DeltaVR: Achieving High-Performance Mobile VR Dynamics through Pixel Reuse

Yong Li
Department of EECS
University of Tennessee, Knoxville
yli118@vols.utk.edu

Wei Gao
Department of Electrical and Computer Engineering
University of Pittsburgh
weigao@pitt.edu

ABSTRACT

Virtual Reality (VR) improves the user's experience when interacting with the virtual world, and could revolutionarily transform the designs of many interactive systems. However, providing VR from untethered mobile devices is difficult due to their limited local capabilities. Existing VR solutions address this difficulty by rendering VR frames at remote computing facilities, but are limited to rendering every VR frame separately. A tremendous amount of VR frame data, hence, needs to be transmitted to mobile devices over low-bandwidth wireless links and seriously impairs VR performance. In this paper, we aim to remove this performance constraint on highly dynamic VR applications with complicated scenes and intensive user movement, by adaptively reusing the redundant VR pixels across multiple VR frames. We leverage the unique characteristics of image warping used in current VR applications, and fundamentally expand the scope of image warping to the entire VR lifespan to precisely capture the fluctuations of VR scene due to VR dynamics. We implemented our design over Android OS and Unity VR application engine, and demonstrated that our design can maximize the mobile VR performance over highly dynamic VR scenarios with 95% less amount of VR frame data being transmitted, by completely removing the pixel redundancy across VR frames.

CCS CONCEPTS

• **Human-centered computing → Ubiquitous and mobile computing systems and tools**;

KEYWORDS

Virtual Reality, Mobile Devices, Dynamics, Frame Redundancy, Pixel Reuse

1 INTRODUCTION

Virtual Reality (VR) stimulates users' immersive senses of the virtual world, and improves user experiences in gaming [13], automobiles [44] and healthcare [18]. Ideally, VR should be provided through untethered mobile head-mounted displays (HMDs) that project rendered frames from the connected smartphones, to be usable anytime and anywhere with low cost. However in practice, smartphones have too limited local capacity to ensure high rates ($\geqslant$60 FPS), low motion-to-photon latency ($\leqslant$20ms) and wide field of views (>120°) when rendering high-resolution VR frames [25]. Their VR performance, hence, are much lower than that of their tethered counterparts (e.g., Oculus Rift [20] and HTC Vive [19]).

A viable solution to this challenge is to offload the expensive VR frame rendering from mobile HMDs to other stronger computing facilities such as the remote cloud, and then wirelessly transmit the rendered frames back for local display. However, current solutions to such mobile workload offloading [16, 22, 23, 27], when being applied to VR applications, fail to provide satisfactory VR performance because their amounts of VR frame data being transmitted will far exceed the capacity of any existing wireless network. For example, more than 2GB of frame data needs to be transmitted every second for a VR application with 4K resolution and 60 FPS [56], but only a small portion of such data can be timely transmitted to mobile HMDs even through gigabit WiFi. Emerging mm-wave wireless could potentially provide the required network bandwidth [9, 10], but requires line-of-sight connectivity which may not be always available in complicated indoor scenarios.

The fundamental reason of such failure is that the cloud *separately* renders and transmits every VR frame to the mobile HMD, and the amount of wireless data transmission is proportional to the VR frame rate and resolution. The majority of VR frame data being transmitted, however, could be *redundant* and wasted in practice. We have experimentally observed that consecutive VR frames are highly correlated because of the unique methods of VR object projection, and redundancy among these frames could be at least 50%, i.e., more than half of pixels in these frames are identical with each other. Removing such redundancy and reuse pixels across multiple VR frames, obviously, is the key to reduce the burden on wireless data transmission for satisfiable mobile VR performance.

An intuitive solution to such pixel reuse is to adopt traditional video encoding techniques such as H.264 [54], which only transmit the key reference frames in full and encode the other frames' differences from these reference frames as delta images. Similar techniques have also been used in 360° video streaming to apply different compression ratios and pixel resolutions for different regions of a panoramic frame being transmitted [34, 55]. However, these approaches fail on VR applications, which are fundamentally different from video streaming due to the unexpected user movements or actions in the virtual world. Such VR user behaviors result in heterogeneous *VR dynamics* such as object reprojection (due to user head rotation), sudden acceleration and shape change (e.g., explosion), and hence reduce the effectiveness of traditional video

IPSN '19, April 16–18, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6284-9/19/04…$15.00
https://doi.org/10.1145/3302506.3310385

frame encoding that is agnostic about the specific VR frame structure and hence always apply fixed encoding strategies. For example, H.264 addresses frame object dynamics through inaccurate motion estimation [58], but can only remove up to 40% redundancy across VR frames. Reducing the pixel resolution of a fixed region in frames, on the other hand, may seriously impair the image quality of key VR objects in this region and hence affect user experience.

To address these limitations, in this paper we present *DeltaVR*, a systematic mobile VR framework that maximizes the mobile VR performance by flexibly adapting the VR frame encoding strategy to the instantaneous VR dynamics. Instead of encoding VR frames based on inaccurate prediction of VR dynamics in the future, DeltaVR leverages the existing VR applications' capabilities of continuously tracking the camera view changes, so as to ensure that the calculation of delta images precisely captures the momentary fluctuations of VR scenes. More specifically, DeltaVR fundamentally expands the scope of image warping, which is widely used by current VR applications to ensure VR image quality via user view tracking and reprojection, from two consecutive VR frames to the entire VR lifespan. As a result, delta images between every two reference frames are calculated by warping a reference frame over different distances, and the size of delta image is hence minimized by avoiding pixel redundancy during such warping process. Such design of DeltaVR leads to the following two unique features.

First, DeltaVR is widely applicable to highly dynamic VR applications. Even when being applied to highly interactive VR games where the user character constantly moves and interacts with foreground objects in the virtual world, DeltaVR can always ensure satisfiable VR frame rates and motion-to-photon latency by eliminating the redundancy among dynamic VR objects, but requires very low wireless network bandwidth for transmitting VR frame data. Hence, it fundamentally outperforms existing mobile VR solutions, which are limited to reusing pixels of the panoramic background image only when the user character remains stationary in the virtual world [13, 17, 28].

Second, DeltaVR encodes VR frames without predicting the VR user behavior or prefetching any VR frame data based on such prediction. It hence minimizes the degradation of VR image quality with high VR dynamics, which cause large amounts of prediction errors. As a result, being different from traditional VR schemes whose performance could be easily impaired by sporadic events or user movements in VR applications [17, 29, 30], the performance of mobile VR applications supported by DeltaVR is only determined by their view resolution and wireless link condition. On the other hand, DeltaVR can also be applied to complement these traditional techniques and reduce their prediction errors, by providing a minimum latency of transmitting VR frame data.

We have implemented DeltaVR over Android OS and Unity VR appl engine[1] as a mobile middleware between VR applications and OS drivers, so as to ensure its generality over different VR applications with heterogeneous dynamics and computation demands. More specifically, DeltaVR is implemented in native language within the Android OS kernel, and we utilize the unified OpenGL APIs for graphics operations such as VR frame rendering, so as to tackle the heterogeneity of shading languages and scripting APIs used by different VR applications. The implementation consists of ~4,000 Lines of Codes (LoC) in total, and our experimental results over real-world VR applications show that DeltaVR can maximize the VR performance over highly dynamic VR scenarios with complicated scenes and intensive user movement, while reducing more than 95% of the VR frame data being wirelessly transmitted.

[1]The Unity engine (https://unity3d.com/) is the most popular tool for commercial VR game creation.

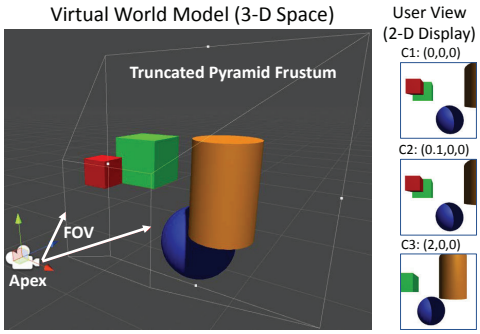


Figure 1: The virtual world in VR

(a) Viking Village (b) Lite (c) Sci-Fi

Figure 2: Screenshots of VR games

2 MOTIVATION

Our design of DeltaVR builds on the unique characteristics of frame rendering and display in VR applications. In this section, we first demonstrate that perspective object projection in VR applications results in very high pixel redundancy across VR frames, which are however, difficult to be eliminated by traditional video encoding techniques due to the heterogeneous VR dynamics. Based on this observation, we further motivate the design of DeltaVR by highlighting the possibility of efficient redundancy removal and pixel reuse through VR image warping over long distance.

