Efficient Radius Search for Adaptive Foveal Sizing Mechanism in Collaborative Foveated Rendering Framework

Yufei Yang, Chenhao Xie, *Member, IEEE*, Liansheng Liu, *Member, IEEE*, Philip H. W. Leong, *Senior Member, IEEE*, Shuaiwen Leon Song, *Member, IEEE*

Abstract—Collaborative Foveated Rendering (CFR) is the latest collaborative rendering framework proposed to enable high frame rate VR applications on mobile devices. Compared with the strategies adopted in conventional collaborative rendering, the pixel-based Adaptive Foveal Sizing (AFS) mechanism in CFR offers a more flexible and intelligent workload trade-off by predicting the radius. However, the performance of the AFS mechanism in actual deployment depends on its adaptability to two factors, including the Sudden Environmental Variations (SEV) and the Random Discrete Latency (RDL). Guaranteeing the performance of the AFS mechanism by adapting to these two factors is of great significance to guaranteeing users' immersive experience. **This paper identifies the existence of the SEV and RDL phenomenon in the AFS mechanism for the first time, and contributes the first method that offers the effective and real-time AFS mechanism implementation for the practical deployment, namely the Efficient Radius Search (ERS).** The ERS method efficiently searches the largest radius online that controls the rendering workload within the foveated layer just below the offline baked threshold, thereby achieving the immediate response to SEV and reducing the oscillating frame rendering latency led by RDL. **Through the experiments on 3 VR applications and 4 mobile devices, the resulting 2.44× to 9.07× higher frame rate precision compared with the state-of-the-art method demonstrate the superiority of the ERS method.**

Index Terms—Mobile devices, virtual reality, collaborative foveated rendering, untethered

1 Introduction

VIRTUAL Reality (VR) devices can be categorized into Mobile-Powered VR (MVR) and PC-Powered VR (PCVR), where MVR is more competitive than PCVR in terms of mobility and price, but has difficulty achieving the high frame rate due to the limited battery power and graphics rendering capabilities. Therefore, providing a high frame rate on MVR is of great importance for its wider adoption in the VR-based future metaverse.

Cloud Rendering is the earliest technology proposed to improve the frame rate of MVR, in which all rendering tasks are offloaded from mobile device to cloud server with powerful graphics processing capabilities [1], then transmit the rendered results back to the mobile devices through the wireless network. Tiled-based panoramic frames transmission with dynamic quality are widely adopted in Cloud Rendering [2]–[5], some are equipped with the user motion prediction to assist the quality selection [6], [7]. Besides, different sorts of bit-rate adjustment algorithms including [8]–[15] are proposed to dynamically encode panoramic frames, thereby increasing its adaptability to wireless network environment variation. However, mobile device in Cloud Rendering is normally treated as the video decoder, and its graphic rendering resources are not utilized.

Collaborative Rendering proposed in [16], can make up for the deficiency in Cloud Rendering by partitioning the graphics rendering tasks into Foreground Interactions (FI) and Background Environment (BE), and distributing them separately to mobile device and cloud server. Technologies developed in the related research including motion prediction [17] and similarity evaluation [18] are effective on reducing the transmission cost of BE frames under the wireless network. Nevertheless, handling the additional overhead brought by those methods such as the storage overhead and motion sickness [19], [20] is still difficult. Furthermore, FI rendered on different mobile devices and VR applications requires to be manually defined by experienced developers to satisfy real-time constraints, which is labor intensive and impractical [21].

To address the challenges in Collaborative Rendering, Collaborative Foveated Rendering (CFR) proposed in Q-VR [21] subtly integrates collaborative mechanism into Foveated Rendering [22]. It designs the Adaptive Foveal Sizing (AFS) mechanism as the core technology to automatically balances the workload distribution between the mobile device and cloud server. However, we find for the first time that the performance of the AFS mechanism in practical deployment is determined by its adaptability to two factors, including Sudden Environmental Variations (SEV) and Random Discrete Latency (RDL) phenomenon. Q-VR's implementation to the AFS mechanism lacks of

- *Yufei Yang and Liansheng Liu are with the School of Electronics and Information Engineering, Harbin Institute of Technology, Harbin, Heilongjiang, China. E-mail: yyf305016@163.com*
- *Chenhao Xie is with High-Performance Computing Group at Pacific Northwest National Laboratory (PNNL), Richland, Washington, the United States.*
- *Philip H. W. Leong is with the School of Electrical and Information Engineering, The University of Syndey, Sydney, NSW 2006, Australia.*
- *Shuaiwen Leon Song is with the School of Computer Science, The University of Syndey, Sydney, NSW 2006, Australia.*

Manuscript received April 19, 2005; revised August 26, 2015.

the consideration to these two factors, thus its performance in practical deployment is limited.

