On Building a Programmable Wireless High-Quality Virtual Reality System Using Commodity Hardware

Ruiguang Zhong[♮◇], Manni Wang[♮§], Zijian Chen[♮⋆], Luyang Liu[♮†],
Yunxin Liu[♮], Jiansong Zhang[♮], Lintao Zhang[♮], Thomas Moscibroda[♮]
[♮]Microsoft Research, Beijing, China
[◇] Beijing University of Posts and Telecommunications, Beijing, China
[§] Xi'an Jiaotong University, Xi'an, China
[⋆] Tsinghua University, Beijing, China
[†]WINLAB, Rutgers University, North Brunswick, NJ, USA

Abstract

All existing high-quality virtual reality (VR) systems (e.g., HTC Vive and Oculus Rift) are tethered, requiring an HDMI cable to connect the head mounted display (HMD) to a PC for rendering rich graphic contents. Such a tethered design not only limits user mobility but also imposes hazards to users. To get rid of the cable, "cable replacement" solutions have been proposed but without any programmability at the HMD side. In this paper, we explore how to build a programmable wireless high-quality VR system using commodity hardware. With programmability at both the PC side and the HMD side, our system provides extensibility and flexibility for exploring various new ideas and software-based techniques in high-quality VR. We present our system design, describe challenges, explore possible solutions to cut the wire, and compare the performance of different approaches for transmitting the high-volume graphics data over a wireless link. We share our experience and report preliminary findings. Experimental results show that building a wireless high-quality VR system is very challenging, and needs extensive effort on both the software and hardware sides in order to meet the performance requirements.

1. INTRODUCTION

In recent years, Virtual Reality (VR) has become increasingly popular. In particular, existing VR head-mounted displays (HMDs) such as HTC Vive [6] and Oculus Rift [10] are able to provide high-quality VR experience by leveraging a powerful desktop PC to render rich graphics content at high frame rates and visual quality. However, current HMDs are all *tethered*: they must be connected to a PC via a USB cable for sending sensor data from the HMD to the PC and an HDMI cable for sending graphic content from the PC back to the HMD. These cables not only limit the user's mobility but also impose hazards such as a user tripping or wrapping the cable around his neck.

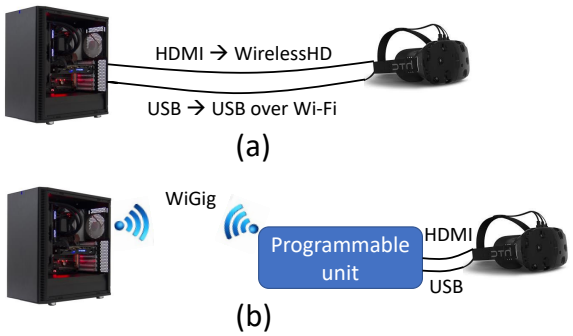

Figure 1: We propose a programmable high-quality wireless VR design. (a) Today's high-quality VR system and cable replacement for wireless. (b) Programmable wireless VR.

Ideally, the cables used to tether the HMD with a PC should be replaced by a wireless link. VR HMD companies have been exploring the use of 60GHz wireless communication to enable high-quality wireless VR [1, 7, 11], and propose a "cable replacement" solution. For example, HTC has revealed a wireless upgrade kit for HTC Vive, built in collaboration with the TPCAST wireless module [7]. This module is based on the proprietary WirelessHD [17] standard and implements the HDMI interface over 60GHz wireless with a special compression algorithm. It can barely support the

*This work was done when Ruiguang Zhong, Manni Wang, Zijian Chen and Luyang Liu were research interns at Microsoft Research.

APSys '17, September 2, 2017, Mumbai, India
 ISBN 978-1-4503-5197-3/17/09…$15.00
DOI: https://doi.org/10.1145/3124680.3124723

transmission of the current HTC Vive's graphics contents of 2160x1200 pixels at 90Hz. It also requires Wi-Fi to replace the USB cable, as shown in Figure 1(a). However, such a "cable replacement" solution is not programmable and thus has limited extensibility and flexibility. It renders it impossible to explore various techniques in software to improve the system performance.