2.1 Pixel Redundancy due to 3D Perspective Projection

As shown in Figure 1, VR applications construct the virtual world as a 3D space, where game objects are modeled and placed. In this 3D world, the user character is represented by a 2D camera, and the application view presented to the user is rendered by projecting each 3D object to the camera surface. Specifically, most of today's VR applications adopt *perspective projection* [43], which emulates how human eyes see the real world. Such projection forms the 3D world as a truncated pyramid frustum, with the camera at the apex point and its range defined as the camera's FOV. Any object within this frustum is projected to and visible in the user view.

Such perspective projection naturally results in high volumes of redundant pixels across VR frames, because it makes distant objects in the 3D world appear smaller than objects close-by and hence reduces the impact of VR dynamics (e.g., user behavior or VR application events) on the 2D user view. For example, when the

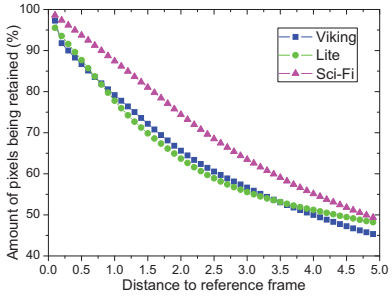


Figure 3: VR pixel redundancy

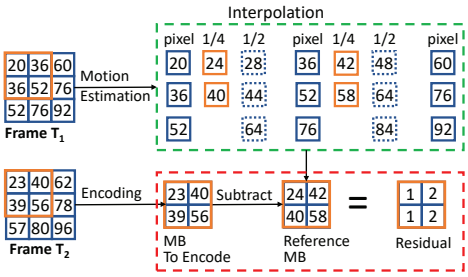


Figure 4: Quarter-pixel interpolation in H.264. Numbers indicate the pixel RGB values.

VR user changes the head orientation, the corresponding camera location change usually ranges in [-0.1, 0.1] in virtual units (~10cm in reality) and results in negligible user view change as shown in Figure 1 (from camera location C1 to C2). To further investigate such redundancy with practical VR dynamics, we randomly pick 10 reference frames from three open-sourced VR games (Viking Village [7], Lite [4] and Sci-Fi [6]), and evaluates the amount of redundant pixels between these frames to the target camera views with free user movements. As shown in Figure 2, these games represent different VR scenarios with heterogeneous scene complexity and character dynamics. Experiment results in Figure 3, then, show that more than 50% of VR pixels can be redundant, even if the distance of camera location change reaches 5.0 (~5m in reality).

2.2 Failure of Traditional Video Encoding

One straightforward solution to eliminating such VR pixel redundancy is to adopt the existing video encoding techniques for VR frame compression. Traditional video encoding techniques, unfortunately, fail when being directly applied to VR frames, because they are generally agnostic about the object projection structure of VR frames and are hence incapable of adapting to the heterogeneous VR dynamics. For example, as the most commonly used technique, H.264 compresses video frames based on their temporal and spatial locality, by dividing the current frame into several macro blocks (MBs). For each MB, it estimates the user's motion to correspondingly find the most similar MB from the previous reference frame, and then uses the per-pixel difference from this reference MB as encoding output [12]. Such motion estimation, however, cannot precisely capture the change of VR user views, which usually does not correspond to an integer number of pixels. In this case, H.264 uses quarter-pixel interpolation to generate the reference MB, but always suffers from the limited granularity of such interpolation.

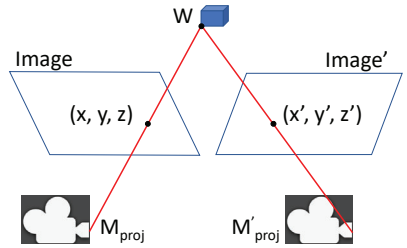
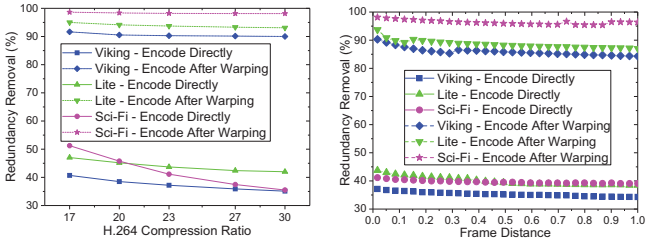


Figure 5: VR image warping



(a) Different compression ratios (b) Different reprojection distance

Figure 6: Effectiveness of image warping in removing VR pixel redundancy

An example of such limitation is shown in Figure 4, which shows two VR frames generated from camera locations that are 0.4 pixel away from each other. When the VR frame T_1 is utilized as the reference to encode another frame T_2, the residual difference calculated from quarter-pixel interpolation still retains high entropy due to the mismatch of reference MB (1/4 pixel away) and subsequent non-zero values. In practice, a residual of 4 pixels will take at least 7 bits event with context-adaptive variable-length coding (CAVLC) [40], resulting in very low ratio of frame data compression.

2.3 VR Image Warping

In contrast, our design of DeltaVR builds on the image warping technique, which is able to always capture even the smallest VR view changes. More specifically, in current VR applications where the user view or VR objects may change in the mean time when a new VR frame is being rendered, image warping is widely used to ensure image quality and avoid motion sickness, by timely reprojecting a rendered frame to the new camera view before display. As shown in Figure 5 which uses *Image-based Rendering (IBR)* [39, 45] as an example, for any pixel (x, y) on the 2D user view plane, its coordinate in the 3D virtual world can be computed as $W = M_{proj}^{-1} \cdot (x, y, z)$, where z is the depth value of (x, y) and M_{proj} indicates the current camera projection. Then, when the camera projection changes to M'_{proj}, IBR produces the new user view by reprojecting W onto the 2D plane as $(x', y', z') = M'_{proj} \cdot W$ for every pixel, without re-rendering these pixels at new locations. The distance of such image warping, then, is defined as the spatial distance between (x, y, z) and (x', y', z') in the VR space. Since such reprojection continuously tracks the movement of user camera location, it is able to warp the reference frame's pixels to the exact positions in the new user view. The per-pixel difference calculated from such a warped image, hence, contains a minimum amount of pixel redundancy.

To investigate the pixel redundancy in image warping, we first use IBR to warp reference frames in the above three VR games over

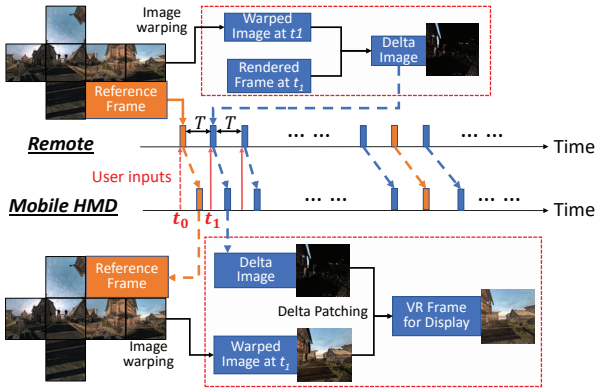
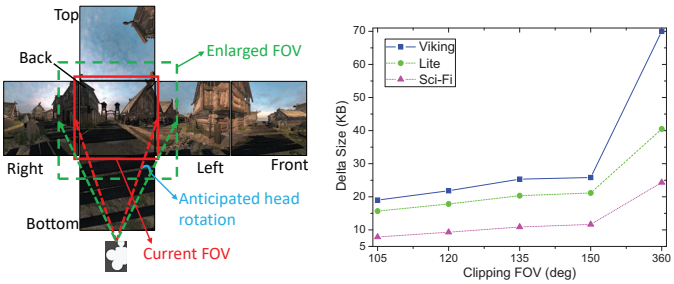


Figure 7: Overall design of DeltaVR



(a) Delta image clipping (b) Delta image size with clipped FOVs

Figure 8: Minimizing the delta image size

different distances, and then apply H.264 to encode the differences between the warped frames and reference frames. The amount of VR pixel redundancy, hence, can be measured by the percentage of zero values in the H.264 encoding output. Experiment results in Figure 8 show that image warping allows removing >85% of pixel redundancy in VR frames, even if the warping distance grows to 1.0. Especially with higher H.264 compression ratio, image warping helps remove 50% more pixel redundancy.

3 DELTAVR DESIGN

Based on the preliminary experiment results in Section 2.3, the design of DeltaVR aims to dramatically expand the scope of VR image warping from two consecutive frames (warping distance <0.1) to the entire time space between every two reference frames (warping distance up to 1.0), so as to fully utilize image warping's advantage on tracking the VR camera view for maximum pixel reuse over all VR frames. Figure 7 illustrates how DeltaVR works: it periodically renders a panoramic VR frame from a remote computing facility (e.g., a nearby PC workstation or the remote cloud) according to the user inputs from the mobile HMD, and uses it as the reference frame to synthesize delta images for other VR frames in the future. Every time when a new VR frame is needed[2] at time t_1, DeltaVR remotely renders this frame in full, and then warps the most recent reference frame from its original camera view at t_0 to the current user view at t_1. As a result, the delta image of this frame at t_1 is remotely synthesized as the difference between the originally rendered frame and the warped image from the reference frame.

