MUVR: Supporting Multi-User Mobile Virtual Reality with Resource Constrained Edge Cloud

Yong Li
Department of Electrical Engineering and Computer Science
University of Tennessee at Knoxville
yli118@vols.utk.edu

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

***Abstract*—Virtual Reality (VR) fundamentally improves the user's experience when interacting with the virtual world, and could revolutionarily transform designs of many interactive systems. To provide VR from untethered mobile devices, a viable solution is to remotely render VR frames from the edge cloud, but encounters challenges from the limited computation and communication capacities of the edge cloud when serving multiple mobile VR users at the same time. In this paper, we envision the key reason of such challenges as the ignorance of redundancy across VR frames being rendered, and aim to fundamentally remove this performance constraint on highly dynamic VR applications by adaptively reusing the redundant VR frames being rendered for different VR users. Such redundancy in each frame is decided at run-time by the edge cloud, which is then able to memoize the previous results of VR frame rendering for future reuse by other users. After a VR frame is generated, the edge cloud further reuses its redundant pixels compared with other frames, and only transmits the distinct portion of this frame to mobile devices. We have implemented our design over Android OS and Unity VR application engine, and demonstrated that our design can efficiently reduce the computation burden at the edge cloud by more than 90%, and reduce more than 95% of the VR frame data being transmitted to mobile devices.**

I. Introduction

Virtual Reality (VR) stimulates users' immersive senses of the virtual world, and improves user experiences in many interactive scenarios such as gaming [15], [64], automobiles [48], healthcare [20], and education [44]. 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 computational capacity and battery lifetime to ensure high rates ($\geqslant$60 FPS) and low motion-to-photon latency ($\leqslant$20ms) when rendering high-resolution VR frames [31]. Their VR performance, hence, are much lower than that of their counterparts being tethered to high-performance workstations (e.g., Oculus Rift [22] and HTC Vive [21]).

A viable solution to this challenge is to offload the computationally expensive VR frame rendering to the nearby edge cloud [58], which then wirelessly transmits the rendered frame data back to the mobile HMD. The edge cloud nowadays, however, could be usually located over individual households with end-user desktop PCs or small-scale workstations, which have much lower capacities in both computation and communication compared to traditional cloud facilities such as data centers. They, hence, fail to provide satisfactory VR performance when serving multiple VR users in a household at the same time (e.g. multiple family members play the same multi-player VR game).

The fundamental reason of such failure is that existing mobile workload offloading techniques [37], [17], [32], [27], [26], when being applied to VR applications, serve each user independently: every VR frame for a user is separately rendered by the edge cloud and fully transmitted back to the mobile HMD. The computation and communication overheads of such remote VR frame rendering, hence, grow with the number of concurrent VR users in the following two perspectives. First, in order to provide 360° immersive experience with satisfiable image quality, every VR frame needs to be panoramic with at least 4K resolution. Comparing to traditional 3D multimedia applications which only render the partial user view in 720p resolution, rendering such panoramic frames results in 6x more computation, and this burden may quickly overload the edge cloud's computing capacity with multiple VR users. Second, rendering these panoramic VR frames also produces more than 2GB of frame data every second with 60 FPS[1], but only a portion of such data can be timely transmitted even through gigabit WiFi network. Existing video encoding techniques such as H.264 [63], on the other hand, may be highly ineffective when being applied to such VR frames, due to their ignorance of the specific VR frame context and the subsequent fixed encoding strategies.

Our solution to such excessive workload on the edge cloud builds on experimental observations from real VR applications, which indicate the VR frames being rendered and transmitted for different users as highly *redundant*. First, even in highly dynamic VR scenarios such as interactive games, our experimental studies show that movement trajectories of different VR users share more than 30% in common when they are near the same Points of Interests (PoIs) in the VR world. Such locality in VR user movements [16] leads to redundant frames with very similar scene views across different users. Second, consecutive frames of the same VR user are also correlated, because of the perspective object projection in VR applications that reduces the impact of user movement on the user view.

[1]Each panoramic VR frame with 4K resolution could contain more than 8.3 million pixels and have a raw size up to 33MB.

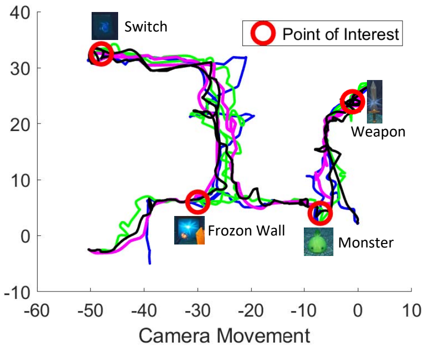
978-1-5386-9445-9/18/$31.00 ©2018 IEEE
DOI 10.1109/SEC.2018.00008

We verified that such redundancy could exceed 50%, i.e., more than half of pixels in these frames are identical with each other.

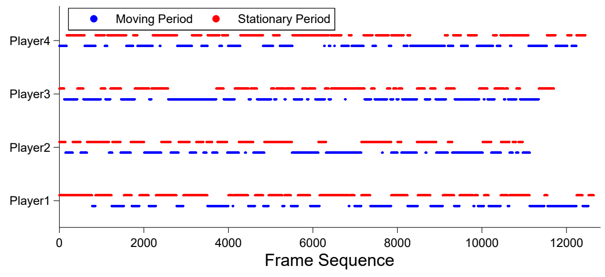
Based on these observations, in this paper we present *Multi-User Virtual Reality (MUVR)*, a systematic mobile VR framework that maximizes the efficiency of edge cloud's resource utilization to support multi-user VR. The key approach of MUVR is to adaptively *reuse* the previous results of VR frame rendering whenever necessary, by identifying and exploiting the aforementioned redundancy when the edge cloud renders VR frames and transmits these frames to the mobile HMD. In particular, MUVR eliminates redundant computations in VR frame rendering via frame memoization, which caches the invariant background view of rendered VR frames. These caches will be opportunistically reused when rendering frames for other users in the future, if they are at the similar camera locations in the virtual world. Furthermore, in order to reduce the amount of VR frame data being wirelessly transmitted to the mobile HMD, MUVR avoids transmitting full VR frames for every user. Instead, it only transmits a small portion of VR frames in full as reference frames. Then, for any other frame produced between reference frames, only its distinct portion will be transmitted to the mobile HMD as a delta image.

The major challenge of designing MUVR, however, lies in the complicated dynamics of user movements in the VR world, which make it difficult to maintain and utilize the cached VR frames. First, it is very rare that the camera locations of two VR users in the virtual world exactly match each other. The dynamic difference of such camera locations across VR users, then, complicates the decision of cache hit. Second, the efficiency of cache indexing and overhead of cache maintenance must be carefully balanced at the edge cloud. Maintaining a distributed cache at individual VR users reduces the overhead of cache indexing, but increases their local consumption of storage because the same VR image may appear in multiple users' local caches. In contrast, a centralized cache at the edge cloud maximizes the efficiency of storage utilization, but may involve frequent inter-process communications (IPC) for delivering cached images across different users.

To address these challenges, our primary idea is to maintain a two-level hierarchical cache at the edge cloud. In particular, the edge cloud maintains a central cache, which aggregates the VR frames rendered for different VR users and reuses these cached frames whenever necessary: for any new camera location being requested for VR frame rendering, the cached VR frame with the closest matching location will be transformed by image warping, so as to be reused with minimum image quality loss. On the other hand, when the VR user stays stationary in the virtual world, individual VR application locally maintains a distributed small-sized cache to reuse a precedent background image, and only requests to the central cache for rendering a new VR frame if the user movement results in perceivable change of the user view. In this way, by dynamically adapting the threshold of image warping, we are able to flexibly balance between using the central and distributed caches, so as to maximize the efficiency of cache



(a) User movement trajectories



(b) User movement timeline

Fig. 1. Users' movements in the mobile VR Fantasy application

utilization while providing satisfactory VR image quality to users.

We have implemented MUVR over the Android OS and Unity VR application engine[2] 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, MUVR is implemented in native language within the Android OS, 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 ∼5,000 Lines of Codes (LoC) in total, and our experimental results over real-world VR applications show that MUVR, when being used to simultaneously serve multiple (>4) VR users, could efficiently reduce the computation burden at the edge cloud by more than 90% with complicated scenes and intensive user movement, while reducing more than 95% of the VR frame data being wirelessly transmitted.

