Déjà View: Spatio-Temporal Compute Reuse for Energy-Efficient 360° VR Video Streaming

Shulin Zhao, Haibo Zhang, Sandeepa Bhuyan, Cyan Subhra Mishra, Ziyu Ying,
Mahmut T. Kandemir, Anand Sivasubramaniam, Chita R. Das
Dept. of Computer Science and Engineering, The Pennsylvania State University
Email: {suz53, huz123, sxb392, cyan, ziy5087, mtk2, axs53, cxd12}@psu.edu

***Abstract*—The emergence of virtual reality (VR) and augmented reality (AR) has revolutionized our lives by enabling a 360° artificial sensory stimulation across diverse domains, including, but not limited to, sports, media, healthcare, and gaming. Unlike the conventional planar video processing, where memory access is the main bottleneck, in 360° VR videos the compute is the primary bottleneck and contributes to more than 50% energy consumption in battery-operated VR headsets. Thus, improving the computational efficiency of the video processing pipeline in a VR is critical. While prior efforts have attempted to address this problem through acceleration using a GPU or FPGA, none of them has analyzed the 360° VR pipeline to examine if there is any scope to optimize the computation with known techniques such as memoization.**

Thus, in this paper, we analyze the VR computation pipeline and observe that there is significant scope to skip computations by leveraging the temporal and spatial locality in head orientation and eye correlations, respectively, resulting in computation reduction and energy efficiency. The proposed *Déjà View* design takes advantage of temporal reuse by memoizing head orientation and spatial reuse by establishing a relationship between left and right eye projection, and can be implemented either on a GPU or an FPGA. We propose both software modifications for existing compute pipeline and microarchitectural additions for further enhancement. We evaluate our design by implementing the software enhancements on an NVIDIA Jetson TX2 GPU board and our microarchitectural additions on a Xilinx Zynq-7000 FPGA model using five video workloads. Experimental results show that *Déjà View* can provide 34% computation reduction and 17% energy saving, compared to the state-of-the-art design.

***Index Terms*—Virtual Reality, Edge Computing, IoT, 360° Video Processing**

I. Introduction

Recent developments in technology, computing and communication have brought significant changes to the lifestyle of common people by providing them access to increasingly sophisticated devices. Especially, VR and AR are now gaining traction because of their versatile nature of providing an immersive sensory experience, which is not possible with the conventional systems – especially in the domain of video streaming. They are emerging as one of the most important entertainment markets and Goldman Sachs predicts that, by 2025, around 79 million users will use online video streaming from the VR/AR ecosystem, resulting in a multi-billion dollar market [20], penetrating the fields of media streaming, VR gaming, education, medicine, communication and many more. Even today, more than 10 million users enjoy 360° videos using Google Cardboard [10], Samsung Gear VR [44], and Oculus VR [8], to experience 360° video [7], art museum [9], live stadium [46], etc.

The 360° videos are created by capturing scenes in all directions typically using omnidirectional cameras or a set of cameras. They are further encoded by the conventional video encoders, as if they are planar videos, for transmission efficiency. The video frames are transmitted to the users, who wear a portable VR headset (like Facebook Oculus or Google Cardboard), via *Youtube* or *Facebook 360* services [7], [61]. 360° video streaming creates an interactive and immersive environment by connecting the user and the video content; the users are allowed to move their heads' orientation to enjoy the surroundings in all perspectives along with a 3D view, i.e., a different view for each of the eyes, and hence creating an illusion that the user is present at the scene rather than viewing it on a projected surface.

This immersive experience comes at the cost of additional computations - not only is the video being streamed, *the streaming itself changes with the head orientation.* Moreover, streaming requires two projections for both the eyes. As the 360° video is not in a planar format, the VR ecosystem converts it to a conformal 2D format by passing it through multiple stages of transformations. Thus, unlike planar videos, in 360° videos, specifically the projection computations for capturing the head movements and eye correlations, are significantly computation-heavy, amounting to 59% of the overall VR (headset) power budget. Current head mounted VR devices use a GPU for this heavy computation. Since the head mounted VR devices are battery-backed, the computations that draw high power from the battery greatly hinder the experience of watching long 360° videos [39].

This heavy computation has become an acceleration candidate/target in previous works, by offloading the entire computation, as is, to an accelerator (GPU [39], or FPGA [28]). However, prior works do not consider other avenues for optimizing the computation. In this context, this paper dives deep to understand the projection computation pipeline for exploring available opportunities and optimizations for speedup as well as power savings. Since *head movement* and *correlations between the left and right eye projections* are the two critical components of the projection computation, we analyze and study them to explore possible opportunities to exploit these relations. Specifically, we analyze four scenarios,

978-1-7281-4661-4/20/$31.00 ©2020 IEEE
DOI 10.1109/ISCA45697.2020.00030

namely, *InterFrame-IntraEye (EA)*, *IntraFrame-InterEye (AE)*, *IntraFrame-IntraEye (AA)*, and *InterFrame-InterEye (EE)*, that are critical in capturing the head movement and eye correlation for projection computation. Out of these four scenarios, we observe that *EA* computation for head orientation can be exploited for *temporal reuse/memoization* since there is little difference between two previous head orientations, and *AE* computation for exploiting the correlation between both the eyes by *spatial reuse – correlating the coordinate relationship between both eyes*. Based on this observation, we develop computation optimization mechanisms for facilitating temporal reuse/memoization and spatial reuse that can be integrated with a VR projection computation pipeline to significantly reduce energy consumption of the device. The proposed architecture is named **Déjà View**, a play on the word Déjà vu, as it uses previous or *already seen* views.

To the best of our knowledge, this is the first work that leverages head orientation and correlation between eyes to do efficient memoization and in turn result in compute reduction in the VR video streaming domain. The major **contributions** of the paper can be summarized as follows:

- From an open-source 360° VR video dataset [3], *we identify both temporal reuse and spatial locality that exists in user behavior*. We formally analyze the potential "input invariability" in the *projection computation* during 360° video streaming, which manifests in the head movement locality (temporal reuse) and the stationarity relationship between two eyes (spatial reuse). Such invariances are leveraged as reuse opportunities to reduce the compute-heavy projection computation.
- *We design two complementary schemes to capture both temporal and spatial reuse opportunities.* We propose a memoization scheme, called *EA*, to capture recent head orientation data for temporal reuse, and for the spatial reuse, we design the *AE* scheme, which leverages the stationary relationship between two eyes to efficiently reduce the amount of projection computation.
- We implement both our schemes as a *software* enhancement to the existing compute pipeline in NVIDIA GPUs. To further exploit the energy efficiency, we also implement our *hardware* prototype using an FPGA to evaluate the energy benefits brought by the microarchitectural augmentations. Both the proposed software and hardware solutions are *modular*, and hence can be integrated to the existing pipeline with little change.
- *We evaluate our integrated design, including both* EA *and* AE*, using an open-source 360° VR video dataset [3] with the traces of 20 users watching 5 different VR videos.* Overall, our experimental results show that, on an average, **Déjà View** can provide 54% compute reduction, which translates to 28% total energy savings compared to the baseline setup. Compared to a state-of-the-art scheme [28], our design provides 34% reduction in projection computations, which translates to 17% additional energy savings.

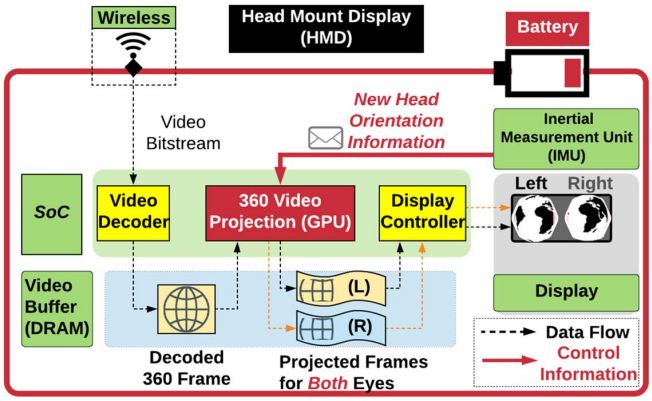


Fig. 1: A 360° video processing pipeline on a battery-backed stereoscopic HMD with an Inertial Measurement Unit (IMU) and an SoC equipped with a GPU [28], [39].

II. Background and Motivation

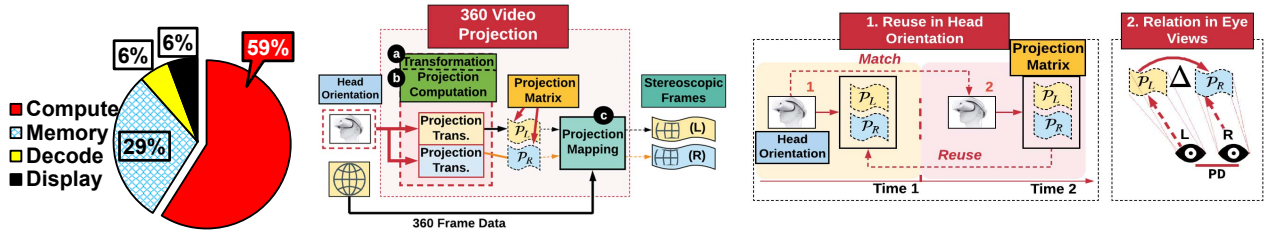
Before getting into the details of the existing issues and possible solutions, we first outline the computation pipeline of the state-of-the-art 360° VR streaming (Fig. 1). Further, we describe the existing energy inefficiencies in processing 360° VR systems, to motivate our design for mitigating the computational inefficiencies by avoiding redundant computations.

