Q-VR: System-Level Design for Future Mobile Collaborative Virtual Reality

CHENHAO XIE, XIE LI, YANG HU, HUWAN PENG, MICHAEL TAYLOR, SHUAIWEN LEON SONG

Presentation by-Ayush Garg

Presentation Overview

- Introduction
- Current VR Systems and Their Limitations
- Q-VR
  - Software-level flexibility
  - Hardware support
- Evaluation
- Summary

Introduction

Min performance requirements

- end-to-end latency (i.e., Motion to-Photon latency or MTP) < 25 ms
- frame rate > 90 Hz ~ 11ms

Mobile VR designs cannot satisfy the realtime performance requirements due to:

- highly interactive nature of user's actions
- complex environmental constraints during VR execution

High quality VR applications are designed on a tethered setup which significantly limits users' mobility

Authors propose Q-VR, a novel dynamic collaborative rendering solution via software-hardware co-design for enabling future low-latency high-quality mobile VR.

- Introduction
- Current VR Systems and Their Limitations
- Q-VR
  - Software-level flexibility
  - Hardware support
- Evaluation
- Summary

Modern VR System Components

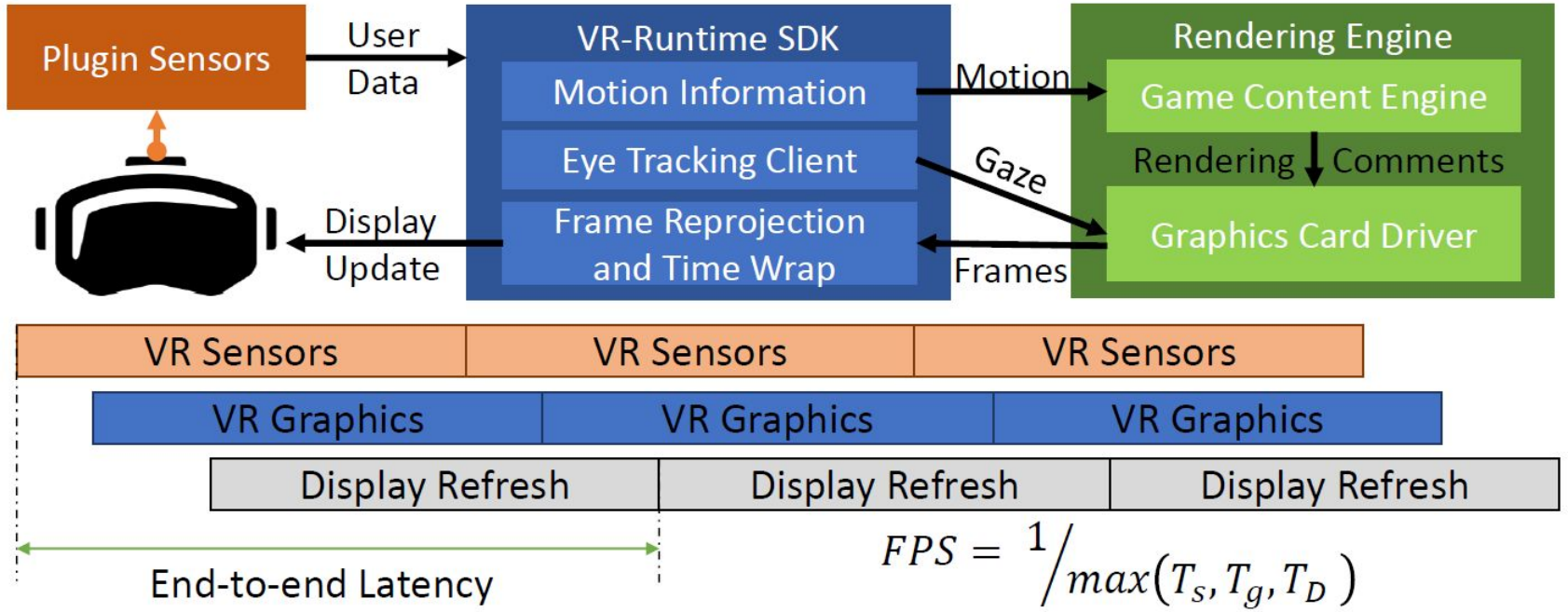
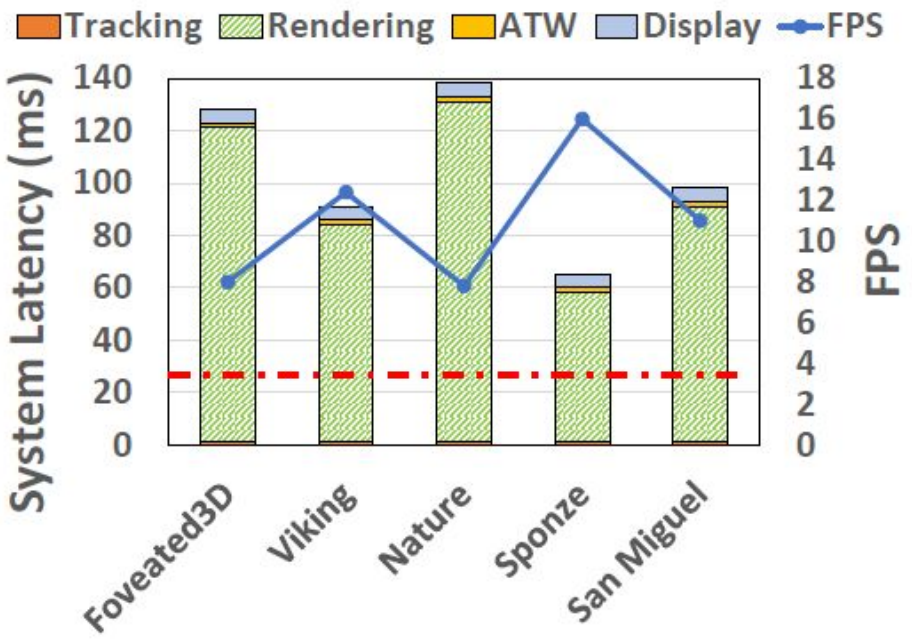
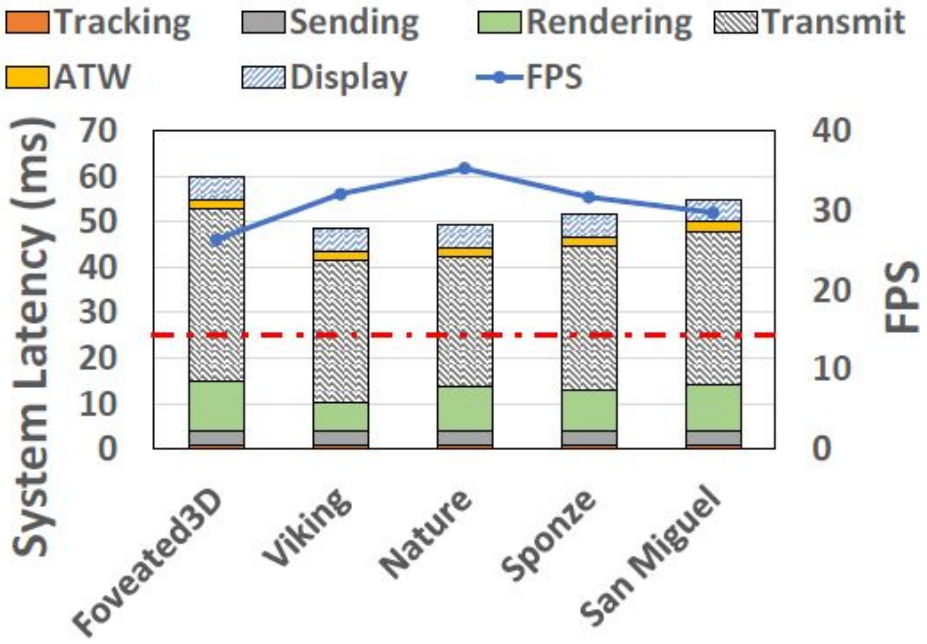