In this paper, we report on our efforts to build a wireless high-quality VR system. We focus on two key aspects. First, the system is developed using *commodity hardware* so that it is flexible and easy to reproduce by other researchers and commercial organizations. Second, different from the "cable replacement" approach, we provide *programmability* to the system. To this end, we propose to use WiGig [16] to provide an IP network between a HMD and a PC. As shown in Figure 1(b), we add a programmable unit to a HMD to receive and process the graphics contents streamed from a PC, as well as to send the sensor data from the HMD to the PC. As a result, we provide programmability at both the PC side and the HMD side of our system for maximum extensibility and flexibility. With full programmability, our system can be used to explore various research techniques on both the PC and the HMD side, and collaboratively improve the VR performance and user experience. Example techniques include different compression algorithms, content prefetching, pose prediction and collaborative rendering (some of them will be discussed in §6). As new VR HMDs are targeting at ever higher resolutions and frame rates, e.g., 4K or 8K at 120Hz, these advanced techniques may play a critical role in future VR systems that are hard to implement without programmability.

In the following sections, we present our system design and the challenges (§2), describe how to replace the wire with wireless connection using commodity hardware (§3), explore different approaches to transmit high-volume graphic data from a PC to a HMD (§4), compare their performance (§5), discuss the limitations of our work and the potential techniques to further optimize the system performance (§6), survey the related work (§7), and conclude our work (§8).

2. SYSTEM DESIGN AND CHALLENGES

Figure 2 shows the architecture of our proposed system design, focusing on the graphics processing. VR apps run on the PC and use its powerful GPU to render rich graphic contents. The rendered frames are transmitted to the HMD over a wireless link. The HMD is integrated with a programmable device that processes the received frames from the PC and displays them on the HMD. Such a programmable device is expected to be small, portable and low power.

At the high level, the above architecture is similar to thin-client systems. However, the key difference is system performance. As the graphic rendering is accelerated using the GPU on the PC, existing tools like Microsoft Remote Desktop [9] and RealVNC [12] cannot be directly used to stream the graphic contents from the PC to the HMD. Indeed, high-quality VR apps (e.g., high-end 3D VR games) impose very high requirements and challenges on the system performance.

Display resolution (pixels)	Raw data rate (Gbps)
2048x1080 (2K)	4.8
2160x1200 (HTC Vive)	5.6
3840x2160 (4K UHD)	17.9
7680x4320 (8K UHD)	71.7

Table 1: Raw data rates of different display resolutions at a frame rate of 90Hz, with three bytes per pixel.

Requirements. High-quality VR apps require high network throughput and low end-to-end system latency. Table 1 shows the required data throughput in different display resolutions with a frame rate of 90Hz. Assuming we use three bytes to encode the RGB data of each pixel [1], without compression, the raw data rate of HTC Vive (2160x1200) is 5.6Gbps, much higher than all the existing wireless-communication products including Wi-Fi and 60GHz wireless communication [2]. In the cases of 4K UHD and 8K UHD, the required data rates are even as high as 17.9Gbps and 71.7Gbps, respectively.

As for latency, with a frame rate of 90Hz, the system must be able to render, transmit, and display a high-resolution frame every 11ms, to ensure a smooth user experience. For future VR targeting at a frame rate of 120Hz, the frame time is even reduced to only 8.3ms. Furthermore, high-quality VR also requires a total end-to-end (i.e., motion-to-photon) latency of 20-25ms [20]. That is, once the HMD moves, the system must be able to display a new frame generated from the new pose of the HMD within 20-25ms.

Challenges. Meeting the above requirements are very challenging. As the required data rate is beyond the capability of existing wireless-communication products, compression is necessary to reduce the data traffic transmitted over the wireless link. However, compressing and decompressing the data also generate extra overheads and thus make it even more challenging to achieve the required system latency.