A. 360° Video Streaming Pipeline

The key *difference* between a 360° VR video compared to a conventional 2D video is that the former provides content-rich immersive user experience. Wearing a head mounted display (HMD), a user navigates in a virtual world by *looking around*, or *moving around* [55], to interact with the virtual world. As shown in Fig. 1, a typical VR HMD [39] has two major components: (i) an SoC with a video decoder, a GPU for processing projection computation, and a display controller, and (ii) a video buffer in DRAM for storing the decoded 360° frames and projected frames for both the left and right eyes. More specifically, the 360° video processing pipeline can be summarized as follows:

Video Decoder: The HMD receives encoded 360° video bitstream from the network (YouTube [61], Facebook-360 [7], etc). Similar to 2D videos, the 360° video bitstreams are encoded in H.264/MPEG formats [19] for network efficiency. The next step is to decode the original frame from the bitstream, and today, this is mostly done using a hardware-based h264/MPEG decoder for more energy efficiency. After decoding, the 360° output frames are then buffered in the video buffer, waiting to be rendered.

Projection: Note that, the output frames from the decoder are still in the *spherical coordinate system*. This is because, for encoding purpose, the original 360° videos are projected into the 2D plane (usually represented in 2D format such as Equirectangular [52], Cubemap [41], etc.). Therefore, unlike the 2D video processing where the display can directly read



(a) Power breakdown. **(b) Overview of 360° video projection.** **(c) Head movement and pupillary distance as inputs.**

Fig. 2: Overview of 360° video projection. (a) Power breakdown consuming 3.4 Watts; (b) Projection pipeline taking head orientation and pupillary distance to compute projection matrices for both the eyes, which map 360° coordinates to 2D coordinates for generating stereoscopic frames. (c) Reusing projection matrices by exploiting relation between both eyes and fusing it with head orientation.

the video buffer to present the *decoded* frames, the 360° video frames require "additional rendering effort" to get displayed. More specifically, the rendering process is a projection from the 360° frame pixels' 3D coordinates to the 2D frame pixels' 2D coordinates on HMDs. The projection process considers two user-side aspects – *head orientation and pupillary distance*[1] – to render stereoscopic views or Field of View (FoV) frames for both eyes, towards the head direction. The head orientation is sensed by an inertial measurement unit (IMU) on the HMD as a triple [Yaw, $Pitch$, $Roll$] for projection computation[2]. For each frame, this computation is processed twice – one for left eye and the other for right eye – to reinforce users' sense of depth.

Display: After the projection, the two generated FoV frames are stored in 2D format in the video buffer. The display controller just needs to read them from DRAM to the screen.

To summarize, compared to 2D video processing, 360° video processing incurs additional projection computation. From our measurements collected from a smartphone [53] running a 360° VR application [11], even with extra computation, the overall processing for rendering one 360° frame can be completed within 22 ms on average (translating to 45 fps). However, since the whole computation/rendering process takes place on a battery-backed device [39], one needs to consider the "energy efficiency" of this computation, i.e., even though we can meet the performance requirements of such video, energy efficiency needs to be improved.

B. Motivation

To understand the energy profile in the current VR devices, we characterize the energy consumption of 360° video processing on a prototype [36] (configured similar to a commercial VR device [39], discussed in Sec. V) in Fig. 2a. Overall, 360° VR video processing consumes 3.4 $Watts$, which is 2.27× the power compared to its planar counterparts (1.5 $Watts$). We also observe that, unlike conventional planar video processing where *memory* is the main bottleneck (43%), in 360° VR video processing, ***compute*** dominates the power consumption, constituting 59%. Previous studies have observed that the computation in 360° video processing is, mainly, the *projection transformation* [28]. These observations motivate us to explore the potential opportunities for reducing the power/energy consumption in the *projection* stage.

We illustrate the 360° video projection/projection transformation[3] computation in Fig. 2b. At a high level, we need two major inputs to generate the final projected frames on the display. The first input is from the user side, including the head orientation and pupillary distance. Since there is a small offset between the two eyes, the projection computation needs to capture the pupillary distance to generate a separate view for each eye. Therefore, the output of the projection computation is *two* projection matrices, each indicating the mapping between each coordinate in 360° video frame and a 2D coordinate on screen for the left and right eye. Note that, this projection process is quite compute-intensive. Furthermore, in a typical VR headset, the above computation needs to repeat millions of times, for processing just one 360° frame. Our characterization indicates that, on average, around 2.3 GFLOPS is required for this projection transformation (details are discussed in Sec. III).

The second input is the decoded 360° frame that contains the pixel values. The decoded 360° frame is fed to the Projection Mapping stage, which uses the projection matrix, locates the coordinates in the decoded 360° frame, and moves their pixel values to the transformed 2D coordinates in the FoV frames. It is to be noted that, when a user's head orientation is changed, the Projection Computation stage needs to *recompute* the transformations to reflect the user's head movement.

The VR headset allows users to freely move their heads and eyes at any time to any degree. Hence, the projection transformation computation has to be executed at every frame to reflect user movements in real-time, and the whole process is very compute intensive (36 times per second [3]) and power hungry. However, it is too conservative to execute the projection transformation in a short period of time even for the same set of inputs. Intuitively, if the inputs of the transformation computation do not change, the output of the transformation will also be same. In fact, we observe the

[1]Pupillary distance is the distance, typically measured in millimeters, between the centers of the pupils of the eyes.

[2]Yaw: vertical; Pitch: side-to-side; Roll: front-to-back.

[3]We use "projection transformation" and "360° video projection" interchangeably.

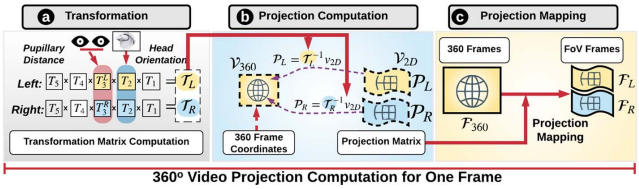


Fig. 3: Detailed illustration of projection transformation. The projection transformation first calculates the transformation matrix ($\mathcal{T}$), and then uses the transformation matrix to map each of the pixel coordinates to generate the projection matrices ($\mathcal{P}$) for the FoV frames. Then projection mapping stage uses this coordinate mapping ($\mathcal{P}$) to move pixels from the original 360 frame $\mathcal{F}_{360}$ to the 2D frame $\mathcal{F}$.

following two properties from a published 360° VR dataset [3], as shown in Fig. 2c:

Head Orientation Proximity: In a short period of time, the user's head orientation is usually stable in a small space range (3D) or even still. In fact, from this dataset we have found that, the head orientation for users does not often change within around $150ms$ period of time. Furthermore, even in cases where head orientation changes, the change is usually within a small range – in a few consecutive frames. Since two identical head orientations lead to the same projection matrices, one opportunity to reduce computation is to *memoize* a set of head orientations as well as their corresponding compute results.

Vision Proximity: It is to be emphasized that, even when the head orientation input for two computations are the same, the two eye coordinates can be different. Because of this, in current designs, the projection transformation is invoked *twice* as it needs to generate two different transformation matrices for the left eye and the right eye. On the other hand, the distance between the two eyes is small and is constant for a particular user[4]. The two transformation matrices are very "similar" as they inherit a relationship between them as a function of the small pupillary distance.

Motivated by these observations, in the following sections, we explore and address two critical questions: *Can we identify the proximity in the projection computation?*, and *Can we leverage this proximity to safely skip some computations to save energy?*

III. 360° Video Projection

To leverage the opportunities in the 360° video projection, we need to understand the execution of the entire projection processing in a 360° VR system. We illustrate the details of 360° video projection in Fig. 3 as three stages (detailed background of this projection transformation can be found in [21], [27]). The first stage, *Transformation* (denoted ⓐ in Fig. 3, is to determine a transformation matrix by combining five different transforms. The second stage, *Projection Computation* (denoted ⓑ in Fig. 3), uses the transformation matrix and the 2D FoV coordinates for both eyes to obtain their corresponding 360° video frame coordinates. Finally, the third stage, i.e., *Projection Mapping*, uses the mapping results from the second stage and the 360° video frame to deduce the pixel values for 2D FoV frames (shown in ⓒ in Fig. 3) ,which can be projected to both eyes on the HMD.

[4]We used an averaged pupillary distance in our evaluations [27], [37].

TABLE I: Projection Computation description.

Label	Description	HO dependent?	Known-time	Eye-dependent?
T_1	Rigid body	No	Compile-time	Left = Right
T_2	An eye's view	Yes	Runtime	Left = Right
T_3	Eye adjusting	No	Compile-time	Left $\neq$ Right
T_4	Perspective	No	Design-time	Left = Right
T_5	Viewport	No	Design-time	Left = Right

The **Transformation** stage, shown in Fig. 3ⓐ for computation of the *Transformation Matrix*, is used for projecting the 360° frame pixels onto the $2D$ FoV plane in the subsequent stages. This matrix is calculated by applying five different transforms – T_1, T_2, T_3, T_4, and T_5 – in a serial fashion.