Figure 2: An example of a modern VR graphics pipeline.

Rendering Schemes



(a) Local-only rendering



(b) remote-only rendering

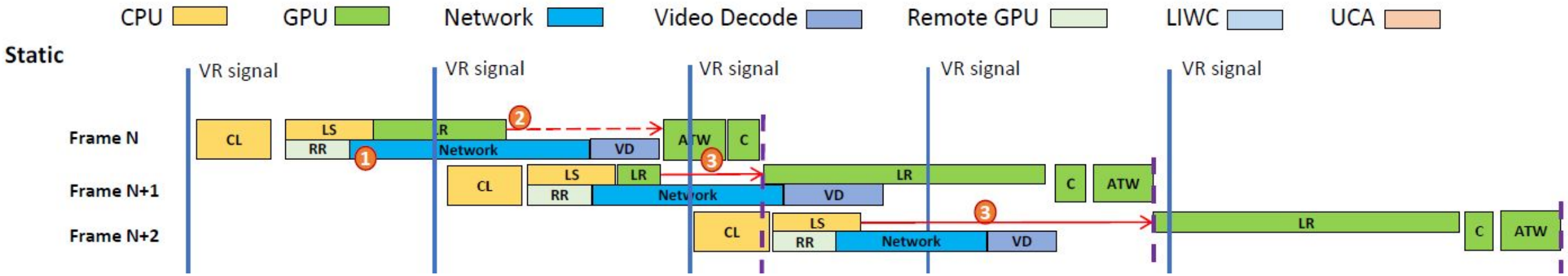
Static Collaborative Rendering

Use both local and remote rendering:

- Local Rendering
  - Handle time-critical rendering workload near the HMD display
  - Pre-defined interactive objects are often more lightweight than the background environment

- Remote Rendering
  - Let the remote system handle the rest
  - Offload background environment to the remote server.
  - Also enable pre-rendering and prefetching for the background environment