As shown in Figure 2, decoupling the HMD from the PC introduces multiple extra steps in the data-processing procedure. In the original tethered case, VR apps render their graphic contents using GPU via the VR SDK and graphics stack, and the rendered results are directly sent to the HMD via an HDMI cable. However, in Figure 2, we introduce the following extra steps: the sender retrieves a rendered frame from the GPU to the host memory using the system graphics API (❶), compresses the frame to reduce the data size (❷), sends the compressed data to the NIC (network interface card) via the network stack (❸), the NIC transmits the data from the PC to the HMD via the wireless link (❹), the receiver receives the compressed data (❺), decompresses

[1] In modern graphics systems, a pixel is represented with four bytes in the RGBA color space where "A" means Alpha channel. As we always transmit the final frames of full screen, we can remove the Alpha-channel byte to reduce the data volume by 25%.

[2] Although the specifications of 60GHz wireless communications define data rates higher than 5.6Gbps, existing products can only achieve less than 2.5Gbps data throughput.

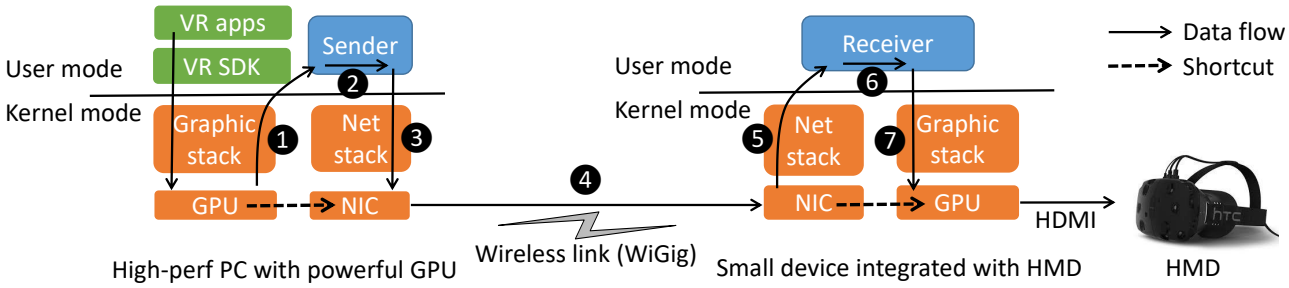

Figure 2: System architecture.

the data (❻), and passes the decompressed data to the GPU (❼).

The above seven extra steps all introduce extra latency. Furthermore, there are multiple extra data copies during the data processing. On the PC side, the frames are first copied from the GPU memory to host memory and then copied from the sender to the NIC. On the HMD side, the data also must be copied from the NIC to the receiver and then to the GPU. Given the high data volume, those data copies also impose significant latency.

3. CUTTING THE WIRE

We have built a prototype wireless VR system using commodity hardware. To cut the wire of a HMD, we studied the commodity WiGig products from two vendors: Intel and Qualcomm. As all high-quality HMDs including HTC Vive used in our system only work on Windows platform but the Qualcomm WiGig modules do not support Windows due to the lack of driver, we choose to use the Intel WiGig modules to build our wireless VR system.

In our system, we use a desktop PC with an Intel Core i7-4790 CPU, 16GB memory and a NVIDIA TITAN X GPU, to run VR apps and render all the graphic contents. We use a Dell XPS M3800 laptop to act as the programmable unit which receives graphic contents from the PC. The laptop has a NVIDIA K1100M GPU. We connect the laptop to a HTC Vive HMD via HDMI and USB. Both the PC and the laptop run Windows 10.

Specifically, we use Intel WiGig modules of 17265 and W13100 to setup a wireless IP network link between the PC and the laptop. On the PC side, we use an Intel WiGig NIC [8] that is based on Intel 17265 module. We connect the NIC to the PC through PCIe. On the HMD side, we use a Dell WiGig dock station [3] that is based on Intel W13100 module. However, the dock station provides only Ethernet port and USB port for data transmission, which limits the achievable data throughput of the dock station. We connect the dock station to the laptop via a Gigabit Ethernet cable and a USB 3.0 cable. Consequently, the HMD sends its sensor information through USB to the dock station, and the dock station receives the rendered graphic contents and passes them back to the laptop through Ethernet.