- T_1 serves as a *rigid body transformation* matrix which applies 3D rotation ($Yaw, Pitch, Roll$) and translation so that, the objects do not get distorted. Since this transformation does not depend on any of the sensor inputs, it can be pre-calculated at compile-time.
- T_2 gives us *eyes' view*; i.e., this changes the virtual world's coordinate frame to match the frame of the eye. This requires knowledge of the head orientation or the direction of gaze, which can be read at runtime from the IMU sensors embedded in the VR headset.
- T_3 transforms the 360° coordinates from a *monocular view* to a *stereoscopic view*. Since each eye sees the same object *differently*, this transformation matrix is different for each eye to give the user a more realistic experience.
- T_4, also known as the *perspective transformation* matrix, maps all 360° coordinates onto 2D coordinates. This transformation depends only on the HMD characteristics, including, but not limited to, the display size and resolution, and hence, is known apriori (at design-time).
- T_5, the last transformation to be applied, performs a viewport transformation[5], bringing the projected points to the coordinates used to index the pixels on the HMD. As in the case of T_4, this transformation is also HMD design-dependent and is known at design-time.

Note that, the *product* of these five transforms gives us the final transformation matrices ($\mathcal{T}_L$ and $\mathcal{T}_R$), which together convert the $3D$ coordinates of the 360° frame to the $2D$ coordinates suitable for HMD. Mathematically, the transformation matrix for each eye is shown in Equation 1.

$$\begin{aligned} \mathcal{T}_L &= T_5 \times T_4 \times T_3^L \times T_2 \times T_1 \\ \mathcal{T}_R &= T_5 \times T_4 \times T_3^R \times T_2 \times T_1 \end{aligned} \quad (1)$$

These five transforms are of dimension of 4×4 (3 dimensions for rotation; 1 for translation), thus producing 4×4 $\mathcal{T}_L$ and $\mathcal{T}_R$ matrices [27]. Note that, given an arbitrary FoV frame, these transformation matrices remain the same for all the pixel

[5]A Viewport Transformation is the process of transforming a 2D coordinate objects to device coordinates [27].

coordinates in that frame, thus are evaluated only *once* for that frame, and account for only $4.8MFLOPS$ without any optimization.

In the **Projection Computation** stage (refer to ⓑ in Fig. 3), we use the transformation matrix ($\mathcal{T}$) to generate the mapping ($\mathcal{P}$) between the 360° frame coordinates and the $2D$ FoV frame coordinates. At any instance, a user is only concerned about the FoV pixels in the entire 360° frame. So, instead of evaluating the mapping for all coordinates in the 360° frame, we only generate the mapping for those pixels which are within the user's view. As the target $2D$ FoV coordinates are already known (VR screen dimensions), these mappings can be performed by multiplying the inverse of the transformation matrix ($\mathcal{T}^{-1}$) with the $2D$ FoV coordinates ($\mathcal{V}_{2D}$), thus generating the corresponding 360° pixel coordinates ($\mathcal{P}$), as shown in Equation 2.

$$\begin{aligned} \mathcal{P}_L^i &= \mathcal{T}_L^{-1} \times \mathcal{V}_{2D}^i; \quad \forall\, i \leq num\_pixels \\ \mathcal{P}_R^i &= \mathcal{T}_R^{-1} \times \mathcal{V}_{2D}^i; \quad \forall\, i \leq num\_pixels \end{aligned} \tag{2}$$

Here, $\mathcal{V}_{2D} = [q_0, q_1, q_2, q_3]^\top$ represents the quaternion equivalent of the $2D$ FoV coordinates used for matrix multiplication with the inverse transformation matrix ($\mathcal{T}^{-1}$). Note that, this operation, which is a matrix multiplication on each FoV pixel coordinate, can be quite compute intensive. In fact, the number of pixels in the FoV is usually around 1 million, and the videos stream at a rate of 30 fps for an immersive experience. This amounts to about *2.3 GFLOPS*, which represents a substantial amount of computation, given the limited compute capabilities and power in such edge devices. Note that, even though theoretically one represents the 360° frame coordinates as quaternions, in practice, they are typically represented using specific projection formats, e.g., equirectangular, cube map, equi-angular cubemap, pyramid format, etc. The details of these formats are in the purview of cartography and computer graphics domain, and hence we do not evaluate all of the aforementioned formats. In our evaluations and experiments, we used the equirectangular format [52], which is one of the most popular projection formats.

The **Projection Mapping** stage (ⓒ in Fig. 3) takes the projection matrices for both the eyes ($\mathcal{P}_L, \mathcal{P}_R$) of Equation 2 as well as the pixel values of the 360° video frame ($\mathcal{F}_{360}$), to obtain the $2D$ FoV frames ($\mathcal{F}_L$ and $\mathcal{F}_R$), which can be further displayed on the HMD. This stage mostly comprises of memory operations, and thus is not a compute bottleneck.

Our discussion in this section is summarized in Tab. I. We have two important takeaways:

- **Computation Dependence Chain:** We note that there exists a data dependence from the 360° frame to generate the final FoV frame, where $\mathcal{F}$ depends on $\mathcal{P}$, which in turn depends on $\mathcal{T}$. This also determines the "order of computation", which is first $\mathcal{T}$, then $\mathcal{P}$, and finally $\mathcal{F}$.
- **Input (in-)Variability:** It should be clear from the discussion above that T_1, T_3 T_4, and T_5 can be determined apriori. However, T_2 can change at runtime, and if any element in T_2 is changed, the transformation matrix needs re-computation along with $\mathcal{P}$ and $\mathcal{F}$, due to their dependencies. However, if T_2 does not change across frames, $\mathcal{P}$ is identical to the previous frame.

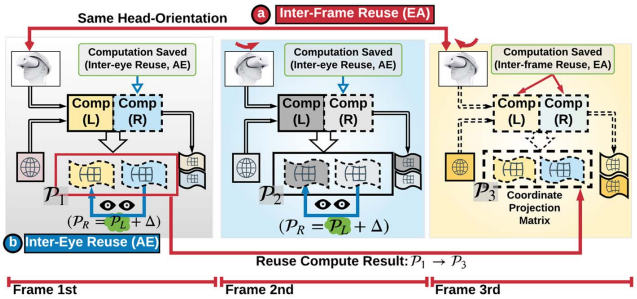


Fig. 4: InterFrame-IntraEye (*EA*) and IntraFrame-InterEye (*AE*) reuse opportunities. This example illustrates 3 consecutive frames processing, each of which consists of two projection matrices ($\mathcal{P}_L$ and $\mathcal{P}_R$) for both eyes. The 3rd frame shares the same head orientation with the 1st, thus can be optimized by *EA*. Moreover, the reuse between both eyes is further optimized by *AE*.

IV. Reducing Projection Computation

As discussed in Sec. II, computations dominate the energy consumption in 360° VR video processing. Further, we also observed in Sec. III that, the main reason behind this is that the compute is executed repeatedly both within a single frame (due to offset between the eyes) and across frames (due to changes in head orientation at runtime). Unlike prior works targeting at optimizing the efficiency of *each computation* [28], [57], we primarily focus on reducing *the amount of computation* to be performed, by exploring the intrinsic "compute reuse opportunities" in 360° VR video processing.

A. Opportunities

Exploring and exploiting computation output reuse opportunities is non-trivial in this context. First of all, the projection transformation is multi-staged and is a composition of multiple mathematical operations, e.g., transformation matrix, projection computation, mapping, etc. As discussed in Sec. III, the projection computation varies across **eyes** even for the same head orientation. Moreover, in many cases, computations are also sensor input-dependent such as the IMU data for determining the head orientation, which is updated across **frames**, at runtime. Thus, to explore computation reuse opportunities, we start by distinguishing between 4 complementary opportunities – *InterFrame-IntraEye (EA)*, *IntraFrame-InterEye (AE)*, *IntraFrame-IntraEye (AA)*, and *InterFrame-InterEye (EE)*, using a represent example shown in Fig. 4.

- In *EA*, as discussed in Sec. III, the transformation matrix ($\mathcal{T}$) is determined by the head orientation, which is sampled from the built-in IMU sensors. We observe that, if the head orientation does not change across two frames, the five transforms and the 360° coordinate inputs remain the same, thereby providing ample opportunities for directly *reusing*

the compute results from the previous frame ($\mathcal{P}_1$), as shown in ⓐ in Fig. 4.

- *AE* comes to play when there is a change in head orientation in consecutive frames, and we cannot enjoy the opportunities in *EA*. For such scenarios, due to the prevailing relationship between the left and right eye transformation matrices ($\mathcal{T}_L$ and $\mathcal{T}_R$), we can further avail the spatial compute reuse opportunity shown in ⓑ in Fig. 4, by reconstructing the computation needed for one eye ($\mathcal{P}_R$) from the other ($\mathcal{P}_L$).
- In *AA*, the input and output mapping are unique, that is, no two input coordinates in 360° frame map to the same coordinates in the 2D FoV frame, thereby eliminating any compute reuse scope. Although, in principle, for computing the transformation for consecutive pixels, one can leverage data value similarity to reduce the computation, in this work we are not focusing on leveraging any such opportunity.
- *EE* offers little chance of reuse, and can only be leveraged in rare occasions, where we have oracular knowledge of head movements. Furthermore, in such cases of head movement, there is likely to be some reuse from inter-eye reusability within a frame, rather than inter-frame reusability.