II. Motivation & Preliminaries

Our design of MUVR is motivated by the unique characteristics of user movement and frame rendering in VR applications. First, different VR users' movements in the virtual world could significantly overlap with each other due to the temporal and spatial locality of such movements, leading to similar background views of these users that can be memoized and reused. Second, for any single VR user, the impact of his/her

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

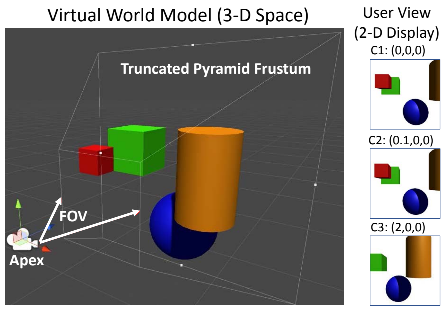


Fig. 2. The virtual world in VR applications

movement on the corresponding user view could be reduced by the *perspective projection* being used in VR applications. Such reduction results in very high redundancy across consecutive VR frames of the same user, which can be utilized by MUVR to reduce the amount of VR frame data being transmitted to the mobile HMD.

A. Locality of VR User Movement

User movements in VR applications are mostly triggered by Point of Interests (POIs) in the virtual world, which are intentionally designed to represent the application contents. Camera trajectories of different VR users, hence, would overlap when they visit the same POI. To investigate such locality of VR user movement, we conducted experimental studies over a real-world mobile VR application downloaded from Google Play: a typical role-playing VR game called *VR Fantasy (Fantasy)* [7] that allows the user to freely explore the virtual world. To collect camera traces of user movements in the application, we hacked into the dynamic-link library (DLL) of the Unity engine inside the application .apk file, and recorded the camera position for each frame being rendered. The camera trajectories of 4 VR users with Google Cardboard, as shown in Figure 1a, demonstrate a 8% to 35% overlap when the users are moving closer to the same POI.

At the same time, we observed that users' movements in the virtual world are intermittent, because they usually stop at POIs to interact with the nearby virtual objects. As shown in Figure 1b, the user character spends more than 53% of time as stationary, with only slight change of their camera positions due to the VR neck model [46] between -0.1 and 0.1 in virtual-world units ($\sim$10cm in reality).

These observations motivate MUVR to eliminate redundant computations in VR frame rendering by exploiting the locality of user movement: once a background view is rendered for a VR user, it can be reused for rendering VR frames of another user in the future, as long as the camera location of frame rendering remains the same or has only minor changes. On the other hand, such a rendered background view can also be reused for rendering consecutive frames of the same user, as long as the user stays stationary.

B. Pixel Redundancy across Frames

As shown in Figure 2, VR applications construct the virtual world as a 3D space, where virtual objects are modeled and placed at certain coordinates. In this 3D world, the user character is represented by a 2D camera, and the application view being 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 [47], [23], which emulates how human eyes see the real world. Such projection forms the 3D world as a truncated pyramid frustum, with the camera sitting at the apex point and its range being defined as the camera's field of view (FOV). Any object within this frustum is projected to and visible in the user view.

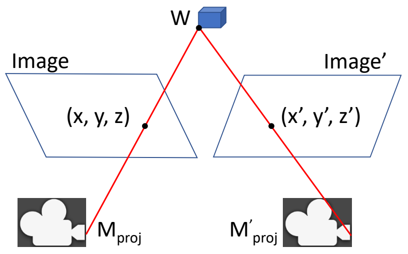


Fig. 3. VR image warping

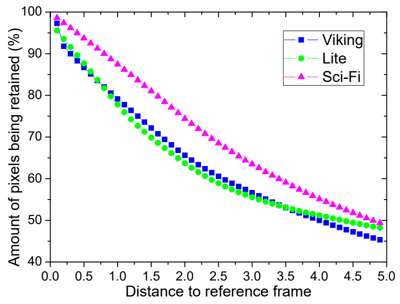


Fig. 4. Frame correlation after image warping

The most significant characteristic of perspective projection is that distant objects in the 3D world appear smaller than objects close-by, and the impact of user movement on the 2D user view will hence be reduced after object projection, which leads to large pixel redundancy between frames. Image warping techniques, in this case, are widely used by existing VR applications to reproject a rendered frame to the new camera view, which may change in the mean time when the VR frame is being rendered. For example, the most commonly used technique, *Image-based Rendering (IBR)* [43], [49], is illustrated in Figure 3. 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 $M_{proj}^{'}$, the new user view can be produced 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. Since such reprojection continuously warps the original frame's pixels to the exact positions in the new user view, it is able to precisely capture the pixel redundancy between VR frames.

In practice, image warping imposes no restrictions on the target camera position to enable the perspective reprojection in the virtual world. However, the visual quality of the warped

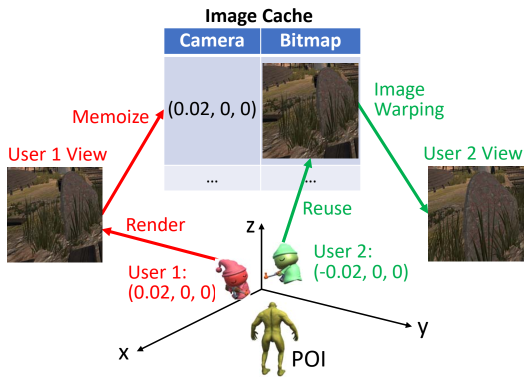


Fig. 5. Overall design of MUVR

image is subject to the accuracy of measuring the user's location in the virtual world, i.e., the user motion should be precisely depicted and reflected. Such accuracy of location measurement is inherently determined by the mobile HMD hardware, which implements the neck model and tracks the users motion in realtime. For example, the motion tracking in Oculus VR reaches a precision of 0.0003 ($\sim$0.03cm in reality) [34] and guarantees high visual quality with accurate camera location in each frame.

Based on such accurate user location measurement in the VR world, we conducted preliminary experiments to measure the extent of pixel redundancy across VR frames in practical VR applications. Our experiments randomly pick 10 reference frames from three open-sourced VR applications (Viking Village [6], Lite [4] and Sci-Fi [5]) with different VR scene complexity and character dynamics, and utilize IBR to warp these frames to the target camera views from different distances away. Figure 4 demonstrates that more than 50% of pixels can be retained in VR frames after image warping, even if the warping distance increases to 5.0 ($\sim$5m in reality). Such redundancy will be exploited in MUVR to minimize the amount of VR frames being transmitted from the edge cloud, by generating multiple delta images at different camera locations over time from the same reference frame.

III. Overview

Figure 5 illustrates how MUVR works: A centralized image cache is maintained by the edge cloud to memoize the previously rendered VR frames from all users. Then, for every new incoming request of VR frame rendering, MUVR searches the image cache with the target camera position, and reuses a cached VR image whenever possible to minimize the computation burden of VR frame rendering. When there is a cache miss, the edge cloud will render the requested VR frame in full and add the rendered frame into cache.

In particular, MUVR considers a cache hit if a cached VR image at a nearby camera position, whose distance from the target camera position is shorter than a given threshold, can be found. For example in Figure 5, the edge cloud will serve User 1's request for rendering the frame at the camera position (0.02, 0, 0) when the cache is empty. Afterwards, for User 2's request with the camera position (-0.02, 0, 0), the edge cloud will reuse and warp the cached image of User 1 to the target camera position. In practice, MUVR can flexibly adopt cache replacement algorithms to improve the cache hit rate, and the distance for image warping can also be controlled to balance between the cache hit rate and image quality loss.

After having generated a VR frame at the edge cloud, MUVR further minimizes the communication overhead of transmitting VR frames by eliminating the redundancy across consecutive VR frames. To achieve such minimization, MUVR only transmits a subset of VR frames as full panoramic images that capture all the possible user orientations at the corresponding camera positions, referred to as reference frames, to the mobile HMD. Every time when a new VR frame is needed, MUVR first renders this frame in full at the edge cloud, and then warps the most recent reference frame from its original camera view to the current user view. As a result, the delta image is synthesized via delta encoding as the difference between the originally rendered frame and the warped image from the reference frame. At the mobile HMD, MUVR warps the received reference frame in the same way to the user view. When the corresponding delta image is received, it reverses the delta encoding operations and applies the delta image to patch the visual artifacts being produced by image warping, so as to restore the full VR frame for display without any image quality loss.

How to maximize the cache utilization? MUVR designs a universal portal with a large central cache at the edge cloud to render the background views for all VR users. Such central cache aggregates the VR frames from all users and enables the cached images to be reused across different users, which significantly improves the cache utilization. However, such central cache inevitably incurs IPC overhead to deliver the rendered images to individual VR applications. To minimize such system operational overhead, a small-sized cache is also created and maintained by each VR user, which memoizes the previous background images that are locally rendered for faster reuse (see Section IV).