Performance of the wireless link. We measured the performance of the wireless link in our system. We used Ping to measure the round trip time (RTT) and Iperf to measure the network throughput of TCP. The results are shown in Table 2. On average, the RTT was less than 2ms and the throughput was 851Mbps. Note that we used a 1Gbps Ethernet cable to connect the Intel WiGig card to the laptop and thus the network bottleneck was indeed the Ethernet cable, which is a limitation of our current prototype. According to WiGig specification, the physical data rate is up to 7Gbps and the data throughput at the application level is expected to be up to 4.7Gbps. We also measured the throughput of Qualcomm WiGig modules (on Linux) and it was 2.5Gbps, without the bottleneck of Ethernet cable. We expect that Intel WiGig chipset has a similar throughput without the Ethernet bottleneck. In the future, we plan to remove the Ethernet bottleneck and connect the WiGig module into the laptop directly, so that the full WiGig bandwidth can be utilized.

Round trip time (RTT)	< 2ms
Throughput	851Mbps

Table 2: Latency and throughput of the wireless link.

4. TRANSMITTING GRAPHIC DATA

As aforementioned, the key challenge in wireless VR systems is how to efficiently transmit the high-volume graphic data from a PC for rendering to a HMD to meet the throughput and latency requirements. Using our prototype system, we explore the following three approaches in this paper.

CPU-based. As shown in Figure 2, a straightforward approach is writing a user-mode Sender program on the rendering PC to retrieve each rendered frame, compress it, and send the compressed data over the network. On the HMD side, the Receiver program receives the compressed data from the network, decompresses it, and displays the frame. All the data compression and decompression are done using the CPU at the both sides. We call this approach *CPU-based* and treat it as the baseline approach to be optimized. We use the Windows Desktop Duplication API to capture rendered frames and compress them into the JPEG format.

GPU-based. As all frames are rendered by GPU and stored in GPU memory, one natural method to optimize the CPU-based approach is using GPU to compress frames. This *in-place* compression reduces the memory-copy cost of copying

large frames from GPU memory to host memory [3]. Also, as the JPEG encoder divides a large frame (2160x1200 pixels in HTC Vive) into small blocks (8x8 pixels by default) and encodes each block individually, we may leverage many GPU cores to compress the blocks in parallel. As a result, the compression time can be significantly reduced. Similarly, on the HMD side, we can also use GPU to decompress the received frames. In our implementation, we use the CUDA API [2] for in-GPU JPEG compression and decompression.

GPU+pipeline. The GPU-based approach may be further optimized by using parallel pipelines. By carefully arranging the data-processing pipelines, we may make the data compression, data transmission, and data decompression parallel. In specific, on the rendering PC, we divide a large frame into multiple block groups [4] and compress each block group one by one. After one block group is compressed, we immediately use a separate thread to send them out over the wireless link. At the same time, we continue to compress the next block group. Similarly, at the HMD side, once a compressed block group is received, we immediately start to decompress it. Therefore, we are able to shorten the total time of the end-to-end data processing.

5. EVALUATION

We have conducted a set of experiments to evaluate the performance of the three data-transmission approaches, in terms of frame-processing time and end-to-end latency.

5.1 Experimental Setup

We implemented the three approaches described in in §4. We use the desktop PC mentioned in §3 as the rendering PC and use a HTC Vive as the HMD. At the HMD side, besides the Dell laptop mentioned in §3, we also use another PC to study what computation power is needed to decompress the received frames. The PC has a NVIDIA GeForce 970 GPU which is more powerful than the GPU of the laptop but still less powerful than the rendering PC's GPU. We conducted two sets of experiments for these two cases and name them as `PC` and `Laptop`, respectively. We repeated each experiment for five times and report the average results. All the three devices (two PCs and one laptop) run Windows 10.

5.2 Experimental Results

Frame-processing time. Table 3 shows the frame-processing time including the total time and the time spent in the following steps: render a frame using the GPU (*Render*), capture the rendered frame (*Capture*), compress the frame (*Compress*), transmit the compressed frame over the wireless link (*Send/recv*), and decompress the received frame (*Decompress*). To measure these times, we used "The Lab" VR game [14].