B. Why have EA/AE Opportunities been previously Ignored?

Based on the above discussion, in this work, we focus on *EA* and *AE* opportunities. We are unaware of any existing implementation or research work that focus on compute reuse by leveraging across-frames and across-eyes memoization. In fact, the existing state-of-the-art software stack, such as GoogleVR-SDK [11], simply uses the IMU sensor inputs to calculate the updated transformation matrices, then passes them to the OpenGL [42] engine to process the projection computation, as shown in Fig. 2b. We would also like to point that capturing these opportunities is not trivial and cannot be efficiently done by just optimizing the existing application layer and software stack. We describe the underlying issues to address and emphasize the non-trivialities:

- To ease development efforts, state-of-the-art VR applications reuse APIs provided by OpenGL [24], [42], and whenever a new frame is decoded, they always invoke the `glDrawFrame` twice for both eyes (see line number 257 in googlevr-video360 application [12]). They do not seem to leverage the fact that the transformation matrices are unique for each head orientation and memoizing them will save re-calculating the transformation matrix ($\mathcal{T}$) as well as the projection matrix ($\mathcal{P}$).
- Even if they do realize such opportunities, the projection matrix ($\mathcal{P}$) is very big ($\approx 8MB$, details in Sec. IV-C), and one edge VR headset cannot afford to memoize them for *all* possible head orientations. To address this, we need to study the impact of head orientation on the computation and whether we can establish a relationship between the computation executed for both the eyes to get rid of any existing redundancies. All these are possible avenues for optimization and demand a detailed study of the computation pipeline, the workloads, user behavior, etc., to find a way to further improve the state-of-the-art.
- Furthermore, a software-only approach may not give us the desired solution as some of these additional execution cycles, control and data path manipulations may need architectural support, especially to reduce memory and power overheads on edge VRs. Therefore, we believe that, achieving benefits by exploiting the *EA* and *AE* opportunities needs an extensive study and a careful design, especially from an architectural perspective, to maximize the benefits.

Driven by the above discussion and the potential optimization opportunities presented by *EA* and *AE*, we propose **Déjà View**, an energy-efficient design for 360° video streaming on VRs. As shown in Fig. 4, **Déjà View** leverages compute locality to bypass computations and provides significant energy savings, with the following two-step optimization strategy:
ⓐ For each frame, if the head orientation remains the same, we take advantage of the *EA* opportunity.
ⓑ If exploiting the *EA* opportunity is not possible, we take advantage of the *AE* opportunity, by performing computation for only one eye (and construct the result for the other eye).

C. InterFrame-IntraEye (EA) Computation Optimization

We plan to leverage the *EA* opportunity when the user's head orientation does not change. Intuitively, as mentioned earlier in Sec. III (Fig. 3), ⓐ Transformation and ⓑ Projection Computation remain unchanged. To understand all the contributing factors which affect computations, we further investigate the important inputs of the VR headset. This can help us identify and isolate proper memoization candidates for carefully tweaking our design decisions to maximize the reuse benefits. Further, we also study the overheads introduced by our design modifications to perform a fair comparison with the state-of-the-art.

What (features) to Memoize? As discussed earlier, at any moment during VR video processing, the execution pipeline is not only impacted by the head orientation, but also by other features such as video frame rate, video types/semantics information, pixel values, user interactions. To better understand which of these are the best candidates (*features*, using machine learning parlance) for memoization and whether they are sufficient or not, we next discuss input parameters and their impact on the computation:

- *Head orientation:* Any changes in this affect the matrix T_2 as discussed in Tab. I, thus changing the transformation matrix $\mathcal{T}$ and eventually leading to *re-computation* of the most compute-intensive projection matrix $\mathcal{P}$. Thus, it is a critical feature in projection computation executions.
- *Pixel values:* The pixel contents/values (denoted as $\mathcal{F}$ in Fig. 3) matter only during data transfer (from the input 360° frame to the framebuffer) in the projection mapping stage, after the coordinate mappings ($\mathcal{P}$ in Fig. 3) are generated. Potentially, content-based optimizations (e.g., content cache [63]) can benefit the data transfer; however, they are not attractive candidates to leverage compute reuse, which is the major power-hungry stage (as shown in Fig. 2a). In

TABLE II: Video workloads.

No.	Video	Type (Cam movement/focus of attention direction)	Frame Rate (fps)	#Frames	Bit Rate (kbps)
V1	Rhinos [4]	Stationary cam, no focus direction	30	3280	13462
V2	Timelapse [56]	Stationary cam, fast-moving objects, no focus direction	30	2730	15581
V3	Rollercoaster [35]	Fast-moving cam hooked in front of a rollercoaster, uni-direction focus	29.97	6194	16075
V4	Paris [51]	Stationary cam, smooth scene cuts, no focus direction	59.94	14629	14268
V5	Elephants [5]	Stationary cam, uni-direction focus	30	5510	16522

this work, we are focusing on reusing computation results rather than reducing the content maintenance/transfer, and hence do not consider that optimization.

- *Video meta-information:* This contains the additional information, such as frame rates, video semantics/types, etc., about the video inputs. This feature can only be used as an add-on, along with other inputs to further improve compute reuse scope. For example, if the $360°$ video frame rate increases from the typical 30 fps to 60 fps, then one can potentially leverage this enhanced compute frequency in conjunction with the head orientation, to further expand the compute reuse window. Note however that, this meta-information is not on the data-dependence chain, and we do not consider it for memoization.

To summarize, among the above discussed features, we identify *head orientation* as the only suitable memoization candidate for boosting the compute reuse scope. Thus, we memoize both head orientation and its corresponding projection matrix (i.e., projection computation results) in a memory buffer, namely, $\mathbb{P}_{buff}$, and use the head orientation to index the address/pointer of that $\mathbb{P}_{buff}$ stored in DRAM.

How Much to Memoize? The occupied DRAM size is mainly determined by $\mathbb{P}_{buff}$. In fact, with a VR screen size of $1,000\times 1,000$, one $\mathbb{P}_{buff}$ occupies $\approx 8MB$ in DRAM. Since this puts a high demand on memory, one edge VR headset cannot afford to memoize for all possible head orientations. Thus, we want to limit the number of $\mathbb{P}_{buff}$ that we need to store.

To address this, we need to carefully decide how much history is to be memoized for leveraging computation reuse. We performed a study on the VR video dataset [3] to investigate the head orientation traces of 20 users watching 5 widely-variant $360°$ VR videos (summarized in Tab. II). Typically, the resolution of the IMU traces can be as high as 20 bits per field [3], [28]. From the dataset, we report the average reuse distance, i.e., the average number of preceding frames with same head orientation to be memoized, and show it in Fig. 5a. It can be concluded from these results that, memoizing the last **two** frames is sufficient for most of the cases. Memoizing more frames may not bring much additional benefits because of the high sensitivity of the IMU sensors. Storing only *two* head orientations (in registers) and their associated $\mathbb{P}_{buff}$ in the DRAM occupies only $\approx 16MB$ memory space.

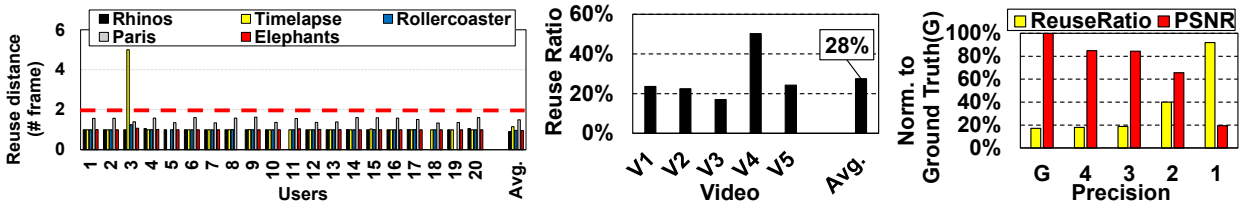
Further, we also observe that, the duration for which the head orientation does not change for three consecutive frames sums up to only $\approx 28\%$ of the video runtime on average (refer to Fig. 5b), limiting the memoization opportunities to those instances. Such low reuse ratio is expected because of the high sensitivity of the IMU sensors. However, a higher reuse ratio can be achieved by relaxing the precision of the IMU output. Furthermore, we study the `V3` (i.e., `Rollercoaster`) video and examine the trade-offs between (i) quantizing/approximating the head orientation (thus compromising video quality) with more reuse, vs. (ii) maintaining the lossless video quality but with a lower reuse ratio, in Fig. 5c, to provide an intuitive comparison in different scenarios. Here, we quantify the video quality with the popular Peak Signal-to-Noise Ratio (PSNR, normalized to the ground-truth; the higher, the better) [25], [40]. From Fig. 5c, we can observe that, as the precision decreases from 4 (resolution is 0.0001) to 1 (resolution is 0.1), the reuse ratio increases from 18% to 92%; however, the PSNR drops from 85% to only 19%. This is because low precision leads to a mis-projection, which fails to reflect the current head orientation. In this work, we do not want to distort video quality and thus explore the ground-truth only.

The Effect of EA: With this *EA* memoization, once a new head orientation is received, we first search it in the two head orientation registers. If there is a match, the associated $\mathbb{P}_{buff}$ will return the memory address of the saved $\mathcal{P}$ so that we can reuse $\mathcal{P}$ and skip the *entire* coordinate projection computation (refer to ⓐ in Fig. 4), with only 1% overhead w.r.t. baseline. If not, we have to execute the entire computation as in the baseline case. As a result, by exploiting the *EA* scheme on the second frame, its compute energy consumption can be reduced to only 1% of that consumed by *Baseline*.