At the mobile HMD, DeltaVR warps the received reference frame in the same way to the user view at t_1. When the corresponding delta image is received, it applies the delta image to patch the visual artifacts being produced by image warping, to restore the full VR frame for display with the minimum amount of pixel redundancy or VR image quality loss. In this way, DeltaVR avoids the expensive rendering of any VR frame pixel at the mobile HMD. It is hence able to guarantee the VR frame rate regardless of the VR scene complexity or dynamics, as long as the mobile HMD can finish delta image transmission and reference image warping within T.

[2]The time interval (T) between two consecutive VR frames is determined by the frame rate, which should be at least 60 FPS for satisfiable VR performance. The value of T, hence, could be at most 16.7 ms.

3.1 Minimizing the Delta Image Size

Due to the larger scope of image warping, the size of a delta image in DeltaVR may grow when the user character keeps moving in the virtual world and results in longer warping distance. To ensure timely transmission of delta images, DeltaVR further minimizes the size of delta images by exploiting the limited FOV of today's mobile HMDs, which is usually smaller than 120° [3]. As a result, instead of synthesizing and transmitting a delta image over the 360° panoramic view, DeltaVR transmits to the mobile HMD with a clipped delta image corresponding to the current user camera orientation and FOV, which are reported from the mobile HMD every time when a new VR frame is needed.

The major challenge of such delta image clipping, however, is that the user view may change during the process of delta image synthesis due to user head rotation, and such change cannot be known by the cloud in advance. Our solution to this challenge, as shown in Figure 8(a), is to further enlarge the FOV of image clipping by X° in both sides, to cover the possible change of user view. In practice, since each delta image is promptly transmitted to the mobile HMD within a very short amount of time, the possible change of user view during this short time period is very limited. For example, even with the most vigorous user head rotation where the angular velocity reaches to 780° per sec [24], the value of X is merely 17.5 for a 11ms latency of delta image transmission.

As shown in Figure 8(b), such clipping further reduces the size of delta images by up to 65%, when being applied to the three open-sourced VR games that we described in Section 2. In particular, such size could be effectively controlled within 25 KB when the warping distance is smaller than 1.0 and the user FOV is smaller than 150°, which could be considered as the optimal FOV that well balances between VR frame rate and user experience in practice. Such a small delta image can be transmitted within 8 ms even through the low-speed WiFi with 24Mbps, and also minimizes the possible VR display lag when the user character keeps moving during the mean time of delta image transmission. Further increasing the warping distance, on the other hand, may result in transmitting another new reference frame, and will be discussed in Section 4.

3.2 Minimizing the Computation Overhead of Image Warping

Such image clipping can be similarly applied to a panoramic reference frame to minimize the computational overhead of image

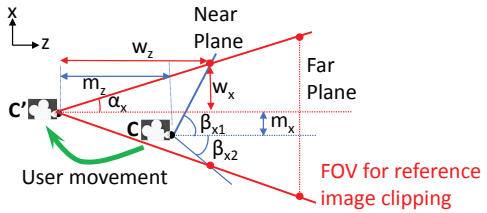


Figure 9: FOV for reference image clipping

warping from the reference frame to the current user view[3]. For example, our experimental studies show that it takes up to 33.5 ms for a LG G5 smartphone to warp a reference frame with 1024x1024 resolution over the warping distance of 0.5. Clipping the reference frame to a FOV of 150°, in this case, could effectively reduce such computation time to <8 ms.

The major challenge of clipping the reference frame, however, is how to decide the portion to be clipped without impairing VR user experience or image quality. Since the user character in the virtual world may move during the warping process, if we clip a reference frame only based on the current FOV and camera orientation at the mobile HMD, certain visible regions in the new target view may be missed. Instead, DeltaVR exploits the property of VR perspective projection, and determines the right FOV of clipping a reference frame according to the real-time user movement in VR world.

According to perspective projection, the 2D user view plane in VR applications is rendered with the VR objects between the near plane and the far plane in the truncated pyramid frustum. Hence, DeltaVR further reduces the overhead of VR frame transmission by only transmitting part of reference images within the reduced user FOV. As shown in Figure 9, when the user character moves from the current camera position C to a new position C′ during the warping process (only showing the X-Z plane for simplicity), the FOV for reference image clipping at C′ is decided based on the displacement between C and C′, and should make sure that the frustum at C (defined as $\beta_{x1} + \beta_{x2}$) is fully covered. According to the Pythagorean theorem [37], β_{x1} and β_{x2} can be determined as

$$\begin{cases} \beta_{x1} = \arctan \frac{w_x + m_x}{w_z - m_z}, \\ \beta_{x2} = \arctan \frac{w_x - m_x}{w_z - m_z}, \end{cases}$$

where w_z is the distance of near plane, $w_x = w_z * \tan \alpha_x$, and α_x is half of the FOV at C′. Similarly, the FOV for reference image clipping in the Y-Z plane can be computed in the same way.

4 ADAPTATION TO VR DYNAMICS

Based on the aforementioned design, both the VR image quality and the amount of VR frame data being transmitted are determined by the number of reference frames being used. More specifically, a new reference frame will be generated and transmitted to the mobile HMD, when the VR image quality is noticeably degraded due to the long warping distance and the subsequent visual artifacts on VR objects. The number of reference frames transmitted by DeltaVR, hence, is closely related to the level of VR dynamics. Higher VR dynamics (e.g., rapid object movement or shape change) lead to a smaller amount of pixel redundancy across VR frames, and hence require more reference frames to be transmitted.

[3]According to [26], the computational complexity of image warping is proportional to the size and resolution of the reference image.

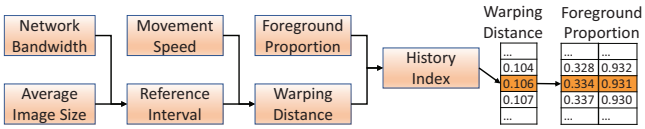


Figure 10: Deriving upper bound of SSIM threshold

To adapt to such VR dynamics, DeltaVR flexibly adjusts the interval of transmitting new reference frames at run-time, and only transmits a new reference frame if the VR image quality drops lower than a threshold, which is adaptively calculated by the mobile HMD at run-time according to the current situations of both user movement and VR scenic changes. Without loss of generality, we use the structural similarity (SSIM) metric [53] to measure the VR image quality. According to [17], SSIM is designed to model the human eye's perception to 3D images, and a SSIM score higher than 0.9 indicates good quality of VR images.

4.1 SSIM Bounds

DeltaVR decides the threshold of transmitting reference frames between a lower bound and a upper bound of the SSIM score of VR image quality. These bounds are computed before each VR frame is being rendered at the cloud, to decide whether this frame will be transmitted to the mobile HMD as a reference frame. While the lower bound can be easily fixed as 0.9 to ensure satisfiable image quality [17], the upper bound is expected to prevent too frequent transmissions that exceed the wireless network capacity and hence reduce the VR frame rate. As a result, it is jointly determined by the warping distance and the amount of foreground pixels at a user view in a future timepoint T_{next}, when the wireless network is expected to be able to afford transmitting another new reference frame. Note that, since VR dynamics are mainly reflected by the fluctuations of foreground VR objects, DeltaVR only takes the foreground pixels at a user view into account when deciding this upper bound.

The process of computing such a upper bound is illustrated in Figure 10. Based on the history system profiles and information about current user movement, DeltaVR first predicts T_{next}, which is measured as the number of VR frame intervals. To do this, the cloud records the elapsed time of each wireless transmission to estimate the bandwidth of wireless link (B). In addition, DeltaVR also monitors the average sizes of the delta and reference images (S_{delta} and $S_{reference}$). Then, T_{next} is computed as follows,

$$T_{next} = (B - S_{reference})/(S_{delta}) - T_{sent}, \quad (1)$$

where T_{sent} represents the number of delta frames that have already been sent since the last transmission of reference image. Afterwards, the target camera position and warping distance at T_{next} can be predicted based on the current speed of user movement.