How to minimize the VR frame data being transmitted? As shown in Section II-B, a large amount of redundant pixels can be retained across consecutive VR frames or even after image warping over long distance. Hence, delta images for multiple VR frames can be synthesized from the same reference frame, and the size of each delta image is always smaller than the corresponding full VR frame. In practice, the sizes of delta images will grow when the user character keeps moving and results in longer warping distance. In order to ensure timely transmission of each delta image, MUVR further reduces the average size of delta images to <25 KB without impairing the VR image quality, through image compression and clipping (see Section V).

IV. VR Frame Memoization

MUVR utilizes the central image cache to memoize and reuse the background views of rendered VR frames at the edge cloud, which are always identical for a fixed camera position.

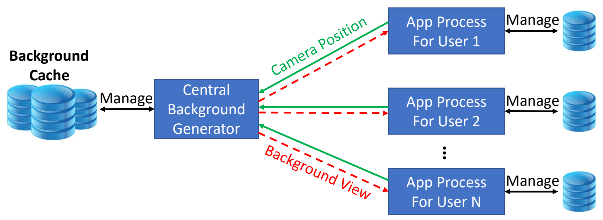


Fig. 6. The two-level cache design in MUVR

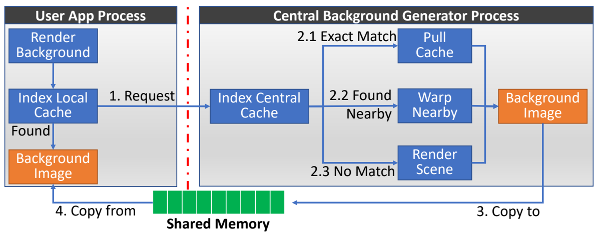


Fig. 7. VR frame rendering in MUVR

In practice, the background views, after being transmitted to the mobile HMD, will be combined with the foreground objects produced by the corresponding local VR application for up-to-date animations and user interactions.

A. Two-level Image Cache Design

An intuitive strategy to memoize and reuse the rendered VR images is to maintain an image cache in the VR application processes of different users. However, the cache utilization in such scheme is impaired because the rendered images cannot be reused across different users, even if their camera positions are the same. In addition, extra memory consumption would be incurred because each user process may store its own copy of the same image.

On the other hand, a centralized cache could reduce the memory consumption by coalescing the rendered VR frames of all users, and improve the cache utilization by allowing these frames to be reused across multiple users. However, IPC operations such as shared memory would be involved to deliver a rendered VR image from the central cache to individual users and consume additional system resources.

Based on the observation that user movement in VR applications is intermittent with long stationary periods in Section II-A, MUVR devises a two-level cache mechanism which optimizes the cache performance with the advantages in both centralized and distributed cache schemes. Specifically, as shown in Figure 6, the edge cloud maintains a small local cache in the VR application process of every VR user, as well as a large central cache through a corresponding central process. These two levels of caches collaborate together to generate the background view for all VR users at the edge cloud: When a VR user keeps stationary, the camera positions would be mostly unchanged and the background view of consecutive frames can be reused from the user-specific local cache without any IPC operations at the edge cloud. On the other hand, in cases of cache miss in the local cache, the corresponding user process sends a request with the latest camera position to the central process, which serves as the universal portal to reuse the rendered images across multiple users with high cache utilization.

B. VR Frame Rendering

Based on this two-level cache design the procedure of rendering a VR frame in MUVR is shown in Figure 7. Whenever a new VR frame is needed at the edge cloud, the corresponding user process first looks up the local cache with the latest camera position, and the cached background view will be reused if a matching entry is found. Otherwise, a request with the current camera position will be sent to the central process at the edge cloud, where a specialized background view generator will produce the target view by looking up the central cache with the following options.

First, when the background view of the current camera position has been previously rendered by other VR users and memoized in the central cache, the cache indexing will find the entry that matches exactly with the requested camera position. Therefore, the memoized background view can be directly pulled from the cache as the rendering result.

Second, when the camera positions of two VR users mismatch, the cache indexing would fail to find an entry with exact match. However, if such mismatch is minimal, their background views still manifest large amounts of pixel redundancy, which could be utilized by MUVR to avoid unnecessary computation. To do so, MUVR exploits such pixel redundancy between adjacent frames and reuses a nearby background view with image warping: the background generator iterates through all cached entries and searches for the entry whose camera position is closest to the target camera position. If such distance is smaller than the given threshold, the background generator warps the view in this cache entry to the target camera position. In particular, MUVR adaptively adjusts such threshold to balance between the image quality and computational overhead: a large threshold enables a view to be warped to a farther distance with more computation reductions, in the exchange of degraded image quality. We will further investigate such tradeoff and the best choice of such threshold via experimentation in Section VIII.

Last, if the background generator cannot find any reusable entry, the background generator would fall back and render the background view with the application engine, which is then added into the cache for possible future use. Particularly, if a new image arrives while the cache has reached its maximum capacity, an existing entry would be removed according to the cache replacement policies (e.g., LRU or LFU).

After the background view is generated, it would be delivered to the user process so as to be combined with the foreground view. In order to ensure the efficiency of image delivery, a chunk of shared buffer will need to be established between the background generator and the user process, so as to avoid the expensive memory operations on the large bulk of image data.

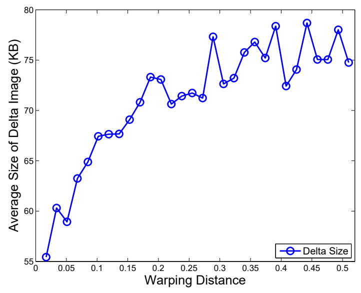


Fig. 8. Average size of delta images after compression

V. Delta Image Synthesis

As stated in Section III, the VR application views being generated at the edge cloud will be further shrunk by delta encoding, so that only their distinct portions will be transmitted to mobile HMDs as delta images with the minimum transmission overhead. MUVR synthesizes a delta image through per-pixel subtraction between the full VR frame and the warped image from the corresponding reference frame. Specifically, for VR frames with 8-bit pixel channels[3], the pixel value in each channel of the delta image is computed as

$$Delta = \frac{Full - Warped}{2} + 127, \tag{1}$$

which maps the positive and negative differences between the full VR frame and the warped image to lighter and darker colors, respectively. Similarly, when restoring the full VR frame, the mobile HMD patches the delta image to the warped image by inversing the subtraction as

$$View = \min[2 \cdot (Delta - 127) + Warped, 255]. \tag{2}$$

Based on such encoding, MUVR further reduces the size of delta image from the following two aspects. First, the edge cloud compresses each delta image before sending it, and uses the decompressed version of compressed reference frames at the edge cloud to ensure the consistency of delta image synthesis with the mobile HMD. Second, the edge cloud clips each delta image according to the current user camera orientation and FOV, In this way, it avoids transmitting any VR frame data outside of the current user view, which is unlikely to be noticeably changed during the short time period of transmitting a delta image. Our experimental studies show that the delta encoding with compression and the viewport clipping reduce $\sim$25% and $\sim$65% of VR frame data respectively, which hence minimize the size of a delta image to be $<$ 25 KB without any VR image quality loss. Such reduction, on the other hand, also allows a reference frame to be used for synthesizing more delta images and further minimizes the total amount of VR frame data being transmitted.

A. Delta Image Compression

The most straightforward approach to reducing the size of a delta image is to compress the image at the edge cloud before transmitting it to the mobile HMD. Since the size of a delta image is much smaller than the full VR frame, each delta image, after being processed by existing lossy compression techniques such as H.264 [63], could be efficiently decompressed by the mobile HMD before the next delta image arrives. As shown in Figure 8, the average size of delta images with H.264 compression continuously increases along with the warping distance, which reduces the amount of redundant pixels in VR frames when it increases. Even when the warping distance is very long ($\sim$0.5), such average size could be lower than 80 KB with compression ratio (23)[4].

[3]The pixel value in an 8-bit channel ranges from 0 to 255.

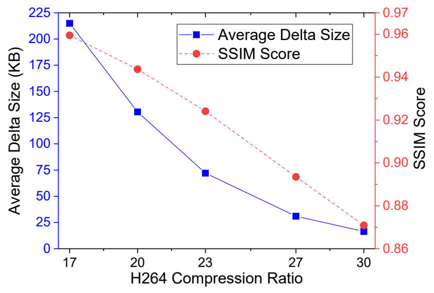


Fig. 9. Balancing between delta size and image quality

However, applying such a lossy compression technique over delta images in MUVR is challenging, because it may result in discrepancy in delta image synthesis between the edge cloud and the mobile HMD, further impairing the VR image quality. More specifically, the edge cloud synthesizes a delta image by warping from an uncompressed reference frame, but has to send such a panoramic reference frame to the mobile HMD after compression. The warped image from the decompressed reference frame at the mobile HMD, hence, will have more visual artifacts due to lossy data compression and affect the correctness of delta patching.

