIXR and BOXR with the

TABLE IV: Quality of Rendering for Different Platforms

	PC		Xavier		Nano	
	ΔTime	ΔSSIM	ΔTime	ΔSSIM	ΔTime	ΔSSIM
Spon	-15.0%	-3.7%	-25.4%	-5.6%	-30.4%	-5.7%
Mat	-16.0%	-2.7%	-21.6%	-4.6%	-26.1%	-4.8%
GL	-6.5%	-1.2%	-16.9%	-1.8%	-22.3%	-3.4%
Plat	-12.9%	-1.9%	-10.5%	-1.3%	-20.8%	-3.0%



(A) Example of Scene Swap (B) Example of Motion Spike

Fig. 15: Scene swap and motion spike

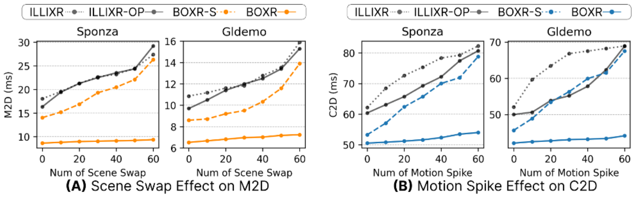
Gldemo application on PC and record the position error from ground truth in Fig. 14 for a one-minute trial. To differentiate the environments for VIO and evaluate the general applicability, we conduct the experiment with the KITTI [34] camera feed for outdoor large motion scenario as well as the EuRoC MAV [18] camera feed for indoor moderate motion scenario. BOXR decreases the position error by 9.7% for the outdoor scenario of KITTI but yields a similar error for the indoor scenario of EuRoC MAV. Since KITTI involves significantly larger velocities, ILLIXR shows a larger variability of position errors due to the prolonged VIO execution time, resulting in unfinished VIO jobs. Benefiting from MVIO, BOXR mitigates the problem by controlling VIO execution time to avoid VIO jobs being dropped and achieve more accurate localization. Another observation is that BOXR accumulates position error more rapidly than ILLIXR when consecutively large motion dynamics happen, as shown in 1000 to 1500 frames. However, it quickly corrects the subsequent poses due to the use of VIO error bounding and reaches the same pose quality when motion magnitude decreases. This experiment shows that BOXR can maintain similar pose quality during indoor moderate motion scenarios, while even improving the pose quality in outdoor large motion scenarios.

Rendering Output Quality. Table IV records the change of render time and quality of output frame measured in the Structural Similarity (SSIM) Index from ILLIXR to BOXR. SSIM captures the perceived quality of the output frame, which is a direct implication of user notice of visual change [25]. Although we see a decrease of SSIM up to 5.7% in Nano, we trade off this rendering quality degradation to a 30.4% decrease in rendering time, which contributes to better M2D and C2D while increasing output frame rate.

Overall Effectiveness: BOXR enhances performance across all platforms by addressing the challenges presented in Obs. 1 and Obs. 2 through its scheduling policy design and tackling the challenge from Obs. 3 with runtime adaptation.

C. Burst-dynamics Scenarios Performance

Burst-dynamics Generation. Abrupt changes in scenes and motions lead to dynamics bursts, which include *scene swaps* when the user changes applications and *motion spikes* when the user experiences sudden acceleration or stops. To test the system's effectiveness in these scenarios, we generate two burst-dynamic scenarios that exhibit similar effects on the system. For scene swaps, we switch from the current scene to a new scene that contains over 2000 objects as shown in Fig. 15(A). We force rendering 10 frames of the new scene to mimic a surge in render demand, which is prevalent in abrupt scene-changing scenarios. For motion spikes, we connect the Zed Mini Stereo camera [35] and drop the camera from a fixed height, leading to a sudden acceleration depicted in Fig. 15(B). We increase the number of these burst dynamics during a 60-second runtime by injecting more occurrences randomly. We evaluate across all four baselines using two applications, Sponza and Gldemo on PC.