D. IntraFrame-InterEye (AE) Computation Optimization

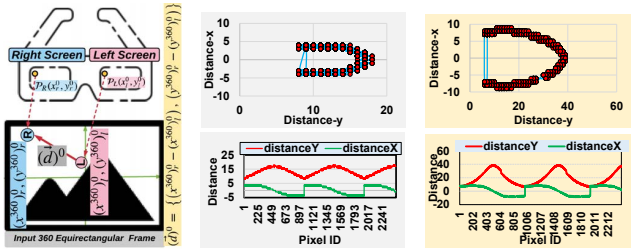
In *EA*, the compute can be bypassed by reusing the pre-computed results, if the head orientation matches with any of the two previously memoized head orientations (stored in registers). However, we also note that these opportunities might be limited owing to the "non-repetitive" user behavior. Despite this variation, there may still be matches/recomputations within a frame between two eyes, i.e., IntraFrame-InterEye as shown in ⓑ in Fig. 4. To leverage this opportunity, we next study the coordinate projection results relationship between left-eye and right-eye. If there exists a simple mechanism to describe the difference between the two projection matrices of the two eyes ($\mathcal{P}_L$ and $\mathcal{P}_R$), one can simplify the computation from matrix *multiplications* to matrix *additions*.

Distance Vector Study: Let us further look into the detailed mapping of a $360°$ frame (in equirectangular format) onto a $2D$ FoV frame in the Projection Mapping stage (refer ❻ in Fig. 3), at a pixel granularity. The pixel rendered at $[x_l^0, y_l^0]$ on the left VR screen is mapped from position $[(x^{360})_l^0, (y^{360})_l^0]$ on the equirectangular $360°$ frame, as shown in Fig. 6a. Similarly, the pixel value rendered at $[x_r^0, y_r^0]$ on the right VR screen



(a) Reuse distance is just 2 for most of cases (b) Reuse ratio (c) Precision vs. reuse ratio tradeoffs

Fig. 5: In *EA*, (a) shows that, on average, how many frame(s) from the current frame to a previous one with the same head orientation, as denoted as reuse distance. This indicates two memoization buffers are sufficient. (b) plots among all the head orientations, how many can be memoized by these two buffers. (c) illustrates the trade-off between the precision level and reuse ratio.



(a) Distance vector. (b) (0.00, 1.57, -0.73) (c) (0.52, 1.05, -0.73)

Fig. 6: In *AE*, distance vector (a) patterns with two different head orientations $(Yaw, Pitch, Roll)$ in (b) and (c).

is mapped from position $[(x^{360})^0_r, (y^{360})^0_r]$ on the 360° frame. To study the relationship between the coordinate projection results between both eyes, for the **same** two coordinates $[x^i, y^i]$ on both $2D$ FoV frames, we determine the distance vector $(\vec{d})^i$ between their equirectangular counterparts, represented in Equation 3:

$$(\vec{d})^i = [(x^{360})^i_r - (x^{360})^i_l, (y^{360})^i_r - (y^{360})^i_l]^\top \quad (3)$$

The knowledge of distance vector $\vec{d}$ is critical to explore the *AE* opportunity, because if it was known apriori, then we only need to process the entire projection transformation to generate the mapping results $(\mathcal{P}_L)$ for one eye, and then deduce the coordinate projection computation results for the other $(\mathcal{P}_R)$ by simply adding $\vec{d}$ with $\mathcal{P}_L$. This encourages us to further study whether this distance vector changes with head orientation or not, and also whether it is invariant for any particular frame – if yes, then how? To investigate these questions, we examine how the distance vector varies within a same frame, with two different head orientations, shown in Fig. 6b and Fig. 6c. On plotting the distance vectors for each row of the FoV frames, we observe a recurring *ellipse* pattern. The intuitive reason behind this *ellipse* pattern is related to the built-in features of the equirectangular format. Second, for different head orientations, their distance vectors plotted in Fig. 6b and Fig. 6c retain the same ellipse behavior but different shapes. Furthermore, by exacting the x (or y) coordinate in the distance vector, the above *ellipse* pattern can be represented as $\Delta x = a \cdot cos(\theta)$ and $\Delta y = b \cdot sin(\theta) + c$, where $\theta \in [0, \pi]$, and *a, b, c* vary with head orientation change but remain same for each row in the same frame. Additionally, there are few pixel positions at the frame edges which can only be viewed by one eye (denoted as *exclusive*), which cannot be captured by the above pattern. These pixels amount to only 2.7% of the entire FoV frame.

How to capture the pattern and utilize the pattern? Due to this inherent nature of compute, the pattern between left eye and right eye can be easily **captured** by profiling the distance vector for only the first row on the screens: $\Delta = [\vec{x}^0_r - \vec{x}^0_l, \vec{y}^0_r - \vec{y}^0_l]$, as shown in line number 2 in Algorithm 1. With the learned pattern, the remaining ith rows for the right-eye ($i \in [1, n-1]$, where n is the $height$ of the VR screen) can be *reconstructed* by using the projection computation results of the left-eye ($[\vec{x}^i_l, \vec{y}^i_l]$) and the pattern Δ, as shown in line 6 in Algorithm 1. Note that, as discussed above, 2.7% *exclusive* pixel coordinates for the right-eye cannot be reconstructed by this algorithm. Therefore, only for this small number of pixel coordinates, the entire coordinate projection computations need to be processed.

Algorithm 1 Algorithm to capture and utilize the pattern Δ

Input: $[\vec{x}^{0:n-1}_l, \vec{y}^{0:n-1}_l]$: left-eye projection (all rows)
Input: $[\vec{x}^0_r, \vec{y}^0_r]$: right-eye first-row's projection
Output: $[\vec{x}^{1:n-1}_r, \vec{y}^{1:n-1}_r]$: right-eye projection

1: **procedure** CAPTUREPATTERN($\vec{x}^0_r$, $\vec{y}^0_r$, $\vec{x}^0_l$, $\vec{y}^0_l$)
2: $[\Delta_x, \Delta_y] := [\vec{x}^0_r - \vec{x}^0_l\ ,\ \vec{y}^0_r - \vec{y}^0_l]$
3: **return** $\Delta = [\Delta_x, \Delta_y]$
4: **end procedure**
5: **procedure** UTILIZEPATTERN($\vec{x}^{1:n-1}_l$, $\vec{y}^{1:n-1}_l$, Δ)
6: $[\vec{x}^{1:n-1}_r\ ,\ \vec{y}^{1:n-1}_r] := [\vec{x}^{1:n-1}_l + \Delta_x\ ,\ \vec{y}^{1:n-1}_l + \Delta_y]$
7: **return** $[\vec{x}^{1:n-1}_r, \vec{y}^{1:n-1}_r]$
8: **end procedure**

The Effect of AE: with this *AE* optimization, for the right eye, the intensive projective transformation computations can now be short-circuited by light-weight *Add* operations. As a result, by exploiting the *AE* scheme on the first frame in Fig. 4ⓑ, its compute energy can be reduced to only 62.35% of that consumed by baseline.

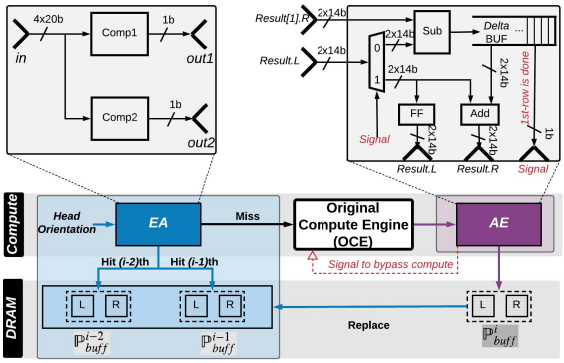


Fig. 7: The proposed *EA* and *AE* design blocks implementation.

E. Design Considerations and Implementation

We designed both our schemes as modular and scalable additions to the existing pipeline (refer the *EA* and *AE* blocks shown in Fig. 7). The *EA* block is placed before the original compute engine (OCE, e.g., GPU) to opportunistically bypass the projection computation. However, when *AE* cannot take advantage of memoization due to a head orientation change, then the compute is distributed across the OCE (51%) and *AE* block (49%); to be precise, only the entire coordinates on the left screen and the first row on the right-screen are processed by the OCE – the remaining rows on the right screen are reconstructed by the less power-hungry *AE* block.

Implementation Details: We abstract the *EA* and *AE* design blocks irrespective of the underlying hardware SoCs, and plot them in Fig. 7. In the *Baseline*, the OCE takes the head orientation as its input, processes the entire projection transformation for each pixel coordinate in the FoV region, and then stores the compute results for both eyes in DRAM for the subsequent mapping stage from the 360° frame to framebuffer. In our *EA* design, the last two head orientations are *cached* in local SRAM ($Comp1$ and $Comp2$ in the *EA* block) and their corresponding projection computation results are stored in DRAM ($\mathbb{P}^{i-2}_{buff}$ and $\mathbb{P}^{i-1}_{buff}$). Once the current head orientation is received, the *EA* block first compares it with the memoized $Comp1$ and $Comp2$. If a match is detected, then the corresponding $\mathbb{P}^{i-2}_{buff}$ or $\mathbb{P}^{i-2}_{buff}$ buffer address pointer is directly returned. If no match is found, the OCE is invoked for the entire left eye and only the first row for the right eye, and then terminates by an external signal sent from our *AE* block, and bypasses the computation for rest rows. In the proposed *AE* design, the Δ pattern buffer is first initialized by subtracting $Result[1].R$ from $Result[1].L$, as shown in the *AE* block in Fig. 7. After the pattern between left eye and right eye is captured, an external signal is propagated to the OCE to bypass the further original projection computations. Consequently, the projection computation results ($Result[2:n].R$) for the remaining rows of the right eye can be easily reconstructed by adding $Result[2:n].L$ and the Δ.

We prototyped our proposed *EA* and *AE* design blocks using System Verilog in Xilinx Vivado 2019.2 [58], targeting the Xilinx Zynq-7000 SoC ZC706 board running at 100MHz (same as state-of-the-art EVR [28]). The evaluation shows that our *EA* and *AE* designs consume only $2mW$ and $65mW$, respectively, and are able to deliver around 100 fps, which is more than sufficient for the current VR application requirements.