To address this challenge, MUVR retains a decompressed version of each compressed reference frame, and uses this version for image warping at the edge cloud to ensure consistency of delta image synthesis. The correctness of delta patching at the mobile HMD, then, could only be impacted by compression over the delta images themselves. In practice, such impact can be controlled by adopting different H264 compression ratios that balance between the VR image quality and delta image sizes. To evaluate such balance, we conducted preliminary experimental studies by using the structural similarity (SSIM) metric [62] over the Viking Village VR application [6]. According to [19], 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. Our experiment results in Figure 9 show that any H264 compression ratio lower than 27 results in a satisfiable level of VR image quality, and could further reduce the average size of delta images down to 25 KB.

B. Delta Image Clipping

The size of delta image could be further reduced by exploiting the limited FOV of today's mobile HMDs, which is usually

[4]H.264 allows different compression ratios by adjusting its Constant Rate Factor (CRF), which decides the amount of data bits being used for each image frame.

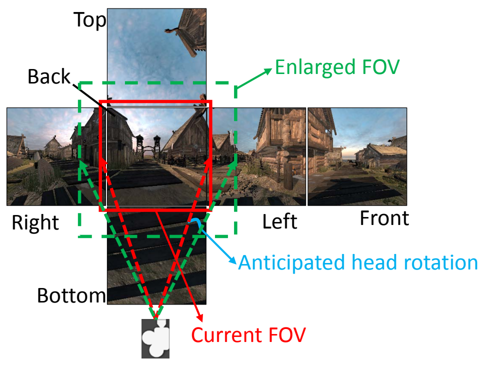


Fig. 10. Delta image clipping

smaller than 120° [3]. As a result, instead of synthesizing and transmitting a delta image over the 360° panoramic view, MUVR 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 to the edge cloud 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 edge cloud in advance. Our solution to this challenge, as shown in Figure 10, is to further enlarge the FOV of image clipping by $X°$ in both sides, to cover the possible change of user view [35]. 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 [28], the value of X is merely 17.5 for a 22ms latency of delta image transmission.

As shown in Figure 11, after H.264 compression, such clipping further reduces the size of delta images by up to 65%, when being applied to the three open-sourced VR applications that we described in Section II-B. In particular, such size could be effectively controlled within 25 KB when 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.

VI. Implementation

We implemented MUVR 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 4,000 lines of C++ code as a plugin to the Unity engine, and 1,000 lines of C# code as a Unity engine script. We use x264 [9] as the encoder and decoder of delta images.

A. Edge Cloud Operations

MUVR runs a clone copy of each VR application at the edge cloud, and renders VR frames according to the user inputs such as controller operations received from the mobile HMD as system events. To retrieve the rendered full VR frames from

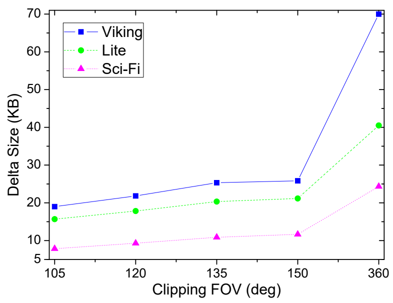


Fig. 11. Delta size with different clipping FOV. H.264 with CRF=23 is being used.

the application binary, we exploit the hook of the application engine and attach a post-processing script to the specialized VR camera. This script transforms the depth buffer into a greyscale image, and then reads the raw pixels of the color and depth images into the main memory.

On the other hand, in order to render the panoramic reference frames at the edge cloud, 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 user view.

B. Central Cache Implementation

One intuitive approach to implementing the central cache at the edge cloud is to assign a dedicated storage space that can be synchronously shared by all VR user processes. Specifically, each user process of VR application needs to synchronize its cache access with other user processes in a distributed manner, during which IPC operations can be involved to coordinate among them. However, the run-time overhead of such synchronization increases significantly when more user processes are being operated at the edge cloud and could hence result in severe contention for cache access. For example, the synchronization may indefinitely block some user processes from execution, or lead to race conditions that may retrieve wrong background views for VR display. To address these problems, MUVR deploys a dedicated central process with shared cache, which greatly simplifies the interaction among user processes with high performance. In particular, during any cache access in background rendering, the user process only needs to interact and synchronize with the sole central process without contention. On the other hand, the central process owns a global view of all VR users, which could help manage the cache resources more effectively.

C. Edge Cloud System Integration

The major challenge of MUVR implementation on the edge cloud is how to efficiently support different VR applications and hardware drivers in a generic manner. First, VR applications are heterogeneous in their shading languages and

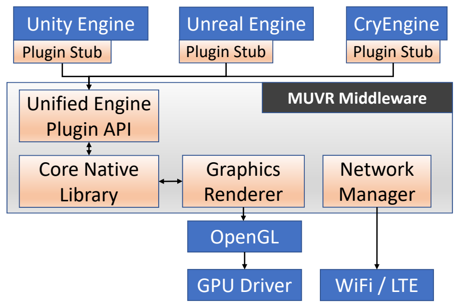


Fig. 12. MUVR as a mobile middleware

scripting APIs being used. For example, the Unity engine uses either JavaScript or C# as the script language, but the Unreal engine[5] only supports C++. Second, hardware drivers from multiple vendors usually provide heterogeneous interfaces for hardware operations. Supporting pixel reuse within the VR application binary or hardware driver, hence, could lead to large amounts of re-engineering efforts for different hardware. Such operations, on the other hand, if being done in the user space, would also be much less effective due to frequent interaction with the system hardware.

To address these challenges and retain generality, we integrate MUVR into the operating system of the edge cloud, and implement it as a middleware of shared libraries between VR applications and OS drivers. Such implementation ensures the isolation between the heterogeneous hardware drivers and VR applications, and hence enables MUVR deployment on any edge cloud platforms with the minimum reprogramming efforts over VR applications. As shown in Figure 12, the core of MUVR is implemented as an OS library in native language to regulate the main MUVR functionality, such as the two-level cache management and IPC operations. The core library then interacts with the graphics renderer, which manages frame buffers and invokes APIs directly from OpenGL for image warping and delta encoding. 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 [61] and Nvidia's Cg [42].

On the other hand, the core library should interact with VR application 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 needs to find out the current camera position and hence will invoke

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

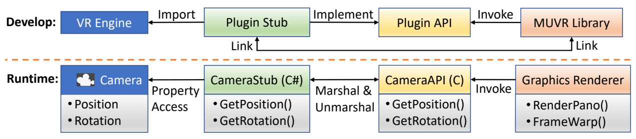


Fig. 13. Unified interaction with application engine

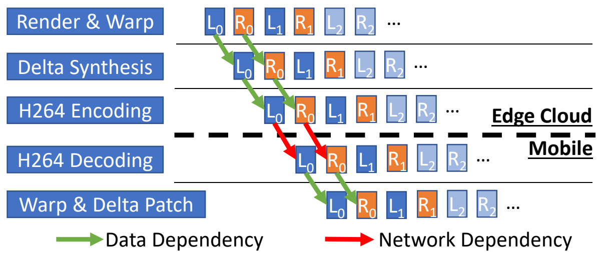


Fig. 14. Pipeline processing of MUVR

the *GetPosition()* function in the plugin *CameraAPI*, which is written in native C. This function marshals the request to the managed format in C#[6] and triggers the engine-specific script *CameraStub* to access the position property of the camera object. Afterwards, the position values of the engine camera are marshaled to the native format and returned to be processed by the graphics renderer.

D. Parallel and Pipeline Processing

MUVR is also implemented to maximize the performance of the edge cloud and reduce the response latency to the mobile HMD. We divide the MUVR operations into individual tasks and execute them in a pipeline manner for the two stereo eyes in each frame. As shown in Figure 14, when the system is working to render and warp for the right eye of frame 0 (denoted as R_0), it is simultaneously encoding the delta image for the left eye of frame 0 (L_0). To avoid pipeline stalls or resource idleness due to the heterogeneous computational complexity in different stages, we maintain a request queue for each stage, which can then proceed to the next task immediately without waiting. In addition, we also share the VR frame memory and allow the memory handle to be passed between stages, so as to avoid copying the bulky VR frame data itself.

With the pipeline, the mobile VR performance is constrained by the most computationally expensive stage in the pipeline, whose processing time is further reduced in MUVR by exploiting the system parallelism. In particular, when the limited GPU resources on low-end mobile HMDs are fully used by image warping and hence incapable of decoding the compressed delta images timely, MUVR splits a delta image into multiple segments and dedicates specialized CPU threads for faster software decoding.