(A) Scene Swap Effect on M2D (B) Motion Spike Effect on C2D

Fig. 16: Scene swap effect and motion spike effect

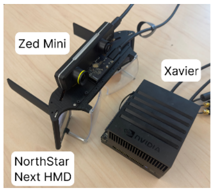
Scene Swap Effect on M2D. Scene swaps cause a significant render time increase, which substantially impacts the M2D metric illustrated in Fig. 8(A). We record the average M2D under an increased number of scene swaps during 60-second trials. As shown in Fig. 16(A), the other three baselines experience up to a 57% increase in M2D. In contrast, BOXR effectively limits the increase of M2D to at most 10% when scene swaps occur every second. Compared to other baselines, BOXR keeps the render time below the designed budget through SFR, which effectively mitigates the rise in M2D even under abrupt scene swaps.

Motion Spike Effect on C2D. Motion spikes significantly increase the execution time of VIO, leading to a substantial rise in C2D depicted in Fig. 8(B). We, therefore, record the average C2D as the number of motion spikes gradually increases in Fig. 16(B). The baselines show a significant increase in C2D, up to 48%, whereas BOXR only experiences a 6% increase in C2D, owing to the use of MVIO, which effectively controls the execution time increase even during motion spikes.

Robustness in extreme cases: Stemming from Obs. 3, BOXR effectively limits the increase of M2D and C2D with its runtime adaptations in highly dynamic scenarios, proving its robustness within extreme use cases.

D. Real-world Experiment

Experiment Setting. To closely match the state-of-the-art XR systems applications while maintaining freedom of movement, we connect the Zed Mini Camera [35] to the Xavier platform and design three XR applications using the Godot game engine [17] to cover the different magnitudes of user motion, shown in Fig. 17. Video Watch creates an



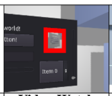
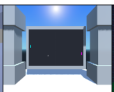
			
App. Name	**Video Watch**	**Room Explore**	**Escape Game**
$v_{3D}(m/s)$	0.04 ± 0.02	0.41 ± 0.13	1.03 ± 0.72
Space(m^2)	2×1	8×15	5×6
Resolution	2560×1440	2560×1440	2560×1440
Target FPS	90	90	90

Fig. 17: Real-world Testcases setup

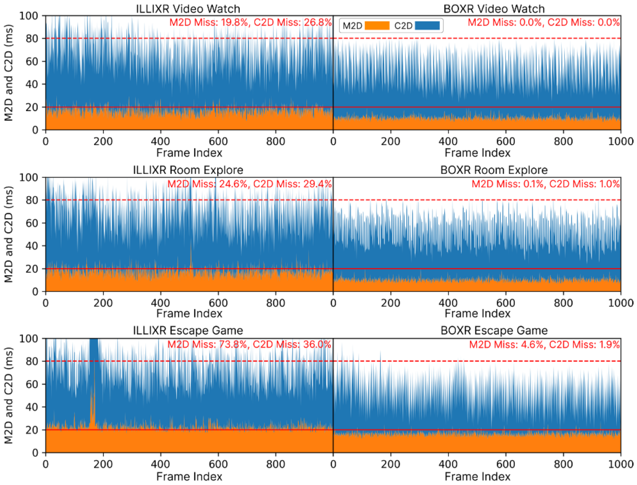


Fig. 18: M2D and C2D in Real-world Experiments

interface that can play any static images or videos. Room Explore invites the users to roam freely in a 3D demo scene. Escape Game utilizes scenes from Platformer and makes the sprites chase and fire at the user, who needs to dodge and avoid being caught. We test the three applications on the default ILLIXR and the complete BOXR frameworks. We also adopt the 20ms M2D requirement and 80ms C2D requirement described in Sec. II-B, which indicate an unnoticeable head motion and body motion delay, respectively.