V. Evaluation

We compare our proposed *EA* and *AE* designs with six different VR streaming setups, by evaluating the computation and the total energy consumption. In this section, we first describe the experimental platforms, datasets and measurement tools used in this study. We then analyze the results measured using these platforms.

A. VR Design Configurations

We evaluate the following six configurations of VR streaming to demonstrate the effectiveness of Déjà View:

- ***Baseline* (SW):** We use a mobile GPU [36] to evaluate the baseline VR video streaming. This GPU is commonly used in contemporary VR devices (Oculus [39], Magic Leap [16], and GameFace [47], etc.). Note that, with this setup, the projection computation is triggered for each frame, and also per projection computation invocation includes computation of two projection matrices for the two eyes.
- ***PTU* (HW):** A recent optimized solution [28] utilizes a more energy-efficient hardware accelerator, i.e., Projective Transformation Unit (*PTU*), to process the compute-intensive projection operations. This is the most recent VR design that uses an FPGA for accelerating the computation. We consider this design as the *state-of-the-art*. However, *PTU* only optimizes the energy per compute through acceleration, with exactly the "same amount of computations" as in the baseline design. In contrast, as explained earlier in Sec. IV, our design skips a huge amount of computations by exploiting the *EA* and *AE*.
- ***EA* (SW)**: We evaluate the $InterFrame, IntraEye(EA)$ design on a GPU, as shown in the *EA* block in Fig. 7. Note that, this implementation is purely done in software, without any hardware modification.
- ***AE* (SW)**: We evaluate the $IntraFrame, InterEye(AE)$ design on a GPU, and bypasses the projection computation for the right-eye by *reconstructing* the results with a learned pattern, as shown in the *AE* block in Fig. 7.
- ***EA+AE* (SW)**: The above two designs can be seamlessly integrated into the original SoC, with the *EA* block placed before the GPU and the *AE* block after the GPU. We denote this design combination as *EA+AE*.
- ***PTU+EA+AE* (HW)**: In addition to the GPU-based design, our proposed designs can also be integrated into any other hardware platforms, including the FPGA-based PTU [28]. The *PTU+EA+AE* implementation combines the *PTU* and our *EA+AE* optimizations together.

B. Experimental Platforms and Datasets

Evaluation Platforms: The *Baseline* GPU platform described in Fig. 8 consists of a 512-core Volta GPU, a 4Kp60 HEVC

codec, 16GB LPDDR4x memory, 32GB eMMC storage, and a power management unit (PMU) that exposes the real-time power traces to users. To evaluate our design implementation in hardware, we use an FPGA platform, which is the same as the state-of-the-art PTU [28], with a 100MHz system clock, onboard configuration circuitry, 2x16MB Quad SPI Flash, 1GB DDR2 Component Memory, and also a hardware PMU. A full seat Vivado design suite [58], [59] is utilized to synthesize the design and report the power and timing numbers. We collect the display traces from a 5-inch (130 mm) 16:9 1080p (1920 × 1080) AMOLED display [54], which is similar to the Samsung Gear VR display [45].

360° VR Video Dataset: We use the published 360° Head Movements Dataset [3], which includes head movement traces from 59 users viewing seven widely-variant 360° VR videos.[6] The meta information of these VR videos are listed in Tab. II.

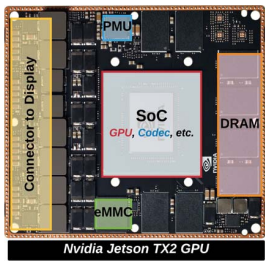


Fig. 8: Evaluation prototype – Nvidia Jetson TX2 GPU board [36] (PMU: Power Management Unit).

C. Experimental Results

We present and compare the energy consumption of the projection computation and the corresponding video quality impact, when running the five VR videos described in Tab. II, with the six design configurations discussed in Sec. V-A. These energy results are normalized w.r.t. the *Baseline* method. Later, we show quality results compared to the baseline design. In addition, we also discuss the *our design's versatility* on other 360° video representation formats.

Energy Savings: Overall, our software implementation EA+AE on GPU can save 54% computation, which translates to 28% total energy savings, compared to the baseline. Compared to the state-of-the-art hardware-modified PTU, our software implementation can still provide 16% computation and 8% total energy savings. Our FPGA results can further provide 18% more computation and 9% more total energy savings, compared to the state-of-the-art design. More specifically, for each of the five video inputs (shown in the x-axis in Fig. 9), we compare the compute energy consumption incurred by six schemes with left-eye and right-eye breakdown, and present the respective compute energy in the left y-axis Fig. 9, which is further translated to the total end-to-end energy savings shown in the right y-axis in Fig. 9. From this figure, we observe that:

- *Baseline:* In *Baseline*, since there are no optimizations, the projection operations for both eyes consume equal energy (on GPU), i.e., each eye's compute consumes 50% energy.
- *EA:* With our proposed *EA* scheme, we fully exploit the *temporal compute reuse* across frames with head orientations unchanged, with a negligible overhead (1% extra overhead, as discussed in Sec. IV-C). In this scheme, one can observe from Fig. 9 that, the compute consumes less energy than the *Baseline*, i.e., only 72% on average. This occurs as a result of reusing the memoized results which have been computed and stored previously, ranging from 21.63% (`Rollercoaster` video) to 50.28% (`Paris` video). By applying the *EA* optimization itself, on an average, the energy benefit is translated to 14% end-to-end energy savings, as shown on the right y-axis in Fig. 9.
- *AE:* For those head orientations not memoized, we further exploited the *spatial compute reuse* across eyes within a frame. In the proposed *AE* scheme, one can observe that for the left-eye computation, the energy consumption is the same as in the *Baseline*. Recall from the *AE* design logic in Fig. 7 that, the results of the left-eye are generated by the Original Compute Engine (GPU in this case) and fed into the *AE* block with the first row for the right-eye, to store the pattern into the Delta Buffer. After that, the computation for the right-eye can be easily reconstructed by the left-eye's compute results and the pattern, which only consumes 13% energy compared to the *Baseline*. Therefore, as shown in Fig. 9, our proposed *AE* optimization alone saves 37% compute energy compared to the *Baseline*, translating to 19% total energy saving.
- *EA+AE:* With both *EA* and *AE* optimizations deployed, as shown in Fig. 9, on average, the left-eye compute consumes only 36% energy w.r.t. the *Baseline*, with only 10% for the right-eye, translating to 28% total energy saving.
- *PTU:* In the current state-of-the-art scheme, which is the hardware-based PTU [28], they explored the energy-efficient hardware accelerator (namely, PTU) to replace power-hungry GPU. Due to this, one can observe from Fig. 9 that, to execute the same amount of the projection computation, the *PTU* scheme consumes only 62% of energy w.r.t. the *Baseline*, which contributes to 20% total energy saving.
- *PTU+EA+AE:* Note that, our proposed *EA* and *AE* designs are "independent" of the underlying hardware used. As a result, they can also be deployed on top of the PTU-based SoC. This can be further asserted from Fig. 9, that only 28% of the compute energy is consumed w.r.t. the *Baseline* (22% for the left-eye, 6% for the right-eye), translating to a 37% total energy saving.

Impact on Quality: The proposed *AE* scheme captures the pattern between both the eyes with only the $1st$ row of the frame, and then uses the same pattern to *bypass* the projection computation for the remaining rows of the right eye. Note that, as shown in Fig. 6b and 6c, the ith-row's pattern may not be exactly the same as the jth-row. This is due to the floating-point hardware rounding (to find the nearest-neighbor integer coordinates) and the transformation matrix's various weights are dependent on the row numbers. To simplify our *AE* design, we simply reuse the pattern captured in the 1st row, and do not consider the deeper information related to the row numbers. To study how this decision affects the video quality, we report

[6]Due to space limitation, here we only present 5 videos and 20 users.

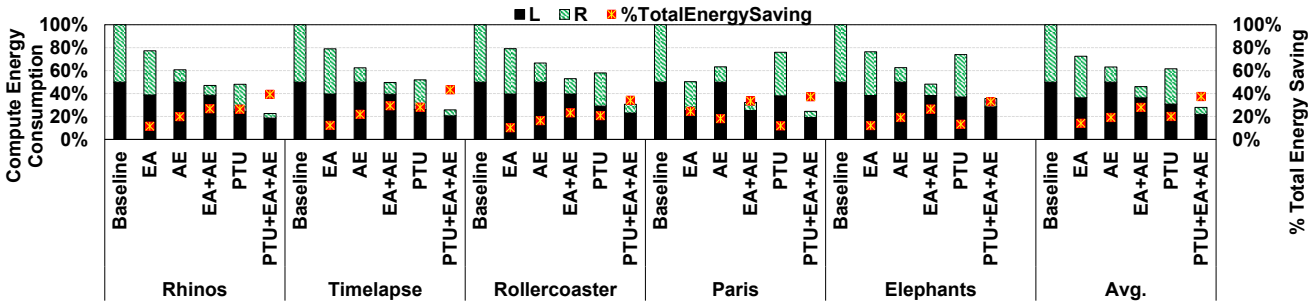


Fig. 9: Normalized energy consumption and savings with different configurations and video inputs. The left y-axis shows the compute energy consumption normalized to the compute energy consumption in Baseline (the lower, the better). The right y-axis shows the amount of energy savings compared to the end-to-end total energy consumption in Baseline (the higher, the better).