VII. Making VR Apps with MUVR

The generic design and implementation of MUVR significantly reduce the burden of VR application development,

[6]http://msdn.microsoft.com/en-us/library/ms235282.aspx

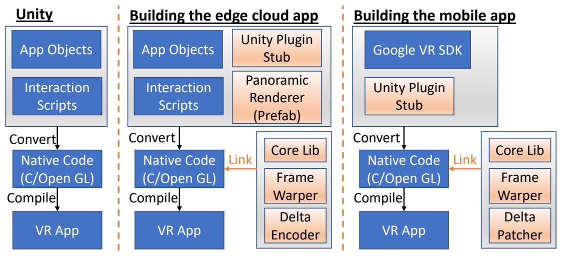


Fig. 15. Making VR apps with MUVR

with the Unity engine as the target VR software platform. Typically, as shown in Figure 15, 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 VI-C as easy-to-use modules, based on which VR applications can be built for both the edge cloud and the mobile HMD. As shown in Figure 15, the developers simply need to import the modules provided by MUVR by copying the libraries to the application folder and create special prefab[7] 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 the edge cloud and the mobile HMD, and a prefab of panoramic renderer should be created at the edge cloud to render panoramic reference frames.

In this way, the components in our implementation of the MUVR middleware are completely decoupled from the specific VR hardware platform, and hence can be directly integrated into any VR application engine. Such integration minimizes the amount of required efforts to build VR applications on top of MUVR, by allowing the application engine to incorporate MUVR into the VR application binaries automatically at compile time.

VIII. Evaluation

In this section, we first evaluate the performance of MUVR on edge cloud, by measuring the computation and communication reductions when multiple users are running VR applications with the edge cloud. Besides, we also evaluate the mobile VR performance in terms of the mobile VR frame rate, image quality and motion-to-photon latency. Our experiment results show that MUVR can significantly improve the mobile VR performance when multiple VR users are being served by the resource-constrained edge cloud.

[7]An object acts as a template with predefined scripts and properties.

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

Fig. 16. Screenshots of VR applications

A. 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 edge cloud server. We use a Google cardboard as the experimental VR headset with a FOV of 90°. The mobile HMD is connected to the edge cloud server via campus WiFi, which has an average throughput of 100 Mbps and transmission latency of 3.5 ms. Each experiment is conducted multiple times for statistical convergence.

TABLE I
Statistics of VR scene complexity

Application	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

Our experiments are conducted over the three open-sourced VR applications listed in Section II-B. As shown in Table I, they present different levels of VR scene complexity and dynamics. The experiment results over them, hence, are representative and can be generally applied to other VR applications with similar levels of VR complexity.

Each panoramic delta image, before being transmitted, is clipped with a FOV of 135°, which allows a 22.5° head rotation with Google cardboard and tolerates 28 ms delay for transmission and decoding. X264 with default CRF=23 is being used for delta encoding and decoding. In our experiments, otherwise explicitly specified, we set the capacity of the central cache as 300 background images, which correspond to 5 seconds of video frames, and 3 images for the per-user cache. We set the threshold of warping distance to reuse a nearby background view as 0.1 virtual unit. The number of concurrent users running VR applications on edge cloud is 4.

We compare MUVR with three existing VR schemes:

- **Local:** VR applications are solely running on the mobile HMD.
- **Thin-client**: VR frame of each user is rendered separately by the edge cloud and transmitted in full to the mobile HMD [8].
- **Furion**: A VR frame is collaboratively rendered at the edge cloud and mobile HMD. Panoramic VR backgrounds are rendered at the edge cloud and pre-fetched by the mobile HMD for all possible directions of user movement. Foreground VR objects are all rendered at the mobile HMD itself [33].

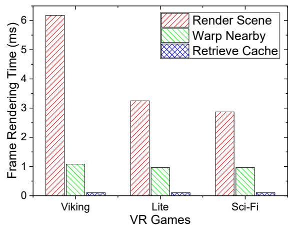


Fig. 17. The average time to render a background view

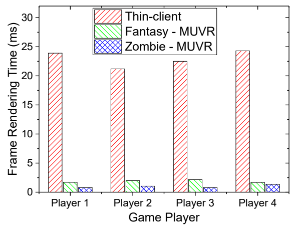


Fig. 18. The average time to render the background views in a session

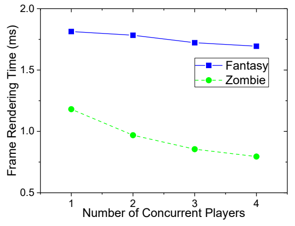


Fig. 19. The VR performance with concurrent users

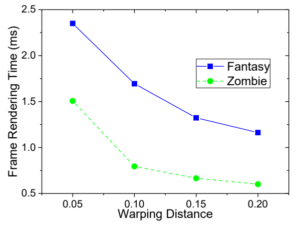
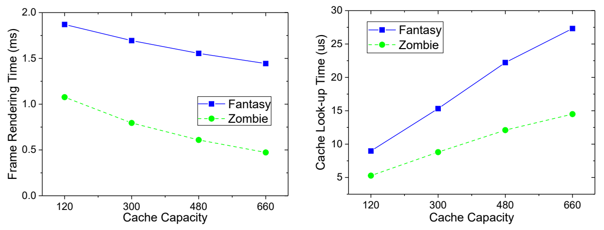


Fig. 20. The VR performance with different warping distance



(a) Frame Rendering Time (b) Cache Look-up Time

Fig. 21. The impact of cache capacity on VR performance

B. *Improvement of Edge Cloud Performance*

Our experiment results show that, by reusing the previously rendered images, MUVR could reduce 90% of the rendering computations and 95% of network communications on edge cloud. In addition, the two-level cache design could reduce 30% of memory consumption with a central cache and reduce 32% of IPC operations with a small per-user cache. Our experiments are being performed over the camera traces of 4 VR users when they are playing two highly active VR games, i.e., *VR Fantasy (Fantasy)* [7] and *Dead Zombies Survival VR (Zombie)* [2]. During trace collection, we ask all VR users to perform the same task to explore the virtual world for 5 minutes. To eliminate the impact caused by the users' unfamiliarity with game operations, we allow VR users to try each application for 1-2 minutes before starting the trace collection. Each experiment session is operated with such a camera trace containing 3,000 VR frames.

1) Computation Reduction: In this section, we evaluate the effectiveness of MUVR on reducing the edge cloud computations of VR frame rendering. We first benchmark the average execution time to render a single VR background frame with different cache indexing results. As shown in Figure 17, the execution time to render a background frame is negligible when an exact match is found in the cache and the cached image is retrieved and reused directly. On the other hand, if a nearby background image is reused and warped to the target camera position, MUVR can still achieve more than 3x speedup to render the background frame, because the pixel reprojection in image warping is more computationally efficient than the pixel value computation in graphics rendering. Moreover, the computational complexity of image warping is correlated only to the size and resolution of the reference image and hence such speedup increases to 6x for the Viking application, which has the most complex scene setup.

We have also evaluated the MUVR's workload reductions on the edge cloud in practical scenarios. Figure 18 shows the average execution time to render a background image for each user. From the figure we can see that MUVR reduces more than 90% and 95% of frame rendering time for the Fantasy and Zombie traces respectively, by reusing the previously rendered results. The Zombie application achieves a higher computation reduction because it has more restrictions on the user movement and higher movement locality is observed, which leads to higher hit ratios during cache indexing.

2) Factors that influence MUVR performance: In this section, we evaluate how the performance of MUVR could be influenced by various factors, such as the number of concurrent users, the threshold of warping distance to reuse a nearby cached entry and the maximum capacity of the cache. During the experiments, we measure the average time to render a frame for player 1 with different system setups.

First, we evaluate how the number of concurrent users could influence the frame rendering time and the experimental results are shown in Figure 19. We can see that the edge cloud spends less time to render VR frames for any user when the number of concurrent users increases, because the locality of user movement leads to a higher chance to reuse a rendered image from other users. Compared to single-user play, the edge cloud could reduce 35% of frame rendering time for the Zombie trace, when 4 players are running the VR applications concurrently.

We also evaluate the influence of the warping distance to the frame rendering, by adjusting the threshold of warping distance to reuse a nearby cached entry. As shown in Figure 20, the average rendering time decreases 51% and 60% for the Fantasy and Zombie traces respectively, when the warping distance increases from 0.05 to 0.2. Such reduction is because a larger warping distance would allow a cached image to be reused by a larger range of camera positions and reduce the number of frames to be generated by expensive geometry rendering. Despite such improvement on cache utilization, the threshold of warping distance cannot be increased arbitrarily because the view disocclusions with farther warping distance lead to more visual artifacts, which degrade the visual quality of the warped image and impair the user experience to an unacceptable level.