In all the three approaches, frames were always rendered on the GPU of the rendering PC. Thus, the frame-rendering time was the same in any approach. It took 6.1ms to render a frame. However, the CPU-based approach took more than 13ms to capture a frame, much longer than the capture time of only 0.2ms in other approaches. This is because in the CPU-based approach, the data of the whole frame (more than 10MB for 2160x1200 pixels) must be copied from the GPU memory to the host memory, and the Desktop Duplication API must wait for a full frame ready to be copied out. In other approaches, we directly locate the address of the rendered frame in the GPU memory and do not need to copy the data to the host memory. Consequently, the capture time was reduced significantly.

The CPU-based approach also took a significant longer time than other approaches in compressing frames. The CPU-based approach took about 110ms to compress a frame but the GPU-based approach took only less than 5ms. This is because that the GPU-based approach is able to use many GPU cores to speed up the compression. Similarly, the GPU-based approach also significantly outperformed than the CPU-based approach in data decompression. The CPU-based approach took 31ms and 55ms to decompress a frame in the `PC` case and the `Laptop` case, respectively. The corresponding numbers in the GPU-based approach were 2.4ms and 17ms, respectively.

The network transmission time was the same in all the approaches because the same JPEG-compression algorithm was used. To ensure high VR quality and good user experience, we choose 95% quality in the JPEG compression, resulting in visually-invisible difference between the compressed frame and the original frame. After the compression, it took less than 6ms to transmit a frame over the wireless link.

With in-GPU compression and decompression, the total frame-processing time was reduced from 166.9ms (`PC` case) and 189.5ms (`Laptop` case) to only 18.8ms (`PC` case) and 33.7ms (`Laptop` case), compared to the CPU-based approach. With the parallel pipelines, we could further reduce the total frame-processing time to 13ms (`PC` case) and 23.9ms (`Laptop` case) [5]. Excluding the frame-rendering time of 6.1ms, the total extra latency introduced by our approach is 6.9ms in the `PC` case and 17.8ms in the `Laptop` case. These results demonstrate that the GPU-based optimization techniques are effective in improving the system performance.

User-perceivable latency. We also measured how much end-to-end extra latency that users may perceive in the approach of GPU+pipeline, compared to the native tethered case. To do it, we connected another HTC Vive HMD to the rendering PC via an HDMI cable but just use it as a normal display without generating any sensor inputs. As a result, we could view the same VR game on the two HMDs: the local one that received frames directly via the HDMI cable and the remote one that received frames remotely via the wireless link. We used two smartphones to record the outputs of the two HMDs at the same time at 240 frames per second. We also generate audio signals for time synchronization between

[3]We still need to copy the compressed data to host memory but the data size is much smaller.

[4]In our implementation, we divide each frame into eight block groups. This number may be further fine-tuned.

[5]Note that we attribute the gain of parallel pipeline to Compress and Decompress and keep the Send/recv time constant.

	Time (ms)											
	Render		Capture		Compress		Send/recv		Decompress		*Total*	
	PC	Laptop	PC	Laptop	PC	Laptop	PC	Laptop	PC	Laptop	PC	Laptop
CPU-based	6.1	6.1	13.9	13.4	109.9	108.9	5.9	5.9	31.1	55.2	*166.9*	*189.5*
GPU-based	6.1	6.1	0.2	0.2	4.2	4.5	5.9	5.9	2.4	17.0	*18.8*	*33.7*
GPU+pipeline	6.1	6.1	0.2	0.2	0.5	0.6	5.9	5.9	0.3	11.1	*13.0*	*23.9*

Table 3: Frame-processing time (each step and the total) of the three data-transmission approaches.

the two smartphones. By analyzing the video clips recorded by the two smartphones, we could decide the time difference between the moment a frame was displayed in the local HMD and the moment the same frame was displayed in the remote HMD. Such a time difference is the extra end-to-end latency perceivable to users. On average, the measured extra end-to-end latency was 12$\pm$2ms (the error is for the 240fps video recording) in the `PC` case. Users may perceive such a latency, but the experience is acceptable. For the `Laptop` case, because the extra latency of 17.8ms is larger than the frame time of 11ms (i.e., 90Hz), the GPU cannot decompress frames in time. Thus, the frames are accumulated and the user experience is very bad. To get a smooth user experience, we have to drop frames periodically. In this case, the measured extra end-to-end latency was 25$\pm$2ms.