Our prediction about the amount of foreground pixels at T_{next}, on the other hand, builds on the fact that the animation for motion transition usually lasts for a lengthy period (up to 2 seconds) [52], and hence smooth foreground VR object changes can be assumed. As a result, we utilize the amount of foreground pixels in the current frame to derive the SSIM upper bound. More specifically, we render the foreground objects in the current frame alone, resulting in a pair of color and depth buffers as shown in Figure 11. Then, the number of foreground pixels equals to the number of non-white

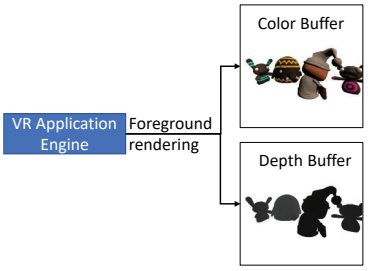


Figure 11: Foreground rendering

pixels in the depth buffer, because a white pixel indicates that no foreground object appears in this location.

4.2 Look-up Table

Intuitively, DeltaVR computes the SSIM upper bound by warping the current VR frame to the predicted camera position, but may incur large amounts of additional computation overhead. Instead, we observe that the VR scene complexity is relatively stable in each VR application, and the SSIM score is hence constant with a given warping distance and amount of foreground pixels. Therefore, DeltaVR records the SSIM scores of the past VR frames into a two-layer look-up table (LUT) that is indexed by the corresponding warping distance and the amount of foreground pixels, and always searches the LUT before warping the current VR frame.

As shown in Figure 10, every time after a new warped frame is generated at the cloud, a new entry in the LUT is correspondingly created if the difference between its warping distance and that of any existing entry in the LUT is larger than 0.001. On the other hand, given a warping distance or foreground proportion to be searched, the LUT returns an entry with the next larger warping distance or foreground proportion, to give a more conservative SSIM score as the upper bound. For example in figure 10, the warping distance of 0.105 and foreground proportion of 33.2% result a SSIM of 0.931 from the warping distance of 0.106 and proportion of 33.4%.

4.3 Determining the Adaptation Threshold

DeltaVR linearly adjusts the SSIM threshold between the lower and upper bounds calculated above, with a coefficient α according to the following formula:

$$SSIM_{threshold} = (1-\alpha)\cdot SSIM_{upper} + \alpha \cdot SSIM_{lower}$$

In practice, the coefficient α could be adjusted either automatically or manually. First, DeltaVR could monitor the user interaction events in VR applications and tune the coefficient correspondingly: when the user is actively interacting with the surrounding environment and results in higher sensitivity to the foreground scenic changes [42], a higher value of α could be adopted. On the other hand, α can also be flexibly set by the VR app developers to balance between the VR image quality and frame rate.

5 IMPLEMENTATION

We implement DeltaVR over Google VR Unity SDK v1.20 and Unity VR application engine v5.5.1, with minimum modification on either the Google VR SDK itself or the VR application binaries. It consists approximately 3,000 lines of C++ code as a plugin to the Unity engine, and 850 lines of C# code as a Unity engine script.

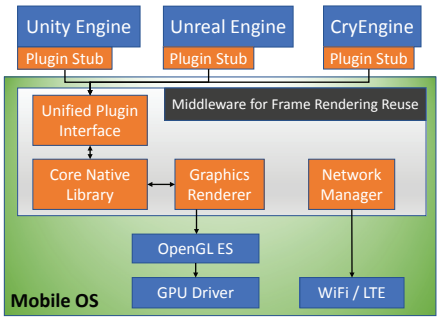


Figure 12: DeltaVR as a mobile middleware

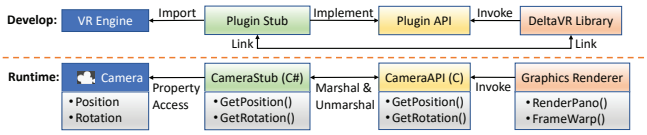


Figure 13: Unified interaction with game engine

5.1 Remote Operations

DeltaVR runs a clone copy of each VR application at the remote computing facility, and renders VR frames according to the user inputs such as the controller operations received from the mobile HMD as system events. To retrieve the rendered full VR frames from the application binary, we exploit the hook of the application engine and attach a post-processing script to VR camera. This script transforms the depth buffer into a greyscale image, and then reads the pixels of color and depth images into main memory.

On the other hand, to remotely render the panoramic reference frames, we create a specialized camera in the VR application binary, which utilizes the VR application engine's API to render the scene as a cubemap. Specifically, the camera renders the scene onto the sides of a cube with six square textures, which represent the view along the directions of the world axes (up, down, left, right, forward and back). Each face of the cubemap has a FOV of 90° and a resolution of 1024x1024 so as to capture a 4K panoramic view of the scene.

5.2 Mobile OS Integration

The major challenge of DeltaVR implementation at the mobile HMD is how to efficiently support different VR applications in a generic manner. First, VR applications are heterogeneous in their shading languages and scripting APIs being used. For example, the Unity engine uses either JavaScript or C# as the script language, but the Unreal engine[4] only supports C++. Supporting pixel reuse within the VR application binary, hence, leads to repetitive efforts of re-programming. Second, operations of pixel reuse, if being done in the user space, would be less effective due to frequent interaction with the system hardware.

To address these challenges and retain generality, we integrate DeltaVR into the OS kernel of the mobile HMD, and implement it as a middleware between VR applications and OS drivers. As shown in Figure 12, the core of DeltaVR is implemented as an OS library in native language to regulate the main DeltaVR functionality. The core library then interacts with the graphics renderer, which

[4]https://www.unrealengine.com/

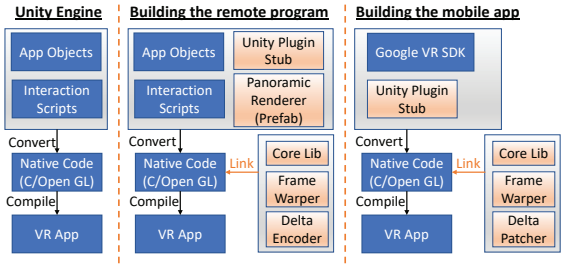


Figure 14: Making VR apps with DeltaVR

manages frame buffers and invokes APIs directly from OpenGL ES for delta patching and image warping. Since the OpenGL provides unified APIs for 3D graphics rendering, the pixels in VR frames are generically reused without involving the engine-specific shading languages such as the Microsoft's HLSL [51] and Nvidia's Cg [38].

On the other hand, the core library needs to interact with VR app binaries to retrieve the necessary metadata for pixel reuse, such as the current camera position, orientation and FOV. An intuitive solution is to invoke engine-specific APIs directly from the core library, but lacks generality.

Instead, we introduce a middle layer with a suite of unified plugin APIs for data exchange as shown in Figure 13. In particular, a plugin stub is implemented with engine-specific scripts to fulfill behaviors of the predefined APIs. Such stub is dynamically linked with the core library during development, so that any invocation to the plugin API will be directed to the plugin stub at runtime. For example in Unity, to warp the reference frame to the target view at runtime, the graphics renderer in the core library needs to find out the current camera position and hence will invoke the *GetPosition()* function in the plugin *CameraAPI*, which is written in native C. This function marshals the request to the managed format in C#[5] and triggers the engine-specific script *CameraStub* to access the position property of the camera object. Afterwards, the position values of the engine camera is marshaled to the native format and returned to be processed by the graphics renderer.

6 MAKING VR APPS WITH DELTAVR

The generic design and implementation of DeltaVR significantly reduce the burden of VR application development, with the Unity engine as the target VR software platform. Typically, as shown in Figure 14, the Unity engine converts the application-specific 3D objects and scripts of user interaction into native codes that are further compiled as executable binaries, so as to render the VR scenes at run-time. Such procedure enables the application developer to easily extend the application's functionality from a basic prototype by simply linking the new native libraries into the existing program binaries.

Our work exports the components implemented in Section 5.2 as easy-to-use modules, based on which VR applications can be built for both the remote computing facility and the mobile HMD. As shown in Figure 14, the developers simply need to import the modules provided by DeltaVR by copying the libraries to the application folder and create special prefab[6] instances in the Unity engine, and these prefabs will then be dynamically linked into the final executable at compile time. Specifically, besides the core library, the modules of image warping and delta encoding/decoding should also be included into the graphics renderer at both sides, and a prefab of panoramic renderer should be created at the remote computing facility to render panoramic reference frames.

[5]http://msdn.microsoft.com/en-us/library/ms235282.aspx
[6]A game object acts as a template with predefined scripts and properties.

7 PERFORMANCE EVALUATION

In this section, we evaluate the performance of DeltaVR, which is measured by the VR frame rate, image quality and motion-to-photon latency. Our experiment results show that DeltaVR can maximize the VR performance over highly dynamic VR scenarios with complicated scenes and intensive user movements. We also demonstrate the effectiveness of DeltaVR in resource-constrained scenarios with limited wireless network bandwidth, wireless link reliability and device energy budget.