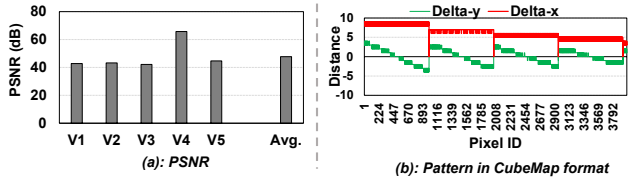


Fig. 10: Sensitivity study. (a): Video quality metric (PSNR [25], [40]) across video inputs. (b): The pattern between left-eye and right-eye in the *front* face in Cube Mapping [41].

the averaged PSNR [25], [40] of the five videos represented in Equirectangular format [52] in Fig. 10ⓐ. These results indicate that, although we ignore the row-number related information, the resulting PSNR is still sufficient (47.71 on average) for VR video applications [26], [62].

General Applicability of Déjà View: The above discussion assumes that the Equirectangular format [52] is used to represent the 360° videos. We want to emphasize that our underlying ideas behind the proposed *EA* and *AE* (designed for the Equirectangular format) can work irrespective of the representation formats used [48]. For example, similar to the distance vector study in Fig. 6b and Fig. 6c, we plot the distance pattern between both eyes of a 360° frame using the CubeMap format [41] in Fig. 10ⓑ. Clearly, a pattern (different from the `ellipse` observed with the Equirectangular format) exists in Δ_x and Δ_y. Note that the pattern behavior also depends on the row numbers. This again validates the quality impact discussed earlier. Putting together, these above observations indicate that, with very little change (to capture the row-dependent information) in our original *AE* design, our idea is able to work with any representation format. This motivates us to target on further improving quality across video formats by capturing the information related to row numbers in future.

VI. Related Work

Optimizations in Planar Video Streaming: Pixel-similarity based optimizations [38], [57] have been exploited to improve performance in 2D rendering. For example, ATW [38] is a post-render technique, which sits between rendering (our focus) and display. To reduce the impact by the long-latency from rendering, ATW either guesses the next head-orientation or only considers the rotation (no translation), then skews two already-rendered planar FoV frames to remove judders [43]. Note that, this computation still happens in planar-format, and remains the same between two-eyes for one frame. Targeting on ATW, PIMVR [57] proposed a 3D-stacked HMC to reduce Motion-to-Photon latency and off-chip memory accesses. Motivated by the observation that the ATW transform matrix generated by rotation on a 2D image is shared by both eyes, PIMVR [57] calculated the transform matrix only once, and scheduled two tiles (one for left-eye one for right-eye) with the same coordinate to the same vault in HMC. However, in contrast to the 360° VR video streaming, ATW is in 2D planar format, and share the same compute results across eyes. These two characterizations indicate that such optimizations in the planar world are infeasible to be applied in 3D PT-rendering.

Hardware assist on VRs: Various energy-efficient hardware modifications [28] have been proposed to reduce energy consumption in the VR domain. For example, PTU [28] uses a hardware-accelerated rendering unit (HAR) to mitigate energy-overheads due to on-device rendering. In this work, we propose two optimizations, i.e., *EA* and *AE*, which can be coupled with the existing 360° video compute engine (without any hardware modifications), and are even more energy efficient than the existing state-of-the-art *PTU* (discussed in Sec. V). Moreover, *EA* and *AE* can further be integrated into *PTU* to save even more energy.

Pixel Content Reuse on VRs: Pixel value reuse has been well-studied in VRs [17], [22], [23], [29], [33], [50], [60], [66] to improve throughput and performance. For example, DeltaVR [29] adaptively reuses the redundant VR pixels across multiple VR frames to improve performance. These works focus on the pixel content reuse, which is the last stage (*Projection Mapping*) in the 360° video projection pipeline (discussed in Sec III). However, none of these existing schemes leverage reducing the large amounts of "redundant" computations in the preceding stage (projection computation).

Our proposed *EA* and *AE* designs focus on these intensive projection computations, and as such are orthogonal to these prior efforts. In our future work, we would like to further explore the benefits by incorporating them into our design.

Head Orientation Prediction for 360° Video Streaming: To optimize both performance and energy, researchers have leveraged the powerful remote rendering engines on cloud to predict the next head orientation for the VR clients [2], [6], [18], [23], [30]–[32]. FlashBack maintains a storage cache of multiple versions of pre-rendered frames, which can be quickly indexed by head orientations [2]. In comparison, Semantic-Aware-Streaming (SAS) exploits the semantic information inherent in a VR video content to precisely predict users' next head orientations [28]. These optimizations rely on the powerful cloud with a high bandwidth access, which may not be always available. However, our work focuses on edge-side optimization, which can also be implemented as a complementary add-on in such cloud-assisted systems.

Energy Optimizations in Conventional Video Processing: In the existing planar video processing pipeline on mobile devices, prior works have looked at memory [1], [63], [64], display [13]–[15], [34] and codec [49], [65], and identified "memory" as the major energy bottleneck. For example, AFBC [1] is proposed to efficiently compress video streams between the processing pipeline blocks. MACH [63] integrates a display cache to reduce the amount of memory bandwidth. Although, these techniques can potentially save memory usage/energy for 360° VR videos, as discussed in earlier sections, due to inherent nature of 360° video processing, which introduces additional overheads for projection computation, we identify *compute* to be the major energy bottleneck. Hence, these memory optimizations are not applicable to reduce compute energy on 360° VR videos.

VII. Concluding Remarks

360° VR videos have become the next trend in entertainment media and soon will become an integral part of the technology influencing many application domains. However, unlike planar videos, the 360° VR video streaming demands significantly more compute power from a battery-operated headset. Thus, prior research has proposed using accelerators for optimizing the computations.

In contrast, this paper attempts to exploit available "redundancies" in computation by analyzing the VR projection computation pipeline. Specifically, we propose and evaluate in detail two pluggable schemes (for taking advantage of intrinsic temporal-spatial reuse), and prototype them as microarchitectural augmentations using FPGA. Our experimental results show 34% computation reduction and 17% energy savings, compared to the state-of-the-art [28]. In the future, we would also like to explore other opportunities to improve energy efficiency and tune the computation pipelines to cater more towards VR applications. We believe, given that the current VR devices are battery-backed, these kinds of energy savings and performance improvements will not only enable the users to experience longer videos, but also encourage both industry and academia to work further on improving the pipeline to make VR more pervasive and versatile.

Acknowledgment

This research is supported in part by NSF grants #1763681, #1629915, #1629129, #1317560, #1526750, #1714389, #1912495, and a DARPA/SRC JUMP grant. We would also like to thank Dr. Jack Sampson, Dr. Aasheesh Kolli and Dr. Timothy Zhu for their feedback on this paper.

References

[1] Arm Holdings, "Arm Frame Buffer Compression (AFBC)." "https://developer.arm.com/architectures/media-architectures/afbc", 2019.

[2] 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*, ser. MobiSys '16, 2016, pp. 291–304.

[3] X. Corbillon, F. De Simone, and G. Simon, "360-Degreee Video Head Movement Dataset," in *Proceedings of the 8th ACM on Multimedia Systems Conference*, 2017, pp. 199–204.

[4] Discovery, "Caring for Rhinos: Discovery VR (360 Video)." "https://www.youtube.com/watch?v=7IWp875pCxQ", 2019.

[5] Discovery, "Elephants on the Brink." "https://www.youtube.com/watch?v=2bpICIClAIg", 2019.

[6] T. El-Ganainy and M. Hefeeda, "Streaming Virtual Reality Content," *CoRR*, vol. abs/1612.08350, 2016. [Online]. Available: http://arxiv.org/abs/1612.08350

[7] Facebook, "Facebook 360," "https://facebook360.fb.com/", 2019.

[8] Facebook Inc., "Facebook Oculus," "https://www.oculus.com/".

[9] Google, "360° videos - Google Arts & Culture," "https://artsandculture.google.com/project/360-videos".

[10] Google, "More Ways to Watch and Play with AR and VR." "https://blog.google/products/google-vr/more-ways-watch-and-play-ar-and-vr".

[11] Google, "Build Virtual Worlds." "https://developers.google.com/vr", 2019.

[12] Google, "GVR Android SDK Samples - Video360." "https://github.com/googlevr/gvr-android-sdk/blob/master/samples/sdk-video360/src/main/java/com/google/vr/sdk/samples/video360/VrVideoActivity.java#L257", 2019.

[13] M. Ham, I. Dae, and C. Choi, "LPD: Low Power Display Mechanism for Mobile and Wearable Devices," in *Proceedings of the USENIX Conference on Usenix Annual Technical Conference (ATC)*, 2015, pp. 587–598.

[14] K. Han, Z. Fang, P. Diefenbaugh, R. Forand, R. R. Iyer, and D. Newell, "Using Checksum to Reduce Power Consumption of Display Systems for Low-motion Content," in *2009 IEEE International Conference on Computer Design*, 2009, pp. 47–53.

[15] K. Han, A. W. Min, N. S. Jeganathan, and P. S. Diefenbaugh, "A Hybrid Display Frame Buffer Architecture for Energy Efficient Display Subsystems," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2013, pp. 347–353.

[16] T. HARDWARE, "Magic Leap One Powered by Nvidia Tegra TX2, Available Summer." "https://support.oculus.com/248749509016567/", 2019.

[17] B. Haynes, A. Mazumdar, A. Alaghi, M. Balazinska, L. Ceze, and A. Cheung, "LightDB: A DBMS for Virtual Reality Video," *Proc. VLDB Endow.*, pp. 1192–1205, 2018.