M2D and C2D. We sample M2D and C2D from 1000 frames during a 60-second trial and present the results in Fig. 18. Across all applications tested, BOXR achieves up to a 42.6% reduction in M2D and a 31.8% reduction in C2D compared to ILLIXR. While no M2D and C2D in BOXR exceed their respective requirements in Video Watch, which involves low-magnitude motion, 4.6% of M2D and 1.9% of C2D values miss the requirements in Escape Game due to greater magnitude of motion. In extremely large motion around frame 200, ILLIXR experiences over 60ms of and 100ms of C2D, whereas BOXR's M2D stays below 20ms and C2D is maxed out at 90ms.

Applicability to real-world scenarios: The consistently low M2D and C2D in real-world test cases prove the usability of BOXR in practical deployment.

VI. Related Work

Latency Metrics for XR. Existing works have proposed multiple latency metrics in the context of distributed asynchronous systems similar to the pub-sub model used by XR systems [13, 36]. Among these metrics, the most widely accepted metric for XR systems is M2D because of its direct association to motion-sickness [3, 37]. Existing work addresses this issue from various perspectives. Some employ reprojection methods for timely pose updates to reduce M2D [7–9]. For example, [8] enables ATW on a commercial XR headset, while [7] uses machine learning to predict and delay ATW execution. However, these methods assume fully preemptive GPU execution with no dependencies, which does not hold true in practical GPU systems [13, 14]. Dedicated hardware for tasks like ATW [38, 39] increases power and complexity, challenging integration into existing XR frameworks. Offloading computation to nearby edge servers [26, 40–43], such as CollabVR [26], reduces M2D but requires high bandwidth and powerful PCs nearby. Despite these efforts, motion sickness persists due to neglect of C2D.

XR Scene Rendering. The increased material resolution makes rendering every frame in raw resolution impractical [24]. Foveated rendering methods degrade peripheral viewport quality while maintaining central resolution [19–23, 44–46]. RITnet [44] uses real-time semantic segmentation for eye-tracking and foveated rendering based on user gaze but lacks adaptive foveation based on scene dynamics. Reprojection-based methods run asynchronously to compensate for frame loss [8, 47–49], such as [49] minimizing frame warping cost to improve frame rates. However, they often use previous frame data, leading to large C2D.

VII. Conclusion

This paper presents BOXR: a Body and head motion Optimization framework for eXtended Reality. Building upon the three critical observations detailed in Sec. III, BOXR employs a scheduling policy alongside two dynamic runtime adaptations in Sec. IV-C and Sec. IV-D. Through comprehensive comparison with ILLIXR, BOXR demonstrates significant performance improvement in controlled and real-world scenarios, proving the design's general effectiveness, robustness in extreme use cases, and adaptability to practical deployment.

Despite all the benefits achieved by BOXR, there are still interesting directions for future work. First, while BOXR focuses solely on software-level optimization, integrating software-hardware co-optimization could further enhance performance. Second, the benchmark we used primarily reflects data-intensive daily working scenarios but not GPU-intensive gaming scenarios. Addressing these will enable the development of a more comprehensive XR system.

Acknowledgement

This work was supported by the National Science Foundation under Grants CPS 2230969, CNS 2300525, CNS 2343653, CNS 2312397, CNS 1943265.

References

[1] KhronosGroup, "OpenXR Software Development kit (SDK) Sources Project," 2024, https://github.com/KhronosGroup/OpenXR-SDK-Source.

[2] Apple, "Apple Vision Pro," 2024, https://www.apple.com/apple-vision-pro/.

[3] D. Wagner, "Motion to photon latency in mobile ar and vr," https://medium.com/@DAQRI/motion-to-photon-latency-in-mobile-ar-and-vr-99f82c480926.

[4] Centers for Disease Control and Prevention, "Motion Sickness," 2024, https://wwwnc.cdc.gov/travel/page/motion-sickness.

[5] E. Cosner, "Snowboarding with apple vision pro - can this actually work?" 2024, https://youtu.be/LTVuGsYY2yc.

[6] Optofidelity, "Apple Vision Pro benchmark test 2: Angular Motion-to-Photon Latency in VR," 2024, https://www.optofidelity.com/insights/blogs/apple-vision-pro-bencmark-test-2.-angular-motion-to-photon-latency-in-vr.