Table 1: Statistics of VR scene complexity

Game	Draw Calls	Triangles (K)	Vertices (K)
Viking	400	2,400	1,600
Lite	212	65.7	52.4
Sci-Fi	227	32.7	36.7

7.1 Experiment Setup

In our experiments, we use a LG G5 smartphone with Android v6.0.1 as the mobile HMD, and a Dell OptiPlex 9010 Desktop PC with an Intel i5-3475s@2.9GHz CPU, Radeon HD 7470 GPU and 8GB RAM as the remote computing facility. We use a Google cardboard as the experimental VR headset with a FOV of 90°. The mobile HMD is connected to the PC via campus WiFi, which has an average throughput of 40 Mbps and transmission latency of 3.5 ms. A Monsoon power monitor[7] is used to measure the energy consumption of the mobile HMD, and each experiment is conducted multiple times for statistical convergence.

We evaluate the performance of DeltaVR over three open-sourced VR games listed in Section 2.3, which are configured to operate with high VR dynamics. Unless explicitly stated, each VR scene contains 4 animated foreground objects moving at 1m/s towards a random direction, and the user character moves at a fixed speed in VR world to constantly change the camera view. The user frequently interacts with foreground objects through hand controllers, and these objects will adaptively change their appearances upon user interaction based on game contents.

As listed in Table 1, the three VR games also present different levels of VR scene complexity. The experiment results over them, hence, are representative and can be generally applied to other VR applications with similar levels of VR complexity.

By default, DeltaVR transmits a new reference image to the mobile HMD every 60 VR frames, resulting in a maximum warping distance of 1 in the virtual world. Each panoramic delta image, before being transmitted, is clipped with a FOV of 150°, which allows a 30° head rotation with Google cardboard. We compare DeltaVR with the following existing VR schemes:

- **Local rendering:** VR applications are solely running on the mobile HMD.

[7]https://www.msoon.com/LabEquipment/PowerMonitor/

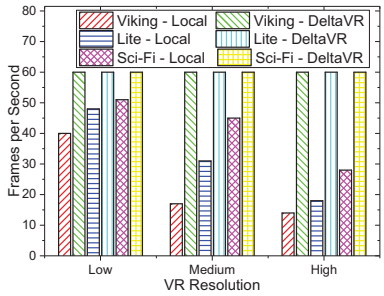


Figure 15: Frame rate with different VR resolutions

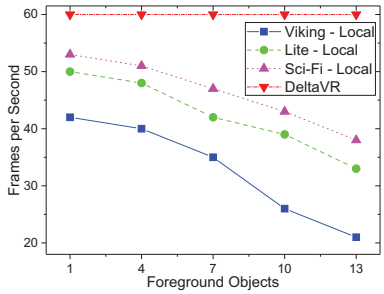


Figure 16: Frame rate with different levels of VR scene complexity

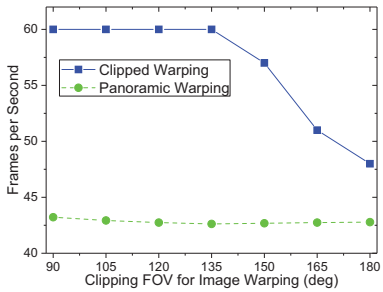


Figure 17: Frame rate with different FOV for image warping

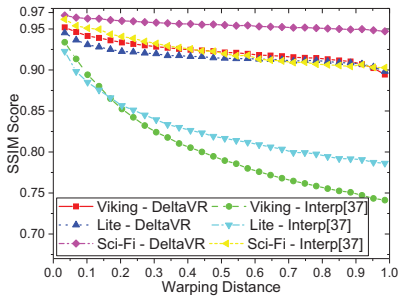


Figure 18: Efficiency of delta patching

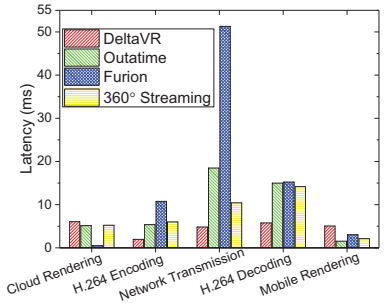


Figure 19: VR system latency

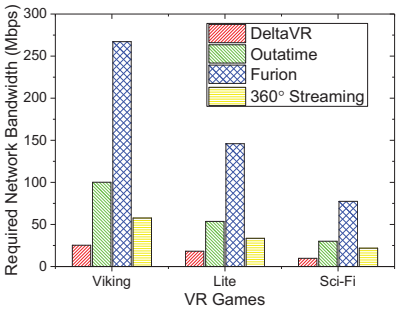


Figure 20: Bandwidth consumption

- **Outatime**: Every VR frame is rendered by the cloud based on the predicted user interactions and then transmitted in full to the mobile HMD [29].
- **Furion**: A VR frame is collaboratively rendered at both the remote computing facility and mobile HMD. Panoramic VR backgrounds are rendered at the cloud for all possible user movements. Foreground VR objects are locally rendered at the mobile HMD [28].
- **360-degree adaptive streaming**: Each VR frame is divided into tiles that are adaptively compressed according to the predicted user FOV [34, 55], where the image quality is high in the center of FOV and decreases toward the edge.

7.2 Improvement of VR Performance

Our experiment results show that DeltaVR always achieves the required 60 FPS with different levels of VR resolution and scene complexity, while providing high image quality with SSIM > 0.92. It also minimizes the motion-to-photon latency within 16ms to ensure smooth VR experience and avoids any possible motion sickness.

7.2.1 Frame Rate. As shown in Figure 15, the frame rate provided by DeltaVR is constantly 60 FPS in all VR resolutions, and greatly outperforms local VR frame rendering whose performance significantly drops to < 15 FPS under high resolution. Similarly, Figure 16 shows that the frame rate in DeltaVR remains constant even when the VR scene becomes highly complicated with 13 foreground objects. Note that in our experiments, the number of foreground VR objects is controlled to keep appropriate sizes of VR objects and avoid motion sickness due to too many small 3D objects. With more foreground objects, VR performance gradually drops as more computations are involved.

In contrast, locally rendering VR frame at the mobile HMD will experience significant frame rate loss when the VR image quality or the number of foreground object increases. Particularly, the maximum FPS that DeltaVR can achieve in our experiment is limited by the screen refreshing rate at the mobile HMD that is capped at 60Hz, and could hence be further improved on future mobile devices which supports higher screen refreshing rates (e.g., 90Hz).

One reason for such improved VR performance, as described in Section 3.1, is the clipping process over panoramic reference frames that reduces the computation overhead of image warping at the mobile HMD. As shown in Figure 17, compared with panoramic image warping that reduces the VR frame rate down to $\sim$ 42 FPS, DeltaVR improves the frame rate by more than 40% as long as the clipping FOV does not exceed 135°, hence allowing a maximum warping distance of 0.98 without any VR performance degradation.

7.2.2 Image Quality. We evaluate the VR image quality provided by DeltaVR using the SSIM metric [53], which quantifies the image quality degradation in DeltaVR from the pristine high-quality image rendered by the remote computing facility. The results in Table 2 show that DeltaVR ensures high image quality (> 0.9) in all the VR applications, and significantly outperforms that of local frame rendering. Such improvement on image quality allows many advanced graphics options such as shadow casting and anti-aliasing at the mobile HMD, and greatly enhances the user experience.

Rendering Scheme	Viking	Lite	Sci-Fi
Local Frame Rendering	0.8133	0.8766	0.8832
DeltaVR w/ Stationary User	0.9569	0.9599	0.9681
DeltaVR w/ Moving User	0.9241	0.9210	0.9557

Table 2: VR image quality (SSIM)

Besides, we also evaluate the efficiency of delta patching in DeltaVR by comparing with traditional image warping that interpolates pixels in disoccluded regions [39]. Figure 18 shows that

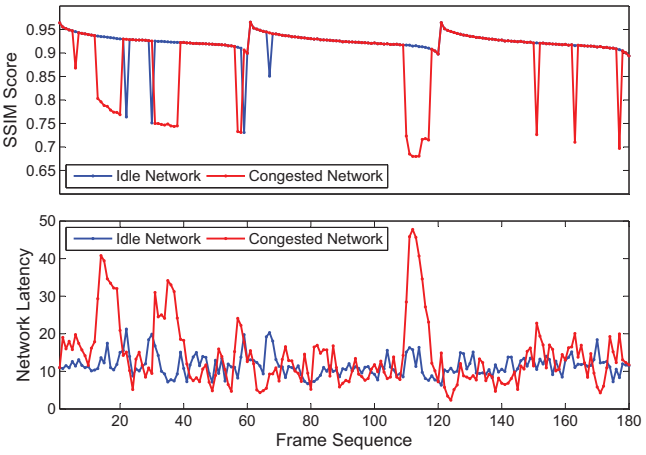


Figure 21: Impact of network delay fluctuation

DeltaVR experiences much less image quality degradation when the warping distance increases, and retains SSIM> 0.9 in all cases. In comparison, the VR image quality in traditional image warping quickly drops when the warping distance is larger than 0.2, because of the disoccluded areas in VR frames with higher VR dynamics.