Static Collaborative Rendering Pipeline



CL: software control logic
LR: local rendering
RR: remote rendering

LS: local setup
C: composition
VD: video decoding

Static Collaborative Rendering Challenges

- Fig4- 2 is caused by misestimating hardware's realtime processing capability and the changing workload during the execution
- Fig.4- 3, several essential tasks including local rendering, composition and ATW all compete for GPU resource

- Challenge I: Design Inflexibility and Poor Programmability.
  - Random workloads due to users' actions at realtime
  - programmers are burdened to accommodate all the realtime constraints and reduce the interactive concepts
- Challenge II: Costly Remote Data Transmission

- Introduction
- Current VR Systems and Their Limitations
- Q-VR
  - Software-level flexibility
  - Hardware support
- Evaluation
- Summary

Solution - QVR

Software-hardware co-design solution for low-latency high-quality collaborative VR rendering

- Reduce T_{remote}
- Dynamically balance local and remote rendering based on realtime constraints
- Eliminate realtime hardware contention

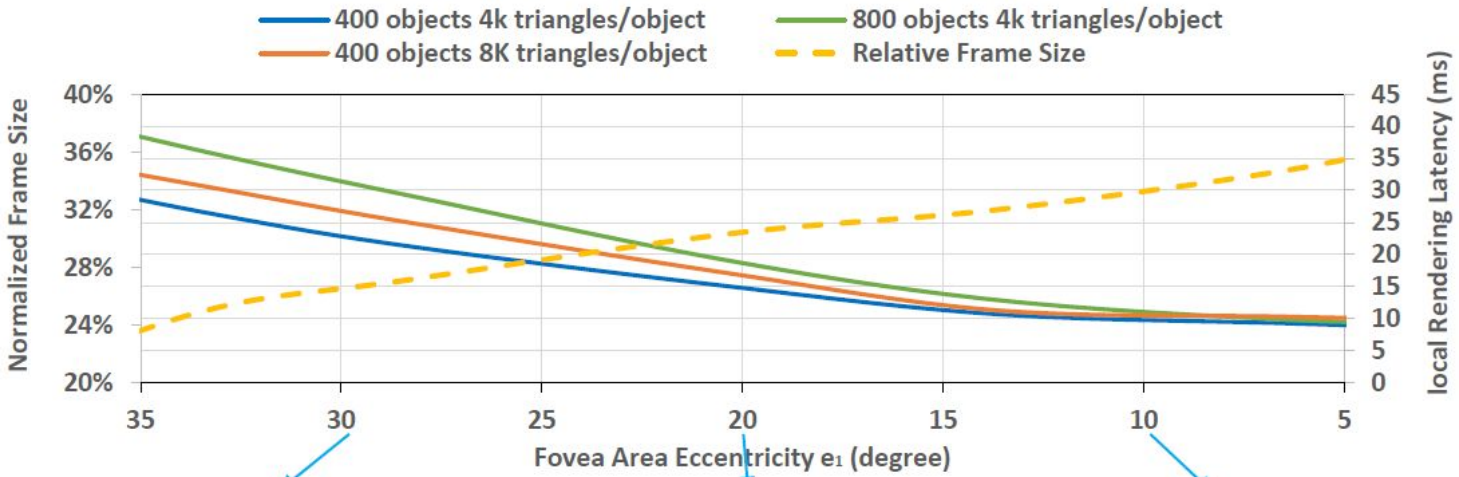
Foveated rendering

- Makes use of *foveated rendering*
  - human visual acuity falls off from the centre (fovea) to the periphery
  - only 5 degrees central fovea area requires fine details
  - for the periphery areas, the acuity requirement falls off rapidly as eccentricity increases
  - *foveated rendering,* uses an eye tracker integrated with a virtual reality headset to reduce the rendering workload by greatly reducing the image quality in the peripheral vision

Traditional foveated rendering

- Traditional foveated rendering decomposes into 3 layers:
  - foveal layer (radius = $e1$) eye tracking center
  - middle layer (radius = $e2$) employs gradient
  - outer layer which renders the periphery area with low resolution

Rendering Latency for varying e1



$e_1 = 30, e_2 = 30$ $e_1 = 20, e_2 = 35$ $e_1 = 10, e_2 = 50$

Average foveal layer rendering latency under the increasing eccentricity when running Foveated3D on Intel Gen9 mobile processor. When the eccentricity is ≤ 15 degrees, all types of scene complexities can be handled within the target latency requirements (≤ 11ms)