[7] A. Dixit and S. R. Sarangi, "Minimizing the motion-to-photon-delay (mpd) in virtual reality systems," *arXiv preprint arXiv:2301.10408*, 2023.

[8] J. M. P. van Waveren, "The asynchronous time warp for virtual reality on consumer hardware," in *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, ser. VRST '16, New York, NY, USA, 2016.

[9] X. Hou, J. Zhang, M. Budagavi, and S. Dey, "Head and body motion prediction to enable mobile vr experiences with low latency," in *IEEE Global Communications Conference (GLOBECOM)*, 2019.

[10] M. Huzaifa, R. Desai, S. Grayson, X. Jiang, Y. Jing, J. Lee, F. Lu, Y. Pang, J. Ravichandran, F. Sinclair, B. Tian, H. Yuan, J. Zhang, and S. V. Adve, "Illixr: Enabling end-to-end extended reality research," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, 2021.

[11] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "Openvins: A research platform for visual-inertial estimation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[12] F. Dellaert and G. Contributors, "borglab/gtsam," May 2022, https://github.com/borglab/gtsam.

[13] S. Kato, K. Lakshmanan, R. Rajkumar, Y. Ishikawa *et al.*, "Timegraph: Gpu scheduling for real-time multi-tasking environments," in *2011 USENIX Annual Technical Conference (USENIX ATC 11)*, 2011.

[14] Y. Wang, M. Karimi, Y. Xiang, and H. Kim, "Balancing energy efficiency and real-time performance in GPU scheduling," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, 2021.

[15] Optofidelity, "Apple Vision Pro Benchmark Test 1: See-Through Latency, Photon-to-Photon," 2024, https://www.optofidelity.com/insights/blogs/apple-vision-pro-benchmark-test-1-see-through-latency-photon-to-photon

[16] CenturyLink, "How to improve latency (ping)," https://www.centurylink.com/home/help/internet/how-to-improve-gaming-latency.html.

[17] Godot, "Godot engine," Aug. 2017, https://godotengine.org/.

[18] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016.

[19] M. Weier, T. Roth, E. Kruijff, A. Hinkenjann, A. Pérard-Gayot, P. Slusallek, and Y. Li, "Foveated real-time ray tracing for head-mounted displays," in *Computer Graphics Forum*, vol. 35, no. 7. Wiley Online Library, 2016.

[20] M. Fujita and T. Harada, "Foveated real-time ray tracing for virtual reality headset," *Light Transport Entertainment Research*, vol. 3, 2014.

[21] M. Weier, T. Roth, A. Hinkenjann, and P. Slusallek, "Foveated depth-of-field filtering in head-mounted displays," *ACM Transactions on Applied Perception (TAP)*, 2018.

[22] J. Kim, Y. Jeong, M. Stengel, K. Aksit, R. A. Albert, B. Boudaoud, T. Greer, J. Kim, W. Lopes, Z. Majercik *et al.*, "Foveated ar: dynamically-foveated augmented reality display." *ACM Trans. Graph.*, vol. 38, no. 4, 2019.

[23] S. Lee, M. Wang, G. Li, L. Lu, Y. Sulai, C. Jang, and B. Silverstein, "Foveated near-eye display for mixed reality using liquid crystal photonics," *Scientific Reports*, 2020.

[24] S. Jabbireddy, X. Sun, X. Meng, and A. Varshney, "Foveated rendering: Motivation, taxonomy, and research directions," *arXiv preprint arXiv:2205.04529*, 2022.

[25] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, 2004.

[26] Z. Ke, X. Zhou, D. Jiang, H. Yan, and T. Qiu, "Collabvr: Reprojection-based edge-client collaborative rendering for real-time high-quality mobile virtual reality," in *IEEE Real-Time Systems Symposium (RTSS)*, 2023.

[27] Nvidia, "Jetson AGX Xavier," 2018.

[28] ——, "Jetson Orin Nano," 2023.