7.2.3 Latency. Motion-to-photon latency is critical in VR to ensure user experience and avoid motion sickness. The breakdown of such latency over the Viking Village game, as shown in Figure 19, is averaged over all VR frames and shows that DeltaVR controls the end-to-end latency to be lower than 20ms. The end-to-end VR latency is mainly determined by computations of VR frame encoding/decoding and wireless transmission of VR frame data. Compared with existing schemes, DeltaVR achieves much lower latency in these computations and data transmissions, leading to significant reduction in the overall VR latency. In particular, the delay of transmitting a delta image in DeltaVR is about 4.8 ms, which is less than 10% of that of Furion [28]. Similarly, the minimized sizes of delta images make the mobile HMD possible to completely decode a delta image and prepare a VR frame ready for display within 10 ms. In contrast, existing 360° streaming techniques spend much higher computation time for image encoding and decoding, due to frequent misprediction of user FOV and hence much less adaptability to the corresponding VR dynamics. Such failure of adaptation also results in 50% more data transmission in 360° streaming.

On the other hand, although DeltaVR spends more time for VR frame rendering at both the cloud and mobile HMDs because of the extra overhead of image warping and delta operations, such extra overhead has little impact on the end-to-end VR latency.

7.3 Impact of Wireless Network Condition

The condition of wireless connection between the remote computing facility and the mobile HMD is another critical factor to the VR performance. Existing schemes such as Furion [28] requires at least 100 Mbps of wireless bandwidth to provide satisfiable VR performance, but such bandwidth may not be available in many practical application scenarios with commodity wireless networks (e.g., 802.11g networks with a maximum bandwidth of 54 Mbps) or heavy wireless channel contentions. In contrast, Figure 20 shows that DeltaVR requires at most 25 Mbps of network bandwidth, which can be easily satisfied by any existing WiFi or even today's cellular networks (e.g., LTE networks with download speeds up to 20 Mbps). On the other hand, other schemes on 360° streaming reduce 43% of frame data transmissions by applying high compression ratio and low resolution toward the edge of VR frames, but still transmits 56% more frame data than DeltaVR does. Such data compression and resolution reduction, as shown in Section 8, will also impair the VR image quality provided to the user.

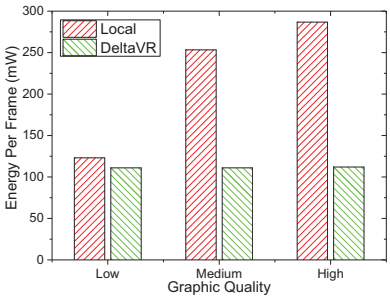


Figure 22: Energy consumption

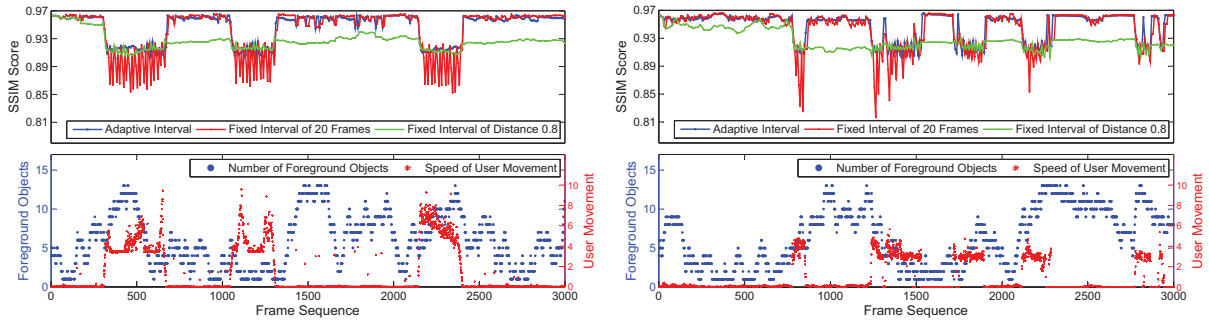
The wireless network condition in practice, however, could fluctuate due to the unreliable wireless links and produce additional transmission delay. To evaluate the impact of such network fluctuation, we test the Viking Village game with a recorded VR frame trace in both idle and congested wireless networks. The experiment results over 180 VR frames are shown in Figure 21, from which we can see that the VR image quality only experiences significant degradation when the link delay increases to >40 ms and such long delay lasts for >10 frames. In this case, the mobile HMD misses a delta image and has to project the previous VR frame to the new camera position. Meanwhile, minor delay increase or temporary delay jitter has negligible impact on the image quality in DeltaVR.

7.4 Energy Efficiency

We evaluate the energy efficiency of DeltaVR over the Viking Village game, by measuring the average amount of power consumed by the mobile HMD for rendering and displaying each VR frame. From Figure 22, we can see that DeltaVR reduces the energy consumption of VR frame rendering by up to 60% with high VR image quality, when compared with local VR frame rendering at the mobile HMD. Besides, DeltaVR also maintains a constantly low level of energy consumption regardless of the image quality setting in VR applications, and is hence well applicable to a large variety of mobile devices with severe resource constraints.

8 REAL-WORLD EXPERIMENTATION

In this section, we evaluate the DeltaVR's capability of adapting to the heterogeneous VR dynamics produced in practice, over two mobile VR games available at Google Play that allow very intensive user actions and movements: *Dead Zombies Survival VR* [2] and *VR Fantasy* [8]. In order to collect real-world camera traces from these applications for such evaluation, we hacked into the dynamic link libraries of the Unity engine in the application APK file, and added logging code segments to record the current camera position after rendering each VR frame. At every frame, we randomly add a new foreground 3D object with a probability of 0.1 to simulate VR dynamics, and then allow a participating user to freely move towards the object.



(a) Dead Zombies Survival VR (b) VR Fantasy

Figure 23: VR image quality with real-world VR dynamics

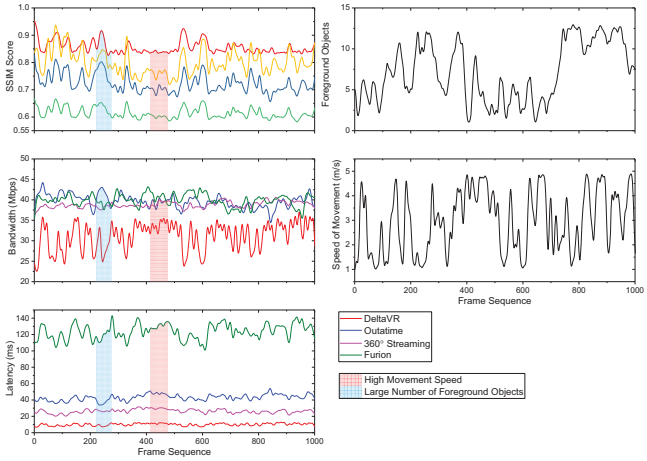


Figure 24: VR performance comparison

8.1 VR Image Quality

Experiment results over 3000 realistic VR frames are shown in Figure 23, from which we can see that DeltaVR effectively retains the VR image quality with SSIM >0.9 in most cases. It also improves the stability and robustness of such VR image quality against heterogeneous VR dynamics, including the varying number of foreground 3D objects and fluctuating level of user movement. Even in extreme cases when there are more than 10 foreground VR objects or the user moves faster than 6m/s, the user can experience noticeable image quality degradation with a SSIM between 0.82 and 0.85.

In addition, for the two VR games being tested, our adaptation approach requires only ~23 Mbps of network bandwidth and achieves an average SSIM score of 0.95. In contrast, when we adopt the fixed strategy of sending a new reference frame per 20 VR frames, the VR image quality will experience significant degradation when the VR dynamics increase. Sending a new reference frame whenever the warping distance reaches 0.8, on the other hand, greatly reduce the VR image quality because of the ignorance of users' interaction with the foreground VR objects. These fixed strategies also increase the network bandwidth required for VR by more than 20%.