The Efficient Radius Search (ERS) presented in this work, is the first method to provide the effective and real-time AFS mechanism implementation for the practical deployment. In the ERS method, an offline algorithm is developed to bake the threshold table for the whole VR application scene. And an online algorithm is designed to firstly derive the correct threshold from the offline baked threshold table with respect to user's coordinates, then search the largest radius to control the rendering workload within the foveated layer just below the weighted threshold. The rendering workload estimation in both offline and online algorithms is time-consuming under the complex VR application, hence a dedicated accelerator is designed to minimize the algorithm execution latency. **Extensive experiments on various VR applications and mobile devices demonstrate that the proposed ERS method has higher adaptability to both SEV and RDL phenomenon than the state-of-the-art Q-VR method, which is reflected through the 2.44× to 9.07× higher frame rate precision and one millisecond-level execution latency.**

2 Background

Since this paper discusses aspects crossing over multiple disciplines, significant background are provided in this section to help different communities to better understand the content.

2.1 Collaborative Foveated Rendering

Foveated Rendering [22] coupled with eye-tracking is a innovative technology that reduce the overall rendering workload based on the human visual perception difference to different parts of the screen. Apart from prior works that focused on exploring the flexibility of shading techniques [23] [24], Foveated Rendering was also extended to light fields [25] and video streaming [26]. Besides, the frequency of eye tracker was dramatically improved in recent work [27].

The lower part of Fig. 1 shows the schematic diagram of the Foveated Rendering in mobile virtual reality. The gaze point is where users are starring at the device screen (captured by Eye Tracker), and the part of screen that surrounds the gaze point is the foveated layer, which should be rendered in a high resolution to guarantee users' vision perception. The horizontal resolution of the foveated layer in pixels, D_{FL}, is defined in Equation (1) [22]:

$$D_{FL}(r) = \frac{2\,D\,H\,r}{W} \tag{1}$$

where r is the radius of the foveated layer, D is the screen horizontal resolution in pixels, H and W are the height and width of the screen in centimeter, respectively.

Human eyes have falling acuity to the rest part of screen, hence the peripheral Layer can be rendered in decreasing resolution to reduce the overall rendering workload. The horizontal resolution of the peripheral layer in pixels, D_{PL}, is defined in Equation (2) [22]:

$$D_{PL}(r) = \frac{D\,tan^{-1}(\frac{2\,W}{V\,D})}{m\,arctan(\frac{r\,H}{V}) + \omega_0} \tag{2}$$

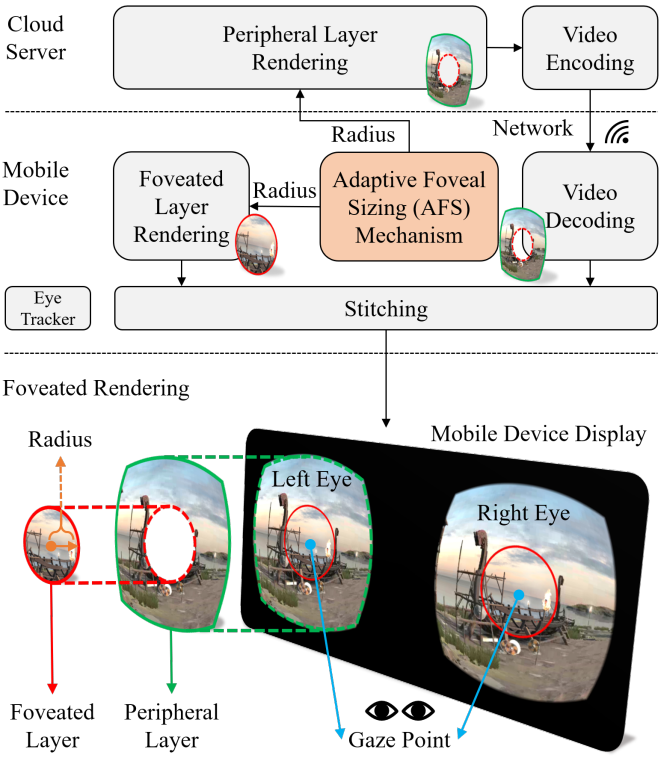

Fig. 1. **Architecture of Collaborative Foveated Rendering [21] in Virtual Reality**

where V is the eye-to-screen distance in centimeter. Both m and ω_0 are constant value, m is within the range of 0.022∼0.034 to cover a wide range of user's sensitivity to blur phenomenon, and the representative value of ω_0 is 1/48. As indicated in Equation (1) and Equation (2), r is proportional to D_{FL} and inversely proportional to D_{PL}. Besides, there is a transition operation at the junction of the foveated layer and the peripheral layer. This part usually uses the interpolation algorithm to fuse the pixels at the junction of the two sides, thereby avoiding the visual perception loss caused by the rendering resolution difference.