[29] Q. Jiang, M. Huzaifa, W. Sentosa, J. Zhang, S. Gao, Y. Pang, B. Godfrey, and S. Adve, "Offloading visual-inertial odometry for low power extended reality," in *2023 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, 2023.

[30] R. Singh, M. Huzaifa, J. Liu, A. Patney, H. Sharif, Y. Zhao, and S. Adve, "Power, performance, and image quality tradeoffs in foveated rendering," in *2023 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, 2023.

[31] H. You, C. Wan, Y. Zhao, Z. Yu, Y. Fu, J. Yuan, S. Wu, S. Zhang, Y. Zhang, C. Li *et al.*, "Eyecod: eye tracking system acceleration via flatcam-based algorithm & accelerator co-design," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022.

[32] NorthStar, "Northstar Next," 2023.

[33] K. Liu, N. Wu, and B. Han, "Demystifying web-based mobile extended reality accelerated by webassembly," in *Proceedings of the 2023 ACM on Internet Measurement Conference*, 2023.

[34] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[35] StereoLabs, "ZED Mini Stereo Camera," 2021.

[36] M. Günzel, K.-H. Chen, N. Ueter, G. von der Brüggen, M. Dürr, and J.-J. Chen, "Timing analysis of asynchronized distributed cause-effect chains," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021.

[37] R. K. Kundu, A. Rahman, and S. Paul, "A study on sensor system latency in vr motion sickness," *Journal of Sensor and Actuator Networks*, 2021.

[38] F. Smit, R. van Liere, S. Beck, and B. Fröhlich, "An image-warping architecture for vr: Low latency versus image quality," in *IEEE Virtual Reality Conference*, 2009.

[39] C. Xie, X. Zhang, A. Li, X. Fu, and S. Song, "Pim-vr: Erasing motion anomalies in highly-interactive virtual reality world with customized memory cube," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[40] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, and N. Dai, "Furion: Engineering high-quality immersive virtual reality on today's mobile devices," in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, 2017.

[41] J. Meng, S. Paul, and Y. C. Hu, "Coterie: Exploiting frame similarity to enable high-quality multiplayer vr on commodity mobile devices," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020.

[42] X. Liu, C. Vlachou, M. Yang, F. Qian, L. Zhou, C. Wang, L. Zhu, K.-H. Kim, G. Parmer, Q. Chen *et al.*, "Firefly: Untethered multi-user {VR} for commodity mobile devices," in *USENIX Annual Technical Conference*, 2020.

[43] C. Xie, X. Li, Y. Hu, H. Peng, M. Taylor, and S. L. Song, "Q-vr: system-level design for future mobile collaborative virtual reality," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021.

[44] A. K. Chaudhary, R. Kothari, M. Acharya, S. Dangi, N. Nair, R. Bailey, C. Kanan, G. Diaz, and J. B. Pelz, "Ritnet: Real-time semantic segmentation of the eye for gaze tracking," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019.

[45] E. Turner, H. Jiang, D. Saint-Macary, and B. Bastani, "Phase-aligned foveated rendering for virtual reality headsets," in *2018 IEEE conference on virtual reality and 3D user interfaces (VR)*. IEEE, 2018.

[46] Z. Zheng, Z. Yang, Y. Zhan, Y. Li, and W. Yu, "Perceptual model optimized efficient foveated rendering," in *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*, 2018.

[47] R. Kijima and T. Ojika, "Reflex hmd to compensate lag and correction of derivative deformation," in *Proceedings IEEE Virtual Reality 2002*, 2002.

[48] W. R. Mark, G. Bishop, and L. McMillan, "Post-rendering image warping for latency compensation unc-ch computer science technical report# 96-020," *Science*, 1996.

[49] E. Peek, C. Lutteroth, and B. Wünsche, "More for less: Fast image warping for improving the appearance of head tracking on hmds," in *2013 28th International Conference on Image and Vision Computing New Zealand (IVCNZ 2013)*, 2013.