MUVR imposes no hard requirements on the minimum cache size required for VR frame reuse. The cache capacity, however, is related to the effectiveness of such reuse and the corresponding frame rendering time on the edge cloud. Such

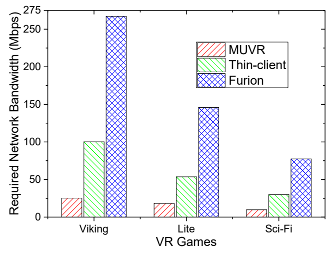


Fig. 22. Network bandwidth required by MUVR

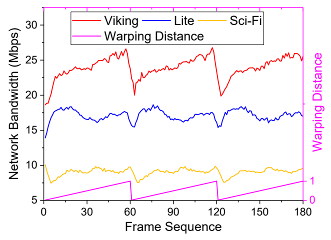


Fig. 23. Temporal fluctation of network bandwidth consumption

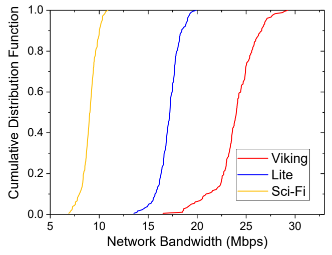


Fig. 24. Cumulative distribution of network bandwidth consumption

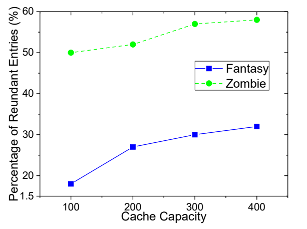


Fig. 25. Redundancy in entries with per-user cache

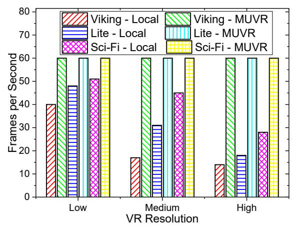


Fig. 26. Frame rate with different VR resolutions

correlation is evaluated in our experiments by adjusting the maximum number of background images in the cache. As shown in Figure 21a, the average frame rendering time reduces 24% and 57% for the Fantasy and Zombie games respectively, when the cache capacity increases from 120 to 660. On the other hand, the cache look-up time would linearly scale up with the cache size as shown in Figure 21b, which however is no more than 30 us and can be negligible. Therefore, the edge cloud could trade the storage space for computation reductions if it is equipped with large system memory or external storage.

3) Communication Reduction: MUVR also aims to address the constraints of communication capacity on the edge cloud. Being different from Furion [33] which requires gigabit WiFi connection to transmit full VR frames, Figure 22 shows that MUVR requires at most 25 Mbps of network bandwidth to support a VR user, which enables to transmit the VR frames of multiple users efficiently with existing WiFi protocols [40], [39], [41]. In addition, we have evaluated the transient consumption of network bandwidth to transmit the delta image over 180 VR frames. As shown in Figure 23, the frame transmission requires higher network bandwidth when the user character moves farther and results in larger warping distance, because the view difference increases in these cases and the corresponding delta image needs to encode more pixel details. Nevertheless, Figure 24 further shows that MUVR is able to restrain the required network bandwidth to be always below 30 Mbps, because of the small size of the delta image. In particular, such bandwidth consumption is also related to the specific scene complexity and dynamics of VR applications. For example, Figure 24 shows that more than 90% of VR frames in the Viking game are consuming less than 26.5 Mbps of network bandwidth, and this number is even as low as 10 Mbps for the Sci-Fi game.

4) Effectiveness of the Two-level Cache: In this section, we evaluate the effectiveness of the two-level cache mechanism. First, we evaluate the effectiveness of the central cache on reducing the memory consumption. To do so, we maintain a cache with different capacities for each user and measure the percentage of redundant entries after merging the cached entries of all users. As shown in Figure 25, more than 30% of redundancy can be observed for the cached entries of all users, which can be coalesced in the central cache so as to save the cache memory consumption.

We also evaluate the effectiveness of the small per-user cache, which improves the cache indexing efficiency with reduced IPC operations. Our experiment results show that the cache hit ratio for the local cache could be as high as 32% and 68% for the Fantasy and Zombie traces respectively, because of the intermittent user movement and long stationary periods. When the cache indexing finds a match in the local cache, it avoids to copy the rendered images from the central background generator, which eliminates the IPC operations and saves the memory bandwidth consumption.

C. Improvement of Mobile VR Performance

In this section, we evaluate the performance of MUVR in terms of the key metrics that directly impact the user experience of mobile VR, including the frame rate, image quality and motion-to-photon latency. In our experiments, by avoiding expensive VR frame rendering at the mobile HMD, MUVR 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 (required by 60 FPS) to ensure responsive user interactions. Such experiment results indicate that MUVR meets the stringent requirements of mobile VR on system performance and enables satisfactory VR experience without any possible motion sickness.

1) Frame Rate: As shown in Figure 26, the frame rate provided by MUVR 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. Note that, the maximum FPS that MUVR can achieve in our experiment is limited by the screen refreshing rate at the mobile HMD that is being capped at 60Hz, and could hence be further improved on future mobile devices which support higher screen refreshing rates (e.g., 90Hz). The reason for such improved mobile VR performance is that the

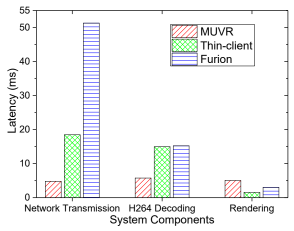


Fig. 27. Breakdown of system latency in MUVR

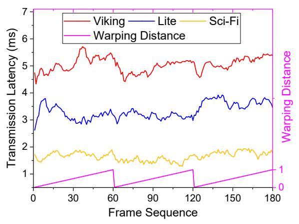


Fig. 28. Temporal fluctuation of frame transmission latency in MUVR

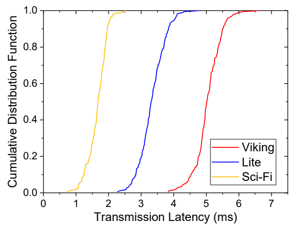


Fig. 29. Cumulative distribution of frame transmission latency in MUVR

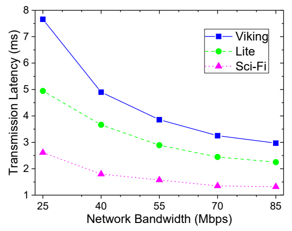


Fig. 30. Transmission latency with different network bandwidth

image warping in MUVR decouples the settings of graphic quality and scene complexity from rendering complexity and renders with high computational efficiency.

TABLE II
VR IMAGE QUALITY (SSIM)

Rendering Scheme	Viking	Lite	Sci-Fi
Local Frame Rendering	0.8133	0.8766	0.8832
Thin-client	0.8783	0.8834	0.9263
MUVR w/ Stationary User	0.9569	0.9599	0.9681
MUVR w/ Moving User	0.9241	0.9210	0.9557

2) Image Quality: We evaluate the mobile VR image quality provided by MUVR using the SSIM metric [62], which quantifies the image quality degradation in MUVR from the pristine high-quality image rendered at the edge cloud. The results in Table II show that MUVR ensures high image quality (SSIM $>$ 0.9 [19]) in all the VR applications with fast user movement, and significantly outperforms that of local frame rendering and thin-client. 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.

3) Latency: Motion-to-photon latency is critical in VR to ensure user experience and avoid motion sickness, and such latency depends on 1) the transmission delay of reference frames and delta images, and 2) the computation delay of frame decoding, image warping and delta patching at the mobile HMD. These delays over the Viking application are averaged over all the VR frames being transmitted, and Figure 27 shows that the total processing latency for each VR frame is less than 16ms. More specifically, the transmission delay in MUVR is about 4.8 ms due to the minimized size of delta images, and is less than 10% of that of Furion [33] which pre-fetches the panoramic background images for all possible directions of user movement. Similarly, the frame decoding delay in MUVR is also 66% lower than the existing schemes because of the smaller amount of VR frame data being transmitted. At the mobile HMD, MUVR takes about 5.1 ms for frame rendering, which is slightly higher than other schemes due to the extra overhead of image warping and delta patching.