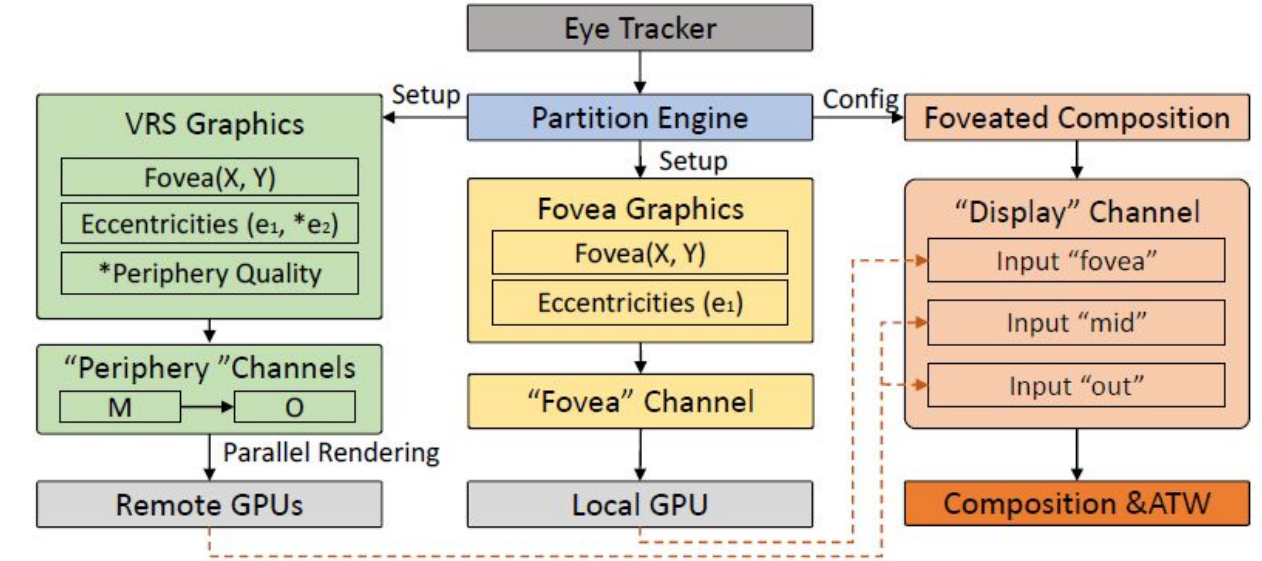
Runtime-Aware Adaptive Foveal Sizing

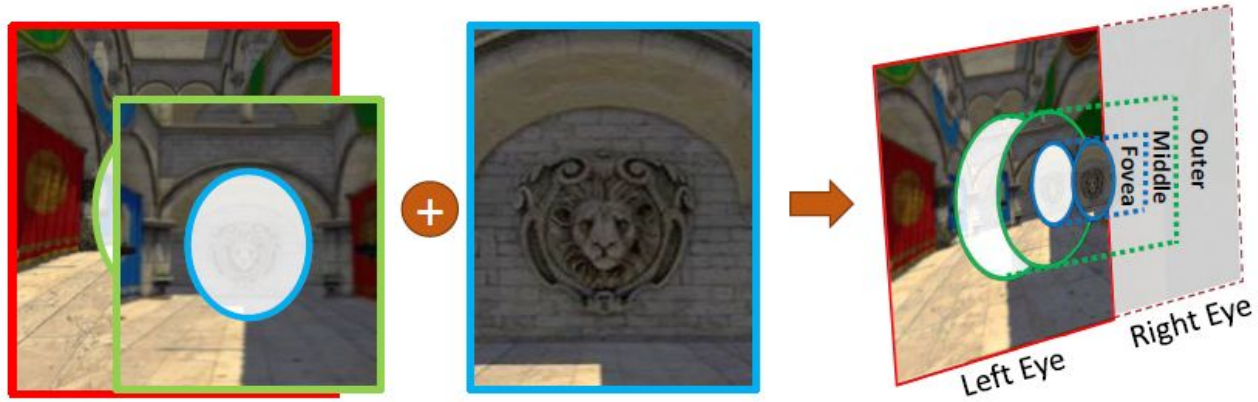
Use Partition engine to dynamically calculate the eccentricity:

- distributed rendering programming model supported by lower-level graphics libraries
- with a software tuning-knob for fine-grained fovea control and software interfaces to the graphics

- at client, gather the $foveat$ (X,Y) and $e1$ to setup the rendering viewports via VR SDK
- at server, extend the SOTA parallel VR rendering pipeline to setup multiple rendering channels for middle and outer layers with calculated eccentricity $(e1,*e2)$

An example of software-level setup and configuration in vision-perception inspired Q-VR, its programming model, and how it interfaces with hardware.






```
node {
    pipe {
      window {
       name "Periphery"
       viewport1[Fovea(X,Y), *e2]
       channel1{ name "mid" }
       viewport2[(0,0)]
       channel2{ name "out" }
       }
    }
}
```

```
node {
    pipe {
      window {
       name "Fovea"
       viewport[Fovea(X,Y), e1]
       channel{ name "fovea" }
       }
    }
}
```

```
component {
      channel {
       name "Display"
       inputframe "fovea"
       inputframe "mid"
       inputframe "out"
       outputframe "framebuffer"
       }
    }
}
```

- Introduction
- Current VR Systems and Their Limitations
- Q-VR
  - Software-level flexibility
  - Hardware support
- Evaluation
- Summary

Lightweight Interaction-Aware Workload Controller (LIWC)

- Key Design Insights
  - Local rendering latency is sensitive to the scene complexity and realtime hardware processing capability
  - Remote latency is dominated by the resolution and network bandwidth

- LIWC:
  - SRAM to store the motion-to-eccentricity mapping table which records the latency gradient offset for all pairs of motion information and eccentricity
  - latency predictor to predict the current latency for the local and remote rendering
  - motion codec to translate the motion information into table entry addresses
  - runtime updater to update the mapping table and latency prediction parameters.