[18] J. He, M. A. Qureshi, L. Qiu, J. Li, F. Li, and L. Han, "Rubiks: Practical 360-Degree Streaming for Smartphones," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, 2018, pp. 482–494.

[19] HEADJACK, "The Best Encoding Settings For Your 4k 360 3D VR Videos + FREE Encoding Tool," "https://headjack.io/blog/best-encoding-settings-resolution-for-4k-360-3d-vr-videos/".

[20] C. Heather Bellini, W. Chen, M. Sugiyama, M. Shin, S. Alam, and D. Takayama, "Virtual and Augmented Reality." "https://www.goldmansachs.com/insights/pages/technology-driving-innovation-folder/virtual-and-augmented-reality/report.pdf", 2016.

[21] L. F. Hodges, "Tutorial: Time-multiplexed Stereoscopic Computer Graphics," *IEEE Computer Graphics and Applications*, pp. 20–30, 1992.

[22] A. Holdings, "White Paper: 360-Degree Video Rendering." "https://community.arm.com/developer/tools-software/graphics/b/blog/posts/white-paper-360-degree-video-rendering", 2019.

[23] J. Huang, Z. Chen, D. Ceylan, and H. Jin, "6-DOF VR Videos with a Single 360-camera," *2017 IEEE Virtual Reality (VR)*, pp. 37–44, 2017.

[24] A. Inc., "Rendering Omni-directional Stereo Content." "https://developers.google.com/vr/jump/rendering-ods-content.pdf", 2019.

[25] N. INSTRUMENTS, "Peak Signal-to-Noise Ratio as an Image Quality Metric." "https://www.ni.com/en-us/innovations/white-papers/11/peak-signal-to-noise-ratio-as-an-image-quality-metric.html", 2019.

[26] B. C. Kim and C. E. Rhee, "Compression Efficiency Evaluation for Virtual Reality Videos by Projection Scheme," *IEIE Transactions on Smart Processing & Computing*, pp. 102–108, 2017.

[27] S. M. LaValle, "The Geometry of Virtual Worlds." "http://msl.cs.uiuc.edu/vr/vrch3.pdf", 2019.

[28] Y. Leng, C.-C. Chen, Q. Sun, J. Huang, and Y. Zhu, "Energy-efficient Video Processing for Virtual Reality," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2019, pp. 91–103.

[29] Y. Li and W. Gao, "DeltaVR: Achieving High-Performance Mobile VR Dynamics through Pixel Reuse," in *2019 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2019, pp. 13–24.

[30] L. Liu, R. Zhong, W. Zhang, Y. Liu, J. Zhang, L. Zhang, and M. Gruteser, "Cutting the Cord: Designing a High-quality Untethered VR System with Low Latency Remote Rendering," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '18, 2018, pp. 68–80.

[31] X. Liu, Q. Xiao, V. Gopalakrishnan, B. Han, F. Qian, and M. Varvello, "360° Innovations for Panoramic Video Streaming," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XVI, 2017, pp. 50–56.

[32] B. Luo, F. Xu, C. Richardt, and J. Yong, "Parallax360: Stereoscopic 360° Scene Representation for Head-Motion Parallax," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1545–1553, 2018.

[33] A. Mazumdar, T. Moreau, S. Kim, M. Cowan, A. Alaghi, L. Ceze, M. Oskin, and V. Sathe, "Exploring Computation-communication Tradeoffs in Camera Systems," in *2017 IEEE International Symposium on Workload Characterization (IISWC)*, 2017, pp. 177–186.

[34] H. Miao and F. X. Lin, "Tell Your Graphics Stack That the Display Is Circular," in *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '16, 2016, pp. 57–62.

[35] mooovr, "RollerCoaster at Seoul Grand Park." "https://www.youtube.com/watch?v=8lsB-P8nGSM", 2019.

[36] Nvidia, "JETSON AGX XAVIER AND THE NEW ERA OF AUTONOMOUS MACHINES." "http://info.nvidia.com/rs/156-OFN-742/images/Jetson_AGX_Xavier_New_Era_Autonomous_Machines.pdf", 2019.

[37] Oculus, "Rendering to the Oculus Rift," "https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-render/".

[38] Oculus, "Asynchronous TimeWarp (ATW)." "https://developer.oculus.com/documentation/mobilesdk/latest/concepts/mobile-timewarp-overview/?locale=en_US", 2019.

[39] Oculus, "Oculus Rift and Rift S Minimum Requirements and System Specifications." "https://www.tomshardware.com/news/magic-leap-tegra-specs-release,37443.html", 2019.

[40] OpenCV, "Similarity check (PNSR and SSIM) on the GPU." "https://docs.opencv.org/2.4/doc/tutorials/gpu/gpu-basics-similarity/gpu-basics-similarity.html", 2019.

[41] OpenGL, "Cubemaps - Learn OpenGL." "https://learnopengl.com/Advanced-OpenGL/Cubemaps", 2019.

[42] OpenGL, "The Industry's Foundation for High Performance Graphics." "https://www.opengl.org/", 2019.

[43] O. Rift, "Oculus Rift - How Does Time Warping Work?" "https://www.youtube.com/watch?v=WvtEXMlQQtI", 2019.

[44] Samsung, "Samsung Gear VR," "https://www.samsung.com/global/galaxy/gear-vr/".

[45] Samsung, "Explore New Dimensions." "https://www.samsung.com/global/galaxy/gear-vr/#display", 2019.

[46] SkySports, "Sky VR Virtual Reality," "https://www.skysports.com/mobile/apps/10606146/sky-vr-virtual-reality", 2019.

[47] Tom's HARDWARE, "Nvidia's Jetson TX2 Powers GameFace Labs' Standalone VR Headset." "https://www.tomshardware.com/news/gameface-labs-standalone-steamvr-headset,37112.html", 2019.

[48] R. Toth, J. Nilsson, and T. Akenine-Möller, "Comparison of Projection Methods for Rendering Virtual Reality," in *Proceedings of High Performance Graphics*, ser. HPG '16, 2016, pp. 163–171.

[49] C.-H. Tsai, H.-T. Wang, C.-L. Liu, Y. Li, and C.-Y. Lee, "A 446.6 K-gates 0.55–1.2 V H. 265/HEVC decoder for next generation video applications," in *2013 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2013, pp. 305–308.

[50] A. Vlachos, "Advanced VR Rendering in Valve." "http://media.steampowered.com/apps/valve/2015/Alex_Vlachos_Advanced_VR_Rendering_GDC2015.pdf", 2019.

[51] F. G. VR360, "Virtual guided tour of Paris." "https://www.youtube.com/watch?v=sJxiPiAaB4k", 2019.

[52] Wikepedia, "Equirectangular Projection." "https://en.wikipedia.org/wiki/Equirectangular_projection", 2019.

[53] Wikipedia, "Pixel 2," "https://en.wikipedia.org/wiki/Pixel_2".

[54] Wikipedia, "Active-Matrix Organic Light-Emitting Diode," "https://en.wikipedia.org/wiki/AMOLED", 2019.

[55] Wikipedia, "Virtual Reality." "https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio#:~:targetText=Typical\%20values%20for%20the%20PSNR,20\%20dB%20to%2025%20dB.", 2019.

[56] B. Worldwide, "NYC 360 Timelapse." "https://www.youtube.com/watch?v=CIw8R8thnm8", 2019.

[57] C. Xie, X. Zhang, A. Li, X. Fu, and S. Song, "PIM-VR: Erasing Motion Anomalies In Highly-Interactive Virtual Reality World with Customized Memory Cube," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2019, pp. 609–622.

[58] Xilinx, "Vivado Design Hub - Installation and Licensing," "https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0013-vivado-installation-and-licensing-hub.html".

[59] Xilinx, "Vivado Design Suite - HLx Editions." "https://www.xilinx.com/products/design-tools/vivado.html", 2019.

[60] xinreality, "Asynchronous Spacewarp." "https://xinreality.com/wiki/Asynchronous_Spacewarp", 2019.

[61] YouTube, "Get Started with YouTube VR." "https://support.google.com/youtube/answer/7205134?hl=en", 2019.

[62] V. Zakharchenko, K. P. Choi, and J. H. Park, "Quality metric for spherical panoramic video," in *Optics and Photonics for Information Processing X*, K. M. Iftekharuddin, A. A. S. Awwal, M. G. Vázquez, A. Márquez, and M. A. Matin, Eds., International Society for Optics and Photonics. SPIE, 2016, pp. 57 – 65.

[63] H. Zhang, P. V. Rengasamy, S. Zhao, N. C. Nachiappan, A. Sivasubramaniam, M. T. Kandemir, R. Iyer, and C. R. Das, "Race-to-sleep + Content Caching + Display Caching: A Recipe for Energy-efficient Video Streaming on Handhelds," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2017, pp. 517–531.

[64] H. Zhang, S. Zhao, A. Pattnaik, M. T. Kandemir, A. Sivasubramaniam, and C. R. Das, "Distilling the Essence of Raw Video to Reduce Memory Usage and Energy at Edge Devices," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2019, pp. 657–669.

[65] D. Zhou, S. Wang, H. Sun, J. Zhou, J. Zhu, Y. Zhao, J. Zhou, S. Zhang, S. Kimura, T. Yoshimura, and S. Goto, "14.7 a 4gpixel/s 8/10b h.265/hevc video decoder chip for 8k ultra hd applications," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, 2016, pp. 266–268.

[66] Y. Zhu, A. Samajdar, M. Mattina, and P. Whatmough, "Euphrates: Algorithm-SoC Co-design for Low-power Mobile Continuous Vision," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2018, pp. 547–560.