In addition, the transmission latency could also fluctuate due to the wireless link condition and different sizes of delta images. Our experiment results over 180 VR frames in Figure 28 show that such transmission latency could increase when

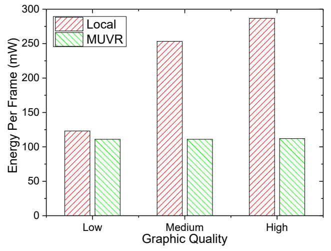


Fig. 31. Energy consumption of rendering a frame

the warping distance increases for generating VR frames, due to the larger view difference and the subsequent increase in delta size. Despite such increase, Figure 29 shows that MUVR is able to effectively control such transmission latency within 6 ms in all cases, which ensures the smooth VR experience with responsive user interactions.

Network bandwidth could also be a key factor that impacts the transmission latency for delta images. From the experiment results in Figure 30, we can see that the transmission latency decreases linearly with the available network bandwidth, which further improves the responsiveness of VR experience.

4) Energy Efficiency: We evaluate the energy efficiency of MUVR over the Viking application, by measuring the average amount of power consumed by the mobile HMD for rendering and displaying each VR frame. From the results in Figure 31, we can see that MUVR 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, MUVR 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.

IX. Related Work

Memoization Memoization techniques have been extensively studied to reduce the computation overhead of computer systems by reusing the previous computation results. CPU memoization in both software and hardware level has been widely studied and applied in real systems for decades. [57], [56], [38] utilize function interception to enable software memoization of any computationally expensive pure function through the compiler and linker based techniques. Hardware memoization schemes [12], [54], [59] exploit the temporal and spatial coherence in programs and reuse the result of instruction executions. Recently, the emergence of general purpose computing on GPU has driven the development of memoization on GPU. GRU [65] efficiently leverages the GPU full-virtualization technology to memoize and reuse GPU computation results, and hence transparently enables VMs in the cloud to share GPU on kernels without modification of existing device drivers and operating systems. Nonetheless, these schemes are orthogonal to MUVR because MUVR focuses on reusing the graphics rendering results instead of general-purpose computing memoization.

Memoization has also been studied to improve the performance of graphics rendering. Flashback [15] pre-renders all the VR frames on the cloud and caches the rendered frames at the HMDs' local storage, so as to alleviate the mobile run-time computation burden. However, it fails to adapt to the run-time dynamics of VR applications and consumes a huge amount of storage space at mobile devices (e.g., 50GB data for each VR application). Reverse projection caching [45], [50], [52] allows fragment shaders to store the calculation results and reuse such results for the visible surface points in future frames, by projecting the point back to the cached camera surface. [13], [29] reduce redundant fragment shading executions by caching the input signatures and reusing the results of fragment shading in mobile GPU hardware, but incur large amounts of computational overhead to compare different inputs.

Mobile Offloading and Cloud Gaming: General-purpose mobile offloading reduces the local computational burden of mobile devices, by adaptively partitioning the computing tasks and offloading only the most appropriate portion to the cloud for remote execution [25], [36], [18], [27]. 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.

Our proposed design of MUVR is related to prior work on cloud gaming [24], [51], [30]. 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. MoVR [11], [10] enables multi-Gbps wireless communication to VR headsets via mmWave wireless technology, but relies on specialized hardware support and line-of-sight connectivity. Instead, MUVR significantly reduces the amount of VR frame data being transmitted, and hence maximizes the mobile VR performance over conventional wireless networks.

Collaborative Rendering: Recently research on collaborative rendering splits the computing workload of rendering individual frames between the cloud and local mobile devices, so as to reduce the local devices' computational burden without impairing the image quality. For example, Kahawai [19] only renders the key frames over mobile devices and renders all the other frames at the cloud, so as to avoid transmitting key frames which usually have large sizes. These existing techniques, nevertheless, are complementary with our design of MUVR. More specifically, MUVR's caching scheme can be easily applied at the edge cloud to reduce its computation overhead of background rendering. The mobile HMDs, on the other hand, could also utilize their idle resources to locally render the panoramic reference images whenever possible, so as to avoid redundant transmissions of VR frames.

Furion [33] separates the VR scenes into background and foreground layers, and streams only the panoramic background images to mobile devices, but has to speculate future user movement for prefetching and hence suffers from misprediction. Instead, MUVR does not involve any pre-fetching of VR frames, and is hence resistant against sporadic VR application events or user behaviors.

Graphic Processing: Image-based rendering [43] 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. [49] aims to mask the network latency by provisioning an extra view at deliberately selected location to fill the disoccluded holes, but requires more computations to warp the extra image. Post-processing techniques [60] interpolate or extrapolate the disoccluded view, but lead to blurry regions. In contrast, MUVR 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.

Some other schemes [53], [55] propose to adapt the resolution for different parts of the panoramic images according to the user viewpoint, so as to reduce the amount of image frame data being transmitted. These techniques, however, still transmit full VR frames and are hence susceptible to vigorous user head rotation or movement in intensive VR scenarios. In contrast, MUVR transmits only delta images with minimum sizes to the mobile HMD, enabling fast response to any dynamic user behavior.

X. Discussions and Future Work

A. Supporting Augmented Reality

Augmented Reality (AR) [14] seamlessly overlays the virtual objects onto the physical world, so as to enhance users' perception to the natural environment and offer enriched interactive user experience. Although the focus of MUVR is to improve the performance of mobile VR via edge cloud, it can also be extended to support AR scenarios with minimum modifications due to the similarity between VR and AR systems. To enhance the AR performance, besides using the edge cloud to render the background view that is directly streamed from the mobile camera, MUVR could also offload the rendering of stationary background AR objects (e.g., a complex AR castle lying on the ground), which are overlaid on top of the camera view, for remote cloud processing. In this way, the rendered AR backgrounds could also be reused among multiple users at the edge cloud, so as to reduce the edge's computation and communication overhead.

Such extension, on the other hand, could be challenging for specific AR applications, especially when they have only few virtual objects being superimposed on the camera view of the physical environment. In this case, the computation overhead of generating the deltas from remotely rendered AR frames may dramatically increase, and also leads to larger sizes of deltas themselves. In the future, we will study the possibility of precisely monitoring such computational complexity of rendering virtual objects, so as to dynamically adjust the decision of remote frame rendering at runtime.

B. Helping with VR Prediction

Many existing techniques improve mobile VR performance by predicting VR user behaviors in the future and pre-act accordingly [33], [35], but could easily suffer from misprediction due to the the unexpected VR dynamics or sporadic VR application events. In particular, if the edge cloud simply clips the rendered panoramic VR frames by predicting the VR user's head motion and FOV for reduced VR frame data transmission, the corresponding misprediction would lead to serious quality degradation of VR images, especially when the user rotates the head abruptly with vigorous angular velocity.

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

C. Defending against Side Channel Attack

Sharing VR frame data among multiple users could bring various security and privacy concerns at the edge cloud, especially the side channel attack that analyzes the cache utilization and the corresponding frame rendering time for inferring other users' behaviors in VR applications. MUVR is able to mitigate such attack by monitoring the average rendering time for VR frames and intentionally delaying the transmission of some frame data when cache hits, so as to obfuscate the cache utilization statistics. Such obfuscation, however, will impair the mobile VR performance, and we will experimentally investigate such tradeoff between VR performance and user protection in our future work.

D. Subjective VR User Experience

Section VIII-C has demonstrated the high performance of MUVR with several key computation and communication metrics that determine the user experience of mobile VR. Such VR user experience in practice, however, also depends on many subjective factors, especially the VR users' preference and sensitivity to motion sickness. In the future, we plan to carry out a field study to learn the subjective VR experience from actual users in real-world VR applications. We plan to collect and analyze the user experience in the following aspects: 1) the smoothness of view display, 2) the responsiveness of user interactions, 3) the visual quality of the user view. We believe that the performance evaluation in Section VIII-C can be projected to this field study and MUVR could outperform existing VR schemes in all metrics.

XI. Conclusion

In this paper, we present MUVR, a systematic mobile VR framework that maximizes the efficiency of utilizing the edge cloud's computation and communication resources to serve multiple VR users from mobile HMDs. MUVR adaptively identifies and utilizes the redundancy across VR frames from different users to avoid unnecessary computation of VR frame rendering at the edge cloud, and also exploits the redundancy across consecutive VR frames of the same user to reduce the amount of VR frame data being wirelessly transmitted. Based on the implementation and evaluation over Android OS and Unity engine, we demonstrate that MUVR significantly reduces the computation and communication burden of the edge cloud, hence improving the performance of supporting multi-user mobile VR.

Acknowledgments

We sincerely thank our shepherd Mahadev Satyanarayanan and anonymous reviewers for their valuable comments and feedback. This work was supported in part by the National Science Foundation (NSF) under grant number CNS-1617198, CNS-1812399, CNS-1812407 and CNS-1826884.