LIWC

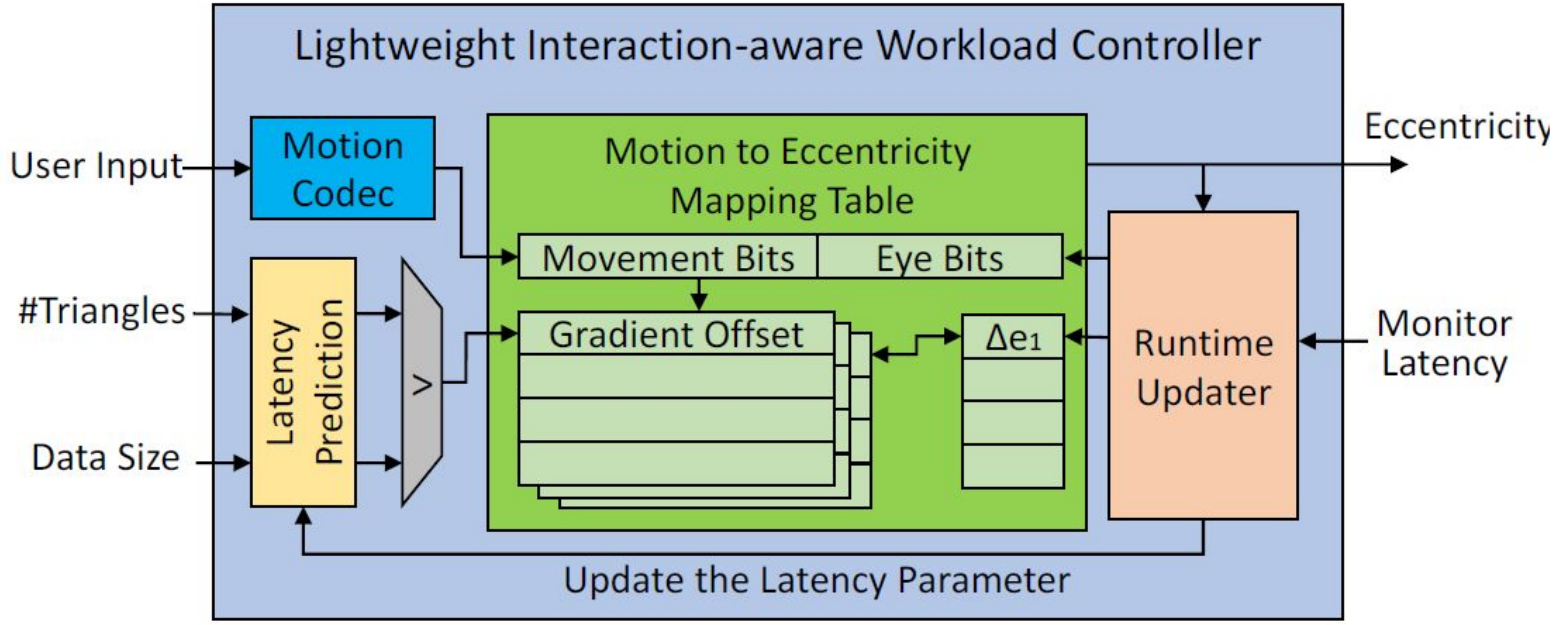


Figure 9: Architecture diagram of our proposed LIWC.

Unified Composition and ATW Unit (UCA)

Key Design insights

- 2 filtering phases:
  - Composition
  - ATW

UCA

- Combine the 2 phases
  - bypass CPU and avoid the software overhead between kernels
  - ATW starts processing the non-overlapping tiles (e.g., tiles require no composition) earlier
  - Can be executed in parallel with GPU
- UCA is implemented as a separate hardware unit on SoC
- Consists of two microarchitecture components: 4 MULs for lens distortion and 8 SIMD4 FPUs for coordination mapping and filtering