8.2 VR Performance Comparison

We further compare the performance of DeltaVR with other existing VR schemes under different levels of VR dynamics, over the 1,000 frames collected from the VR Fantasy game with free user movement in the real game scene. According to experiment results shown in Figure 24, DeltaVR well adapts to the fluctuating levels of VR dynamics and is able to retain the SSIM metric higher than 0.85 at all times, even with the highest user movement speed or the largest number of foreground VR objects. In comparison, existing VR schemes experience serious image quality degradation due to VR dynamics, especially when the user moves fast in the virtual world. In particular, the SSIM metric of Furion could drop to 0.6, which significantly impairs VR user experience.

In addition, Figure 24 also shows that DeltaVR outperforms existing VR schemes in aspects of network bandwidth required and motion-to-photon latency being produced. In particular, DeltaVR only consumes >30 Mbps wireless bandwidth when high frame dynamics are presented in the VR game due to fast user movement. Otherwise, it only requires 25 Mbps network bandwidth to well adapt to heterogeneous VR dynamics. At the same time, our proposed adaptation schemes in Section 4 effectively constrain the motion-to-photon latency within 15 ms. Such latency is also able to well resist against fluctuations of VR dynamics and hence avoid VR motion sickness in all cases.

9 RELATED WORK

Mobile Offloading and Cloud Gaming: General-purpose mobile offloading reduces the local computational burden of mobile devices, by adaptively partitioning the computing tasks and offloads only the most appropriate portion to the cloud for remote execution [16, 23, 32, 49]. However, it is difficult to partition the process of rendering a VR frame, which is operated by GPU hardware. The amount of frame data sent to the mobile HMD, hence, remains unchanged and unaffordable by any existing wireless network connecting mobile HMDs to the cloud.

Our proposed design of DeltaVR is related to prior work on cloud gaming [21, 46]. Existing commercial systems such as PlayStation Now and NVidia Shield, consider frontend mobile devices as a thin client, to which the game's output is streamed as compressed video. However, these designs cannot scale to mobile VR, because its requirements of high resolution and low response latency make it impossible to stream game scenes at real-time. Other schemes enable multi-Gbps wireless communication to VR headsets via mmWave wireless technology [9, 10] or propose new technologies to improve the wireless network throughput under fluctuating channel conditions [35, 36], but rely on specialized hardware support and

line-of-sight connectivity. These techniques are orthogonal to the major focus of DeltaVR, which aims to provide high VR performance using commodity wireless networks with low bandwidth.

Collaborative Rendering: Recently, researchers advocate the idea of collaborative rendering, which splits the computing workload of rendering individual frames between the cloud and local mobile devices. Flashback [13] pre-renders all the VR frames offline and caches the rendered frames at the HMDs' local storage, so as to alleviate the run-time computation burden. However, the system performance deteriorates quickly with the number of dynamic VR objects and the unavoidable cache miss. It also consumes a huge amount of storage space at mobile devices (e.g., 50GB data for each VR application). Kahawai [17] exploits the cloud GPU to render high quality images that enhance the visual quality of the locally rendered images, but still leaves heavy rendering workloads on the mobile. Outatime [29] and Furion [28] reduce the amount of VR frame data being transmitted by speculating future user movement for prefetching, but may suffer from misprediction with high VR dynamics. Instead, DeltaVR does not involve any prediction of VR user behavior or pre-fetching of VR frames, and is hence resistant against sporadic VR application events or user behaviors.

Graphic Processing: Image-based rendering [39] is widely used in today's VR applications, but incurs fast degradation of image quality with large warping distance due to the view disocclusion. Asynchronous TimeWarp technique [1] in mobile VR compensates and displays the previous frame with the current head rotation when the mobile fails to render on time, but leads to flickering edges with vigorous head motions. Post-processing techniques [50] interpolate or extrapolate the disoccluded view, but lead to blurry regions. In contrast, DeltaVR captures all disoccluded views in advance as the delta image, and hence guarantees high VR image quality regardless of the heterogeneous dynamics in VR applications.

Recent schemes of 360-degree video streaming [47, 48, 55] adapt the resolution and compression ratio of panoramic frames to the user view, so as to reduce the amount of image frame data being transmitted. These techniques, however, are limited to pre-known and fixed video contents. When being applied to interactive VR applications with volatile and unpredictable dynamics, they may suffer serious image quality degradation due to FOV prediction errors. In contrast, DeltaVR adaptively avoids transmitting VR frame pixels outside the user's FOV without using any prediction, and hence well adapts to VR dynamics.

10 DISCUSSIONS

10.1 Supporting Next-generation VR Systems

The VR industry is rapidly evolving towards wider FOV and higher pixel density. For example, the FOV and resolution of Pimax 8K [5] reach to 200° and 3840x2160 for each eye. Such change of image resolution quadratically increases the image size, which aggravates the insufficiency of wireless network bandwidth. However, DeltaVR is less impacted because the redundancy between VR frames persist, resulting in constantly small delta images. As described in Section 3, DeltaVR can also be naturally integrated with emerging techniques of eye gaze tracking [31, 41] towards more efficient foveated rendering for overhead reduction. On the other hand, the mobile VR performance will be impaired by the computational overhead of image warping, which is proportional to the image resolution. To address this issue, we plan to adaptively adjust the density of the mesh grid in mobile IBR, so as to reduce the computational overhead of image warping.

10.2 Humans' Sensitivity to VR Latency

Our DeltaVR design has been proved to effectively reduce the VR latency that is subjectively experienced by human beings. Such VR latency can be divided into two categories. First, the motion-to-photon latency represents the elapsed time for the user's head motions to be reflected on the VR screen. As noted in [14], humans are more sensitive to such latency, which needs to be less than 20 ms in VR systems. DeltaVR reduces such latency by sampling the user pose right before mobile rendering, to exclude the influence of networking and decoding delay. In contrary, the latency of users' body interactions to the virtual environment is much less important and users can tolerate up to 50 ms of such latency. The high performance of DeltaVR, as shown in Figure 19, allows sufficient time for cloud processing and ensures robustness to large system variance without compromising the user experience.

10.3 Multi-user Support

Multiple VR users may execute VR applications at the same edge cloud server with finite resources, which may exceed the cloud capacity [33]. Our future work plans to share GPU between individual VR users so as to reuse the results of cloud rendering and hence reduce the computational overhead in the cloud. It is observed that temporal and spatial locality widely exists in VR games [15] and hence the pixels in the rendered images across multiple users should be redundant as well. By reusing the pixels from other users [11, 57], the cloud GPU can avoid the repetitive computations and reduce resource utilization.

10.4 Improving the Accuracy of VR Prediction

Many existing mobile VR schemes predict VR user behaviors and prefetch the corresponding VR frames to the mobile HMD, so as to compensate the excessive latency of transmitting VR frame data at run-time [28, 29]. However, the unexpected VR dynamics or sporadic VR application events could easily compromise the accuracy of such prediction and lead to wrong VR frame data being prefetched. These frame data, if being used by VR applications, will result in significant degradation of VR image quality.

DeltaVR, on the other hand, could help improve the accuracy of such VR behavior prediction, by minimizing the latency of transmitting VR frame data. First, such minimum transmission latency reduces the difficulty of prediction, because the prefetched frame data will be delivered and used sooner in the future and hence has lower chance of VR camera view misplacement. Second, the bias in camera position could also be reduced when misprediction happens, which incurs unnoticeable visual difference to the correct user view.

11 CONCLUSION

In this paper, we present DeltaVR, which achieves high-performance mobile VR over heterogeneous VR dynamics, by adaptively reusing the redundant VR pixels across consecutive VR frames. DeltaVR utilizes the remote computing facility to determine the pixel redundancy between frames and transmits only the distinct portions

to the mobile HMD, so as to fundamentally reduce the amount of VR frame data being transmitted. Based on the implementation and evaluation over Android OS and Unity engine, we demonstrate that DeltaVR maximizes the mobile VR performance with 95% less amount of wireless data transmission.

ACKNOWLEDGMENTS

We sincerely thank the anonymous shepherd and reviewers for their valuable comments and feedback. This work was supported in part by the National Science Foundation (NSF) under grant number CNS-1826884, CNS-1812399 and CNS-1812407.