References

[1] Asynchronous TimeWarp. https://developer.oculus.com/documentation/mobilesdk/latest/concepts/mobile-timewarp-overview/.

[2] Dead zombies survival VR. https://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] Sci-Fi Modular Environment. https://assetstore.unity.com/packages/3d/environments/sci-fi/sci-fi-modular-environment-3426.

[6] Viking Village. https://assetstore.unity.com/packages/essentials/tutorial-projects/viking-village-29140.

[7] VR fantasy. https://play.google.com/store/apps/details?id=com.Chibig.VRFantasy.

[8] Wowza Streaming Cloud. https://www.wowza.com/solutions/streaming-types/virtual-reality-and-360-degree-streaming/.

[9] x264. http://www.videolan.org/developers/x264.html.

[10] O. Abari, D. Bharadia, A. Duffield, and D. Katabi. Cutting the cord in virtual reality. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 162–168. ACM, 2016.

[11] O. Abari, D. Bharadia, A. Duffield, and D. Katabi. Enabling high-quality untethered virtual reality. In *Proceedings of USENIX NSDI*, pages 531–544, 2017.

[12] C. Alvarez, J. Corbal, and M. Valero. Fuzzy memoization for floating-point multimedia applications. *IEEE Transactions on Computers*, 54(7):922–927, 2005.

[13] J.-M. Arnau, J.-M. Parcerisa, and P. Xekalakis. Eliminating redundant fragment shader executions on a mobile gpu via hardware memoization. In *Proceedings of the ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 529–540. IEEE, 2014.

[14] R. T. Azuma. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4):355–385, 1997.

[15] K. Boos, D. Chu, and E. Cuervo. Flashback: Immersive virtual reality on mobile devices via rendering memoization. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 291–304. ACM, 2016.

[16] K.-T. Chen, P. Huang, and C.-L. Lei. Game traffic analysis: An mmorpg perspective. *Computer Networks*, 50(16):3002–3023, 2006.

[17] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. CloneCloud: Elastic execution between mobile device and cloud. In *Proceedings of the 6th Conference on Computer Systems*, pages 301–314, 2011.

[18] 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 the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[19] 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 the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 121–135. ACM, 2015.

[20] 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.

[21] P. Dempsey. The teardown: Htc vive vr headset. *Engineering & Technology*, 11(7-8):80–81, 2016.

[22] 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.

[23] M. Dhome, M. Richetin, J.-T. Lapreste, and G. Rives. Determination of the attitude of 3d objects from a single perspective view. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(12):1265–1278, 1989.

[24] 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.

[25] W. Gao, Y. Li, H. Lu, T. Wang, and C. Liu. On exploiting dynamic execution patterns for workload offloading in mobile cloud applications. In *Proceedings of the IEEE 22nd International Conference on Network Protocols (ICNP)*, pages 1–12. IEEE, 2014.

[26] 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 the 13th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 137–150. ACM, 2015.

[27] 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*, pages 93–106, 2012.

[28] 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.

[29] S. Hong, J. Im, S. Islam, J. You, and Y. Park. Enabling energy efficient image encryption using approximate memoization. *Journal of Semiconductor Technology and Science*, 17(3):465–472, 2017.

[30] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen. Gaminganywhere: an open cloud gaming system. In *Proceedings of the 4th ACM multimedia systems conference*, pages 36–47. ACM, 2013.

[31] 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 the 18th International Workshop on Mobile Computing Systems and Applications (HotMobile)*, pages 61–66. ACM, 2017.

[32] 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 IEEE INFOCOM*, 2012.

[33] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, and N. Dai. Furion: Engineering high-quality immersive virtual reality on todays mobile devices. In *Proceedings of the 23rd International Conference on Mobile Computing and Networking (MobiCom). ACM, Snowbird, Utah, USA*, 2017.

[34] S. M. LaValle, A. Yershova, M. Katsev, and M. Antonov. Head tracking for the oculus rift. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 187–194. IEEE, 2014.

[35] 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 the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 151–165. ACM, 2015.

[36] Y. Li and W. Gao. Interconnecting heterogeneous devices in the personal mobile cloud. In *Proceedings of IEEE INFOCOM*. IEEE, 2017.

[37] Y. Li and W. Gao. Minimizing context migration in mobile code offload. *IEEE Transactions on Mobile Computing*, 16(4):1005–1018, 2017.

[38] R. LiKamWa and L. Zhong. Starfish: Efficient concurrency support for computer vision applications. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 213–226. ACM, 2015.

[39] H. Lu and W. Gao. Supporting real-time wireless traffic through a high-throughput side channel. In *Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 311–320. ACM, 2016.

[40] H. Lu and W. Gao. Continuous wireless link rates for internet of things. In *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 48–59. IEEE Press, 2018.

[41] H. Lu, W. Gao, et al. Scheduling dynamic wireless networks with limited operations. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pages 1–10. IEEE, 2016.

[42] 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.

[43] W. R. Mark, L. McMillan, and G. Bishop. Post-rendering 3d warping. In *Proceedings of the symposium on Interactive 3D graphics*. ACM, 1997.

[44] Z. Merchant, E. T. Goetz, L. Cifuentes, W. Keeney-Kennicutt, and T. J. Davis. Effectiveness of virtual reality-based instruction on students' learning outcomes in k-12 and higher education: A meta-analysis. *Computers & Education*, 70:29–40, 2014.

[45] D. Nehab, P. V. Sander, J. Lawrence, N. Tatarchuk, and J. R. Isidoro. Accelerating real-time shading with reverse reprojection caching. In *Graphics hardware*, volume 41, pages 61–62, 2007.

[46] N. Nguyen and T. D. Wilson. A head in virtual reality: Development of a dynamic head and neck model. *Anatomical sciences education*, 2(6):294–301, 2009.

[47] T.-i. Ohta, K. Maenobu, and T. Sakai. Obtaining surface orientation from texels under perspective projection. In *Proceedings of IJCAI*, volume 81, pages 746–751, 1981.

[48] 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 the 18th International Workshop on Mobile Computing Systems and Applications (HotMobile)*, pages 67–72. ACM, 2017.

[49] 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.

[50] D. Scherzer, L. Yang, O. Mattausch, D. Nehab, P. V. Sander, M. Wimmer, and E. Eisemann. Temporal coherence methods in real-time rendering. In *Computer Graphics Forum*, volume 31, pages 2378–2408, 2012.

[51] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui. Cloud gaming: architecture and performance. *IEEE network*, 27(4):16–21, 2013.

[52] P. Sitthi-amorn, J. Lawrence, L. Yang, P. V. Sander, D. Nehab, and J. Xi. Automated reprojection-based pixel shader optimization. *ACM Transactions on Graphics (TOG)*, 27(5):127, 2008.

[53] R. Skupin, Y. Sanchez, C. Hellge, and T. Schierl. Tile based hevc video for head mounted displays. In *Proceedings of the IEEE International Symposium on Multimedia (ISM)*, pages 399–400. IEEE, 2016.

[54] A. Sodani and G. S. Sohi. *Dynamic instruction reuse*, volume 25. ACM, 1997.

[55] 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 the IEEE International Symposium on Multimedia*, pages 583–586. IEEE, 2016.

[56] A. Suresh, E. Rohou, and A. Seznec. Compile-time function memoization. In *Proceedings of the 26th International Conference on Compiler Construction*, pages 45–54. ACM, 2017.

[57] A. Suresh, B. N. Swamy, E. Rohou, and A. Seznec. Intercepting functions for memoization: A case study using transcendental functions. *ACM Transactions on Architecture and Code Optimization (TACO)*, 12(2):18, 2015.

[58] L. Tong, Y. Li, and W. Gao. A hierarchical edge cloud architecture for mobile computing. In *Proceedings of IEEE INFOCOM*, pages 1–9. IEEE, 2016.

[59] T. Tsumura, I. Suzuki, Y. Ikeuchi, H. Matsuo, H. Nakashima, and Y. Nakashima. Design and evaluation of an auto-memoization processor. In *Parallel and Distributed Computing and Networks*, pages 230–235, 2007.

[60] C. Vázquez, W. J. Tam, and F. Speranza. Stereoscopic imaging: filling disoccluded areas in depth image-based rendering. In *Proc. SPIE*, 2006.

[61] I. Viola, A. Kanitsar, and M. E. Groller. Hardware-based nonlinear filtering and segmentation using high-level shading languages. In *Proceedings of the 14th IEEE Visualization Conference*, 2003.

[62] 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.

[63] 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.

[64] 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.

[65] H. Zhou, Y. Fu, and C. Liu. Supporting dynamic gpu computing result reuse in the cloud. In *Proceedings of ACM HotCloud*, 2015.