UCA

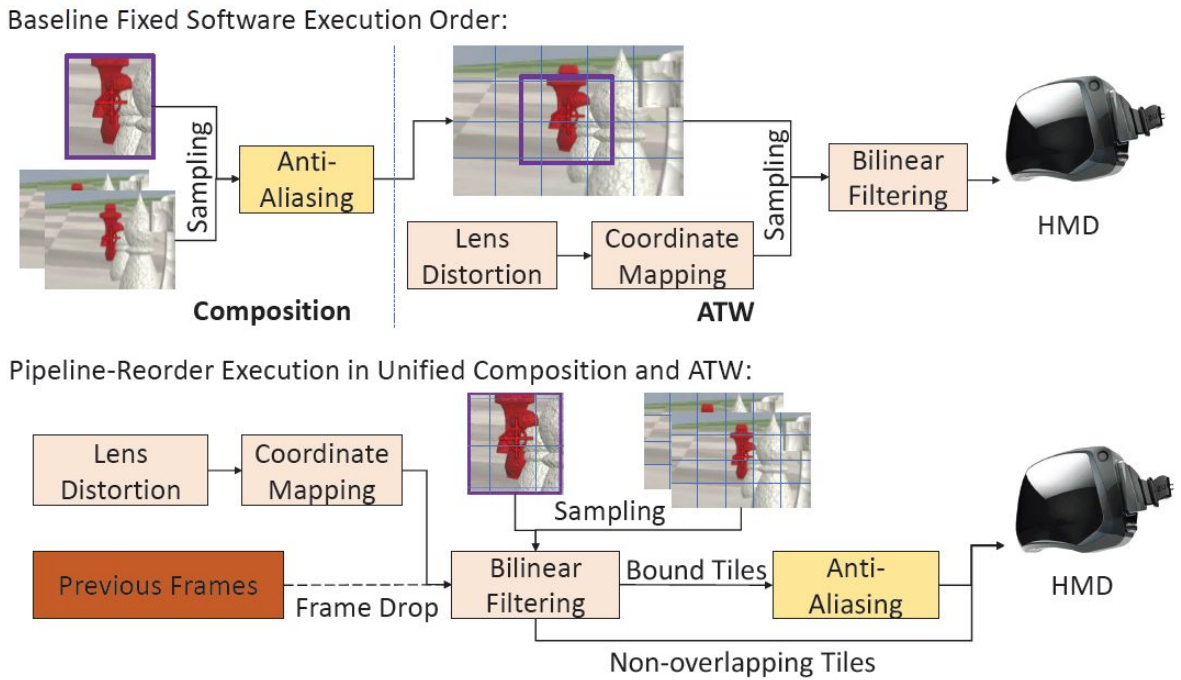


Figure 10: Comparison between baseline sequential execution and Unified Composition and ATW (UCA).

Results

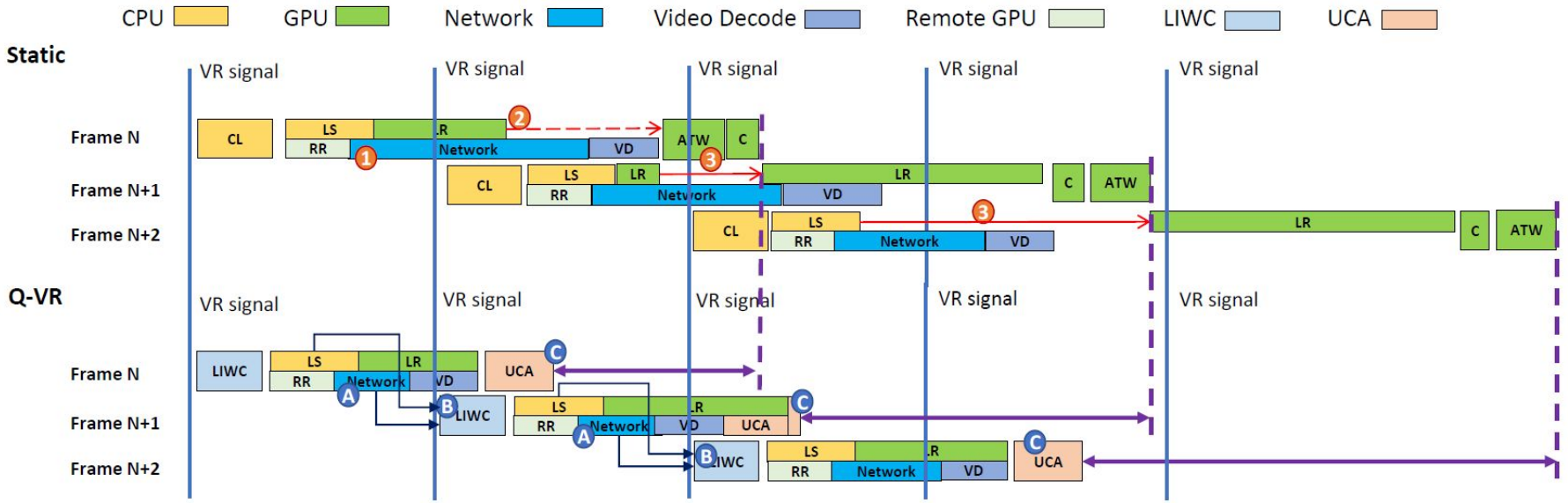


Figure 4: Execution pipeline of static collaborative rendering and our proposed Q-VR. Q-VR's software and hardware optimizations are reflected on the pipeline. Rendering tasks are conceptually mapped to different hardware components, among which LIWC and UCA are newly designed in this work. Intra-frame tasks may be overlapped in realtime (e.g., RR, network and VD) due to multi-accelerator parallelism. CL: software control logic; LS: local setup; LR: local rendering; C: composition; RR: remote rendering; VD: video decoding; LIWC: lightweight interaction-aware workload controller; UCA: unified composition and ATW.