5.3 Observations

We make the following observations from the above experimental results. First, the CPU-based approach does not work due to the poor performance in compressing and decompressing frames. Second, using GPU can effectively help improve the performance of compression and decompression, and the parallel pipeline technique can further reduce the extra latency in processing frames. However, even with the optimizations of using GPU and parallel pipeline, it is still hard to meet the performance requirement of high-quality VR, unless we use a very powerful GPU to decode the frames. However, in practice, we cannot attach a very powerful device to a HMD due to the large size and the high power consumption.

One possible way may be using special hardware. Indeed, most GPUs have dedicated hardware codec for image/video encoding and decoding. We also studied the performance of this hardware-based approach. We used the NVIDIA Video Codes SDK [15] (NVENCODE and NVDECODE) to encode and decode frames on the rendering PC and the Dell Laptop, respectively. We use the H.264 encoder and decoder with the highest quality. However, with this approach, we cannot make the encoding/decoding process parallel with the network transmission and the total frame-processing time (compression, transmission and decompression) is more than 30ms. Thus, it still cannot meet the performance requirements of high-quality VR.

Consequently, we argue that building wireless high-quality VR systems is a very challenging task and that is probably why the industry still does not have such a system available yet. New, powerful and low-power hardware is desirable to solve the problem.

6. DISCUSSION

In this section, we discuss the limitations of our work and more optimization techniques to further improve the performance of our wireless VR system.

6.1 Limitation

Our current wireless VR prototype is incomplete and has many limitations. Particularly, it cannot achieve the performance requirements of high-quality VR in terms of latency. In this paper, we focus on exploring possible solutions to enable high-quality wireless VR systems and reporting preliminary results, rather than providing a complete solution. Given the gap between the performance requirements of high-quality VR and the capability of the current hardware, we argue that software techniques will play a critical role in enabling high-quality wireless VR systems and thus we propose the programmable design.

The use of the laptop as the programmable unit is impractical for commercial products. Practical wireless HMDs must be light weighted and portable. To this end, a smartphone sized programmable unit is desirable. Power consumption is also a critical consideration. In our prototype, the HTC Vive itself and the WiGig wireless consume a small power of less than 4W (both in working state) but the power consumption of the laptop can be more than 40W, which kills a battery quickly that users could carry. We are investigating more hardware choices for a better balance among size, weight and power.

6.2 Further Optimization

In this section, we discuss more optimization techniques to further improve the performance of our wireless VR system.

Direct data passing between GPU and NIC. To shorten the data-processing pipeline in Figure 2 and further reduce the latency, it is desirable to setup a direct data path between GPU and NIC so the frames do not need to traverse the kernel/user mode boundary with multiple times of data copying. This is illustrated by the *shortcut* lines in Figure 2. At the rendering PC side, a frame is compressed and stored in the GPU memory. Instead of copying the compressed data to the host memory and then sending the data to the NIC, we may allow the NIC to directly access the GPU memory and send the compressed data over the wireless link without data copying. Similarly, at the HMD side, the NIC may directly store the received data into the GPU memory without going through the host memory. This approach is possible because that both GPU and NIC are con-

nected to the same PCIe bus, and thus they can directly access each other's memory via PCIe protocol after properly configured [22]. Besides careful configurations and memory management, doing so also requires a network stack implemented in the GPU or NIC. We plan to study how to design a simple network protocol specifically tailored for our VR scenario to enable this direct-data-passing and zero-data-copy technique between GPU and NIC.

Some NVIDIA GPUs (e.g., Tesla and Quadro GPUs) support GPUDirect RDMA [5] that enables a direct path for data exchange between a GPU and a third-party peer device using standard features of PCIe. However, GPUDirect RDMA cannot help in our VR system because there is no RDMA-enable wireless NIC available that can work with GPUDirect RDMA technology.