Q-VR [21] innovatively applied the collaborative mechanism adopted in [16]–[18] into Foveated Rendering and proposed the Collaborative Foveated Rendering, whose schematic diagram is shown in the upper part of Fig. 1. In each frame during VR application run-time, the eye tracker will firstly capture the user eyes' gaze point on the screen and use the Adaptive Foveal Sizing (AFS) mechanism to set up the optimal radius r for the foveated layer, then perform foveated layer rendering on the mobile device. Simultaneously, the cloud server will perform rendering for the peripheral layer with dedicated resolution determined by radius r, and transmit the rendered result back to the mobile device to stitch with the locally rendered foveated layer. Finally, show the stitch results on the device screen.

2.2 AFS Mechanism

Under the Collaborative Foveated Rendering context shown in Fig. 1, the composition of the local latency L_{Local} on mo-

bile device and the remote latency L_{Remote} on cloud server are described in Equation (3) and Equation (4), respectively:

$$L_{Local} = L_{FL} \tag{3}$$

$$L_{Remote} = L_{PL} + L_E + L_{NT} + L_D \tag{4}$$

where L_{FL} and L_D are the foveated layer rendering latency and the peripheral layer decoding latency on the mobile device, L_{PL} and L_E is the peripheral layer rendering and encoding latency on the cloud server, L_{NT} is the network transmitting latency of the encoded results. As indicated in Equation (5), the instant frame rate FR is the reciprocal of the larger value between L_{Local} and L_{Remote}:

$$FR = \frac{1}{max[L_{Local}, L_{Remote}]} \tag{5}$$

The instant frame rate FR must be maximized for the best immersive experience, which means both L_{Local} and L_{Remote} should be minimized. The target of the Adaptive Foveal Sizing (AFS) mechanism is to generate the optimal radius r_o that balances the workload distribution between mobile device and cloud server, thereby minimizing both L_{Local} and L_{Remote}.

3 Influence Factors of AFS Mechanism Performance

To better understand the research problem, two influence factors that highly affect the performance of the AFS mechanism are defined and discussed separately in this section.

3.1 Factor I: Sudden Environmental Variation

During the run-time of the VR application with the high user interaction requirements, for example, the racing game and the first-person shooting game, the environment variations between adjacent frame is sometimes violent and unpredictable. This phenomenon is named Sudden Environmental Variation (SEV).

As illustrated in section 2.2, the AFS mechanism implementation in Q-VR applies the Q-learning algorithm to gradually learn the most appropriate radius variation $\triangle r$ for the current environment. Such settings are not able to perform real-time reactions to SEV because of three reasons. Firstly, the Q-learning algorithm must take a certain amount of frames to learn how to provide an optimal $\triangle r$ no matter how hyperparameters are tuned. Secondly, users' motion and gaze point changes in adjacent frames are entirely random, which further increases the convergence difficulty because the status selection s is random. Thirdly, whenever the user moves to a new environment whose rendering workload distribution is slightly different from the former environment, the quality score in the former Q table is no longer of the reference value, and the algorithm has to restart the learning process. During the learning process, the AFS mechanism can not work normally, and the inappropriate radius output will lead to the oscillating frame rate.

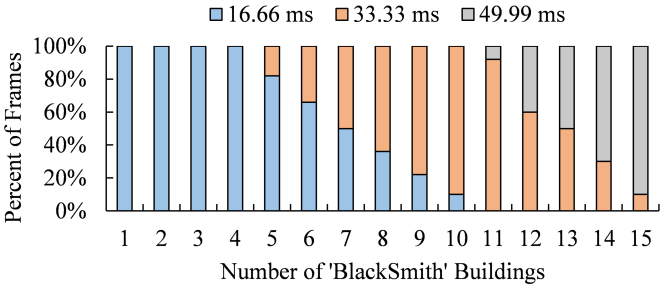

Fig. 2. Instant Frame Latency Distribution of 100 frames under Different Number of 'Blacksmith' Buildings