- Introduction
- Current VR Systems and Their Limitations
- Q-VR
  - Software-level flexibility
  - Hardware support
- Evaluation
- Summary

Network Transmission

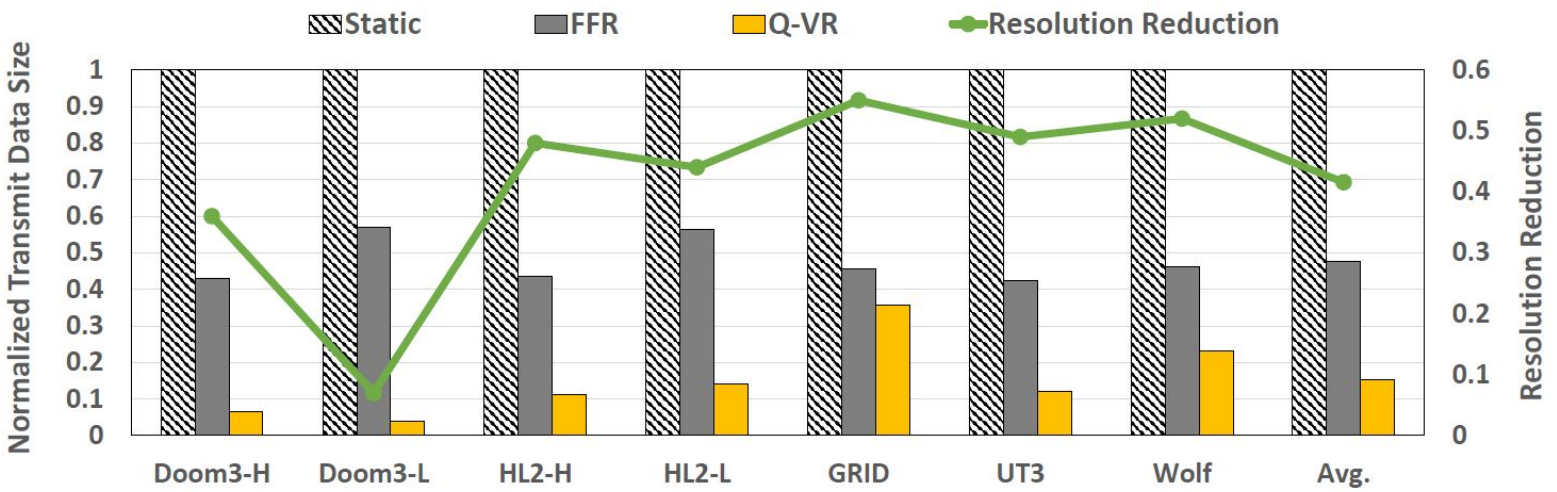
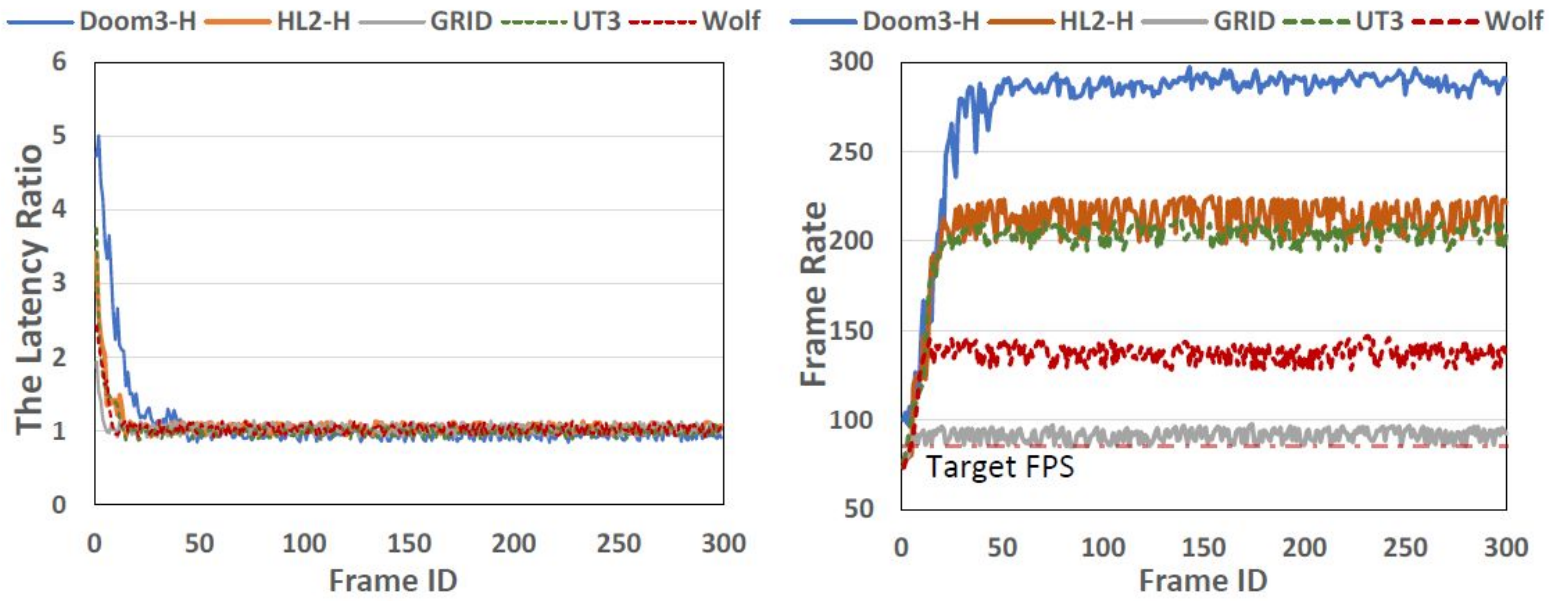