REFERENCES

[1] Asynchronous TimeWarp. https://developer.oculus.com/documentation/mobilesdk/latest/concepts/mobile-timewarp-overview/.
[2] Dead zombies survival VR. play.google.com/store/apps/details?id=com.dead.zombies.survival.vr.
[3] FOV of VR headsets. https://virtualrealitytimes.com/2017/03/06/chart-fov-field-of-view-vr-headsets/.
[4] Lite. https://assetstore.unity.com/packages/3d/environments/fantasy/make-your-fantasy-game-lite-8312.
[5] Pimax 8K VR. www.pimaxvr.com/8k/.
[6] Sci-Fi Modular Environment. https://assetstore.unity.com/packages/3d/environments/sci-fi/sci-fi-modular-environment-3426.
[7] Viking Village. https://assetstore.unity.com/packages/essentials/tutorial-projects/viking-village-29140.
[8] VR fantasy. play.google.com/store/apps/details?id=com.Chibig.VRFantasy.
[9] O. Abari, D. Bharadia, A. Duffield, and D. Katabi. Cutting the cord in virtual reality. In *Proceedings of ACM HotNets*, 2016.
[10] O. Abari, D. Bharadia, A. Duffield, and D. Katabi. Enabling high-quality untethered virtual reality. In *Proceedings of USENIX NSDI*, pages 531–544, 2017.
[11] J.-M. Arnau, J.-M. Parcerisa, and P. Xekalakis. Eliminating redundant fragment shader executions on a mobile GPU via hardware memoization. In *Proceedings of ACM/IEEE ISCA*, 2014.
[12] V. Bhaskaran and K. Konstantinides. *Image and video compression standards: algorithms and architectures*. Springer Science & Business Media, 1997.
[13] K. Boos, D. Chu, and E. Cuervo. Flashback: Immersive virtual reality on mobile devices via rendering memoization. In *Proceedings of ACM MobiSys*, 2016.
[14] J. Carmack. Latency mitigation strategies. *Twenty Milliseconds*, 2013.
[15] K.-T. Chen, P. Huang, and C.-L. Lei. Game traffic analysis: An MMORPG perspective. *Computer Networks*, 50(16):3002–3023, 2006.
[16] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: making smartphones last longer with code offload. In *Proceedings of ACM MobiSys*, 2010.
[17] E. Cuervo, A. Wolman, L. P. Cox, K. Lebeck, A. Razeen, S. Saroiu, and M. Musuvathi. Kahawai: High-quality mobile gaming using gpu offload. In *Proceedings of ACM MobiSys, year=2015,*.
[18] J. Dascal, M. Reid, W. W. IsHak, B. Spiegel, J. Recacho, B. Rosen, and I. Danovitch. Virtual reality and medical inpatients: A systematic review of randomized, controlled trials. *Innovations in clinical neuroscience*, 14(1-2):14, 2017.
[19] P. Dempsey. The teardown: Htc vive vr headset. *Engineering & Technology*, 11(7-8):80–81, 2016.
[20] P. R. Desai, P. N. Desai, K. D. Ajmera, and K. Mehta. A review paper on oculus rift-a virtual reality headset. *arXiv preprint arXiv:1408.1173*, 2014.
[21] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.
[22] M. S. Gordon, D. K. Hong, P. M. Chen, J. Flinn, S. Mahlke, and Z. M. Mao. Accelerating mobile applications through flip-flop replication. In *Proceedings of ACM MobiSys*, 2015.
[23] M. S. Gordon, D. A. Jamshidi, S. A. Mahlke, Z. M. Mao, and X. Chen. COMET: Code offload by migrating execution transparently. In *Proceedings of OSDI*, 2012.
[24] G. E. Grossman, R. J. Leigh, L. Abel, D. J. Lanska, and S. Thurston. Frequency and velocity of rotational head perturbations during locomotion. *Experimental brain research*, 70(3):470–476, 1988.
[25] T. Kämäräinen, M. Siekkinen, A. Ylä-Jääski, W. Zhang, and P. Hui. Dissecting the end-to-end latency of interactive mobile video applications. In *Proceedings of ACM HotMobile*, 2017.
[26] S. B. Kang. Survey of image-based rendering techniques. In *Videometrics VI*, volume 3641, pages 2–17. International Society for Optics and Photonics, 1998.
[27] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proceedings of INFOCOM*, 2012.
[28] 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 ACM MobiCom*, 2017.
[29] 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 mobile cloud gaming. In *Proceedings of ACM MobiSys*, 2015.
[30] K. Lee, D. Chu, E. Cuervo, A. Wolman, and J. Flinn. Delorean: using speculation to enable low-latency continuous interaction for mobile cloud gaming. In *Proceedings of ACM MobiSys*, 2014.
[31] T. Li, Q. Liu, and X. Zhou. Ultra-low power gaze tracking for virtual reality. In *Proceedings of ACM SenSys*, 2017.
[32] Y. Li and W. Gao. Interconnecting heterogeneous devices in the personal mobile cloud. In *Proceedings of IEEE INFOCOM*, 2017.
[33] Y. Li and W. Gao. MUVR: Supporting multi-user mobile virtual reality with resource constrained edge cloud. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018.
[34] X. Liu, Q. Xiao, V. Gopalakrishnan, B. Han, F. Qian, and M. Varvello. 360-degree innovations for panoramic video streaming. In *Proceedings of ACM HotNets*, 2017.
[35] H. Lu and W. Gao. Supporting real-time wireless traffic through a high-throughput side channel. In *Proceedings of ACM MobiHoc*, 2016.
[36] H. Lu and W. Gao. Continuous wireless link rates for internet of things. In *Proceedings of ACM/IEEE IPSN*, 2018.
[37] E. Maor. *The Pythagorean theorem: a 4,000-year history*. Princeton University Press, 2007.
[38] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard. Cg: A system for programming graphics hardware in a c-like language. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 896–907. ACM, 2003.
[39] W. R. Mark, L. McMillan, and G. Bishop. Post-rendering 3d warping. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 7–ff. ACM, 1997.
[40] D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the h. 264/avc video compression standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):620–636, 2003.
[41] A. Mayberry, P. Hu, B. Marlin, C. Salthouse, and D. Ganesan. ishadow: design of a wearable, real-time mobile gaze tracker. In *Proceedings of ACM MobiSys*, 2014.
[42] A. Ninassi, O. Le Meur, P. Le Callet, and D. Barba. Does where you gaze on an image affect your perception of quality? applying visual attention to image quality metric. In *Proceedings of IEEE ICIP*, 2007.
[43] T.-i. Ohta, K. Maenobu, and T. Sakai. Obtaining surface orientation from texels under perspective projection. In *Proceedings of IJCAI*, pages 746–751, 1981.
[44] H. Qiu, F. Ahmad, R. Govindan, M. Gruteser, F. Bai, and G. Kar. Augmented vehicular reality: Enabling extended vision for future vehicles. In *Proceedings of ACM HotMobile*, 2017.
[45] B. Reinert, J. Kopf, T. Ritschel, E. Cuervo, D. Chu, and H.-P. Seidel. Proxy-guided image-based rendering for mobile devices. In *Computer Graphics Forum*, volume 35, pages 353–362. Wiley Online Library, 2016.
[46] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui. Cloud gaming: architecture and performance. *IEEE network*, 27(4):16–21, 2013.
[47] R. Skupin, Y. Sanchez, C. Hellge, and T. Schierl. Tile based hevc video for head mounted displays. In *Proceedings of IEEE Int'l Symposium on Multimedia*, 2016.
[48] K. K. Sreedhar, A. Aminlou, M. M. Hannuksela, and M. Gabbouj. Viewport-adaptive encoding and streaming of 360-degree video for virtual reality applications. In *Proceedings of IEEE International Symposium on Multimedia*, 2016.
[49] L. Tong, Y. Li, and W. Gao. A hierarchical edge cloud architecture for mobile computing. In *Proceedings of IEEE INFOCOM*, 2016.
[50] C. Vázquez, W. J. Tam, and F. Speranza. Stereoscopic imaging: filling disoccluded areas in depth image-based rendering. In *Proc. SPIE*, 2006.
[51] I. Viola, A. Kanitsar, and M. E. Groller. *Hardware-based nonlinear filtering and segmentation using high-level shading languages*. IEEE, 2003.
[52] J. Wang and B. Bodenheimer. Computing the duration of motion transitions: an empirical approach. In *Proceedings of the ACM symposium on Computer animation*, pages 335–344, 2004.
[53] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.
[54] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.
[55] X. Xie and X. Zhang. POI360: Panoramic mobile video telephony over lte cellular networks. In *Proceedings of ACM CoNEXT*, 2017.
[56] R. Zhong, M. Wang, Z. Chen, L. Liu, Y. Liu, J. Zhang, L. Zhang, and T. Moscibroda. On building a programmable wireless high-quality virtual reality system using commodity hardware. In *Proceedings of the 8th Asia-Pacific Workshop on Systems*. ACM, 2017.
[57] H. Zhou, Y. Fu, and C. Liu. Supporting dynamic gpu computing result reuse in the cloud. In *HotCloud*, 2015.
[58] S. Zhu and K.-K. Ma. A new diamond search algorithm for fast block-matching motion estimation. *IEEE transactions on Image Processing*, 9(2):287–290, 2000.