Collaborations between a HMD and its rendering PC. A HMD may work together with its rendering PC to further improve the VR experience. For example, with gaze tracking on a HMD, we may send high-resolution pixels only for the area of the display where the user is looking at. For the rest of part of the display, we may provide low-resolution images (e.g., by compressing the images deeply) without compromising the user experience. This selective-compression approach may be very valuable for extremely-high-resolution HMDs, e.g., 4K, 8K or even higher, as the required data throughput may be far beyond the capacity of the possible wireless communications. Another example is frame prefetching and caching. Based on predicted pose of the HMD, the PC may pre-compute future frames and push them to HMD before they are requested. If the pose predication is correct, the required frames are immediately ready in the HMD and thus the latency is significantly reduced. Similarly, by caching previous frames, if the HMD cannot receive a required new frame in time, it may use its current pose and cached frames to generate a new frame to ensure smooth user experience. Furthermore, the HMD and its rendering PC may render frames in a collaborative matter, e.g., each rendering different frames or different parts of a frame. Such a collaborative rendering approach may fully utilize the computation power of both the rendering PC and the HMD and thus further reduces the latency.

Exploring these advanced techniques requires programmability not only at the rendering PC side but also at the HMD side, which is the value that the system we have proposed in this paper may provide.

7. RELATED WORK

Standalone VR systems. Besides the tethered VR HMDs like HTC Vive [6] and Oculus Rift [10], there are also self-contained, portable VR systems such as Samsung Gear VR [13] and Google Cardboard [4] that run VR apps by sliding a smartphone into the headset. Those VR systems are limited by the capability of smartphones, cannot run high-quality VR apps, and thus are not the focus of this paper.

High-quality wireless VR. There is a huge interest in high-quality (PC-based) wireless VR systems. HTC [7], Optoma [11] and SiBeam [1] announced 60GHz wireless VR products but none of them are publicly available yet. No or very few details on those products are provided. All of them are not programmable. Our goal is to provide a programmable high-quality wireless VR system so that researcher and developers can leverage it to explore many different techniques and ideas.

VR research. The prosperity of modern VR systems has also attracted many interests in research communities. For example, MoVR [18] tries to solve the blocking and mobility issues of mmWave radios. MoVR built a mmWave reflector that is able to reconfigure its angles of incidence and reflection to mitigate the effect of mmWave blockage. Our work is complementary to MoVR and can benefit from MoVR reflector to improve the network performance. FlashBack [20] proposes to use pre-rendered graphics cache to improve VR experience on mobile devices. It is also complementary to our work.

Game streaming and offloading. In the higher-level context of gaming, Outatime [23] is a speculative execution system to mask network latency in streaming games from cloud to mobile devices. It speculatively renders frames of future for low-latency continuous interactions of mobile cloud gaming. Kahawai [21] is a system that provides high-quality gaming on mobile devices through offloading the GPU computation to a nearby server. In Kahawai, a mobile device and a nearby server work together to render game contents. For example, the mobile device may render frames with reduced detail and the server renders rich graphic contents with more details. The ideas of Outatime and Kahawai may be applied to VR systems. They require full programmability at both ends and thus can benefit from our design and is complementary to our work.

Wireless communication. Wireless communication is a fundamental enabling technique for high-quality wireless VR systems. Given the limited data throughput provided by Wi-Fi, the trend is exploring higher-frequency radios such as 60GHz wireless communication. Existing efforts include WirelessHD [17] and WiGig [16] etc. However, compared to Wi-Fi, 60GHz wireless communication relies on directional transmission and tends to require light of sight, and thus has the issues of blockage and mobility. For our wireless VR scenario, we require sender antenna to be placed on ceiling, thereby providing light of sight transmission to HMD in most cases. On the other hand, we expect the recent active research on mobility and blockage handling [25, 26, 19, 24] will greatly mitigate the issues.

8. CONCLUSION

In this paper, we have explored how to build a wireless high-quality VR system. Using commodity hardware, our aim is to build a ready-to-use system for researchers. By providing programmability at both the rendering PC side and the HMD side, our system enables researchers to explore various new ideas and software-based techniques. We implement three approaches for transmitting the high-volume

graphics data and conduct experiments to evaluate their performance. Experimental results show that our optimizations of using GPU and parallel pipeline significantly improve performance, but they are still not enough to meet the performance requirements of high-quality VR systems. We also discuss other advanced techniques for further improving the system performance. New hardware and new software techniques such as collaborative rendering are desirable to enable wireless high-quality VR systems.

9. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers and our shepherd, Prashant Shenoy, for their valuable comments.

10. REFERENCES

[1] 60 GHz: Taking the VR Experience to the Next Level. http://www.sibeam.com/en/Blogs/2016/March/60GHZTakingtheVRExperience.aspx.

[2] CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit.

[3] Dell Wireless Dock. http://www.dell.com/en-us/shop/accessories/apd/452-bbux?sku=452-BBUX.

[4] Google Cardboard. https://vr.google.com/cardboard/.

[5] GPUDirect RDMA. http://docs.nvidia.com/cuda/gpudirect-rdma/index.html.

[6] HTC Vive. https://www.vive.com/.

[7] HTC Vive TPCAST wireless upgrade kit. https://www.vive.com/cn/accessory/tpcast/.

[8] Intel Tri-Band Wireless-AC 17265. https://www.intel.com/content/www/us/en/wireless-products/tri-band-wireless-ac-17265.html.

[9] Microsoft Remote Desktop Clients. https://technet.microsoft.com/en-us/library/dn473009(v=ws.11).aspx.

[10] Oculus Rift. https://www.oculus.com/rift/.

[11] Optoma's wireless VR headset frees you from PC cables. http://www.pcworld.com/article/3044542/virtual-reality/optomas-new-wireless-vr-headset-frees-you-from-pc-cables.html.

[12] RealVNC: Remote Access Software for Desktop and Mobile. https://www.realvnc.com/.

[13] Samsung Gear VR. http://www.samsung.com/global/galaxy/gear-vr/.

[14] The Lab VR game. http://store.steampowered.com/app/450390/.

[15] The NVIDIA Video Codes SDK. https://developer.nvidia.com/nvidia-video-codec-sdk.

[16] Wireless gigabit Alliance. http://www.wigig.org/.

[17] WirelessHD. http://www.wirelesshd.org/.

[18] O. Abari, D. Bharadia, A. Duffield, and D. Katabi. Enabling high-quality untethered virtual reality. In *Proceedings of the NSDI'17*, 2017.

[19] O. Abari, H. Hassanieh, M. Rodriguez, and D. Katabi. Millimeter wave communications: From point-to-point links to agile network connections. In *Proceedings of the HotNets'16*, 2016.

[20] K. Boos, D. Chu, and E. Cuervo. Flashback: Immersive virtual reality on mobile devices via rendering memoization. In *Proceedings of the MobiSys'16*, 2016.

[21] E. Cuervoy, A. Wolmany, L. P. Coxz, K. Lebeck, A. Razeenz, S. Saroiuy, and M. Musuvathi. Kahawai: High-quality mobile gaming using gpu offload. In *Proceedings of the MobiSys'15*, 2015.

[22] S. Kato, J. Aumiller, and S. Brandt. Zero-copy i/o processing for low-latency gpu computing. In *Proceedings of the ICCPS'13*, 2013.

[23] K. Lee, D. Chu, E. Cuervo, J. Kopf, Y. Degtyarev, S. Grizan, A. Wolman, and J. Flinn. Outatime: Using speculation to enable low-latency continuous interaction for cloud gaming. In *Proceedings of the MobiSys'15*, 2015.

[24] T. Nitsche, A. B. Flores, E. W. Knightly, and J. Widmer. Steering with eyes closed: Mm-wave beam steering without in-band measurement. In *Proceedings of the INFOCOM'15*, 2015.

[25] M. E. Rasekh, Z. Marzi, Y. Zhu, U. Madhow, and H. Zheng. Noncoherent mmwave path tracking. In *Proceedings of the HotMobile'17*, 2017.

[26] S. Sur, X. Zhang, P. Ramanathan, and R. Chandra. Beamspy: Enabling robust 60 ghz links under blockage. In *Proceedings of the NSDI'16*, 2016.