Figure 13: The normalized transmitted data size and resolution reduction from different designs under the default hardware and network. The results are normalized to the remote rendering design in commercial cloud servers.

Latency ratio and FPS



(a) Latency Ratio (b) FPS

Figure 14: The Latency Ratios and FPS across 300 Frames.

T_{remote}/T_{local}

Eccentricity Selection Under Different Configurations

Table 4: Best Eccentricity Under Different Configurations

Freq.	Net.	Benchmarks						
		D3H	D3L	H2H	H2L	GD	NFS	WF
500 MHz	Wi-Fi	46.4	85.3	27.4	33.2	9.9	27.2	15.3
	4G LTE	74.5	90	42.2	44.3	22.1	39.1	25.7
	Early 5G	22.4	45.2	11.3	14.3	5	10.9	8.6
400 MHz	Wi-Fi	34.5	77.3	23.1	26.1	7.8	22.5	13.2
	4G LTE	64.3	90	34.5	39.2	15.5	32.4	18.5
	Early 5G	15.3	30.2	7.8	11.5	5	7.4	6.1
300 MHz	Wi-Fi	27.5	65.4	16.4	24.5	6.5	14.3	11.3
	4G LTE	43.2	90	30.2	35.1	12.4	27.2	16.4
	Early 5G	13.1	27.1	6.9	8.3	5	6.1	5

Performance

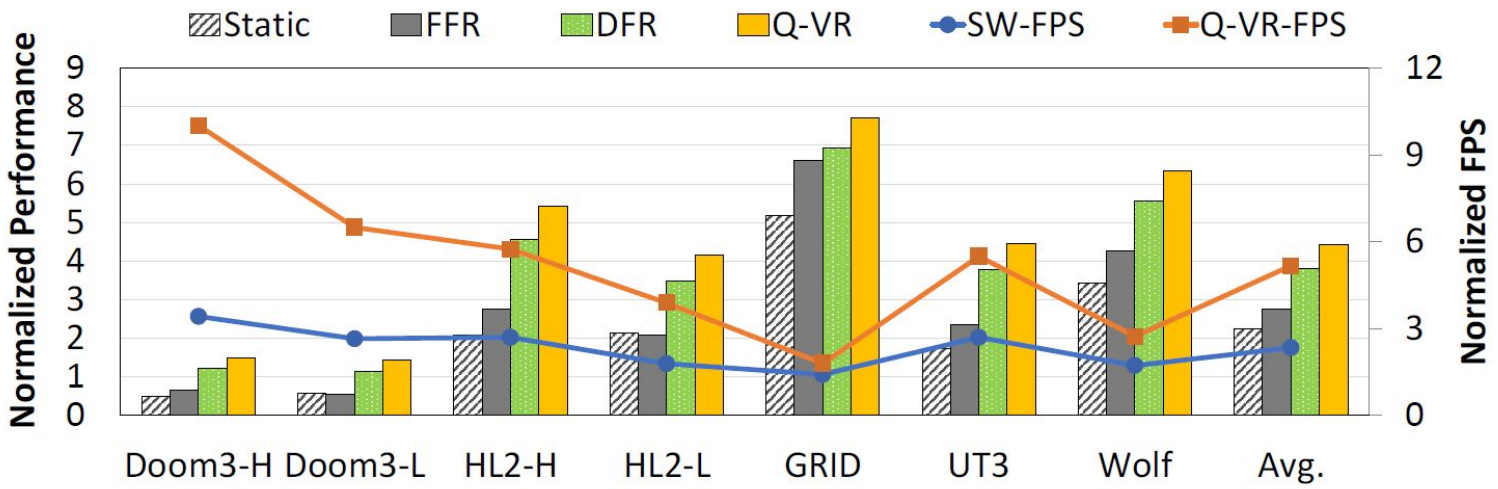


Figure 12: The normalized performance improvement from different designs under the default hardware and network. The results are normalized to the traditional local rendering design appeared in today's mobile VR devices.

- Introduction
- Current VR Systems and Their Limitations
- Q-VR
  - Software-level flexibility
  - Hardware support
- Evaluation
- Summary

Summary

- Identify the fundamental limitations of the state-off-the art collaborative rendering design
- Design the first software-hardware co-designed collaborative rendering architecture
- Create software level flexibility to reduce network limitation
- Introduce 2 new hardware components

Thank You