3.2 Factor II: Random Discrete Latency

Randomness is one of the fundamental characteristics of frame rendering latency, which is affected by a large number of factors in the rendering system. Lots of studies propose to predict the frame rendering latency based on different factors. Some studies start from the bottom level of the rendering pipeline, and predict the frame rendering latency/workload based on the intermediate factors such as vertex [28], vertices [29] [30], triangles [28] [31], projected pixels [29], Light [32], API calls and textures [33] [30]. Others take the CPU and GPU working frequency or utilization ratio as reference factors [34] [35] [36] [37] to predict the frame rendering latency/workload. Although these methods have good performance under specific restrictions and different test platforms including Mobile [28] [33] [30] [34] [36] [37], PC [31] [29] [32] [35], they cannot achieve stable prediction accuracy. On one hand, some rendering factors is on-the-fly data so that it is difficult to take all rendering factors at very beginning into consideration. On the other hand, the hardware level factors are usually invisible from software level and they also contain certain randomness, such as hardware preemption, in real-case.

Instead of continuous on developing a higher precision frame rendering latency/workload estimation method like the research mentioned above, we consider if it is possible to directly model the random behavior of the frame rendering latency using large amount of real mobile devices experiments, and try to find a solution based on the modeling results.

To model the random behavior of the frame rendering latency, a preliminary modeling experiment is performed as follows. We firstly create a simplified 'Viking Village' test scene [38] in Unity that only contains specific amounts of 'Blacksmith' building duplicates. Then fix the camera towards the buildings to keep the rendering content unchanged, and attach a script that can record the instant frame rate of consecutive 100 frames to repeatedly render this content. Finally, we repeatedly export the test scene and run it on Moto G9 Plus mobile device each time with one more 'Blacksmith' building to simulate the linearly increased rendering workloads.

The experiment results shown in Fig. 2 provide two significant observations. Firstly, only three discrete frame rendering latency values 16.66, 33.33, and 49.99 ms related to $n = 1, 2, 3$ are detected along 15 levels of complexity

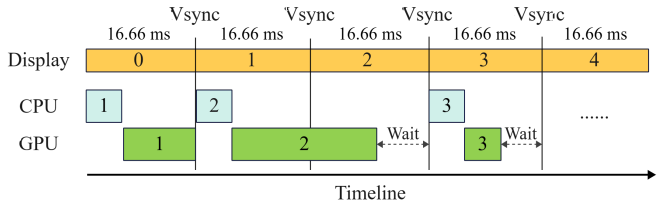

Fig. 3. Schematic Diagram of the Random Discrete Latency Phenomenon

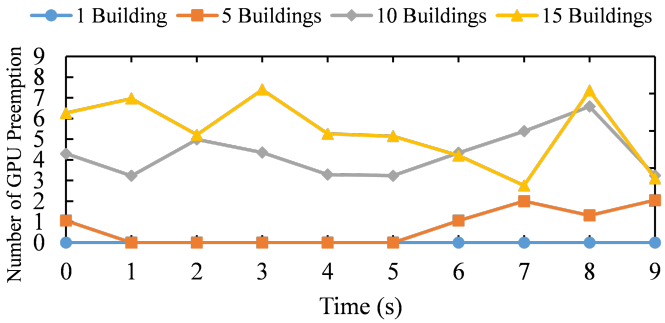

Fig. 4. **The Amount of GPU Preemption Occured per Second under Different Number of 'Blacksmith' Buildings**

due to the forced Vsync mechanism on mobile devices. In order to optimize the display performance, Google has reconstructed the Android display system since Android 4.1 and realized Project Butter, that is, only after receiving the Vertical Synchronization (Vsync) pulse, the CPU and GPU will start the rendering of the next frame, just like it is shown in Fig. 3. The iOS system also possesses a similar measure. For mobile devices with the 60 Hz screen refresh rate, the interval between adjacent Vsync pulse is the fixed 16.66 ms, which is the root reason for the discrete frame rendering latency. The system will not render in advance or later, which extends the battery life and optimizes the user interaction experience.

Secondly, the frame rendering latency is the random value among 16.66 ms, 33.33 ms, and 49.99 ms or even bigger if the number of 'Blacksmith' buildings exceeds a threshold 4. **One reason for the random phenomenon is the hardware resource preemption between the VR application and the hidden process during run-time. Fig. 4 shows the amount of GPU preemption occured per second captured by Qualcomm Snapdragon Profiler [39] 'Realtime' function when rendering different amounts of 'Blacksmith' buildings. The data is obtained from Moto G9 Plus mobile device. When there is only 1 building appeared in the camera, no GPU preemption is triggered. But With more buildings rendered simultaneously, GPU preemption occurs more frequently.** We argue that if the rendering task of the current frame is not able to be finished within 16.66 ms or 33.33 ms due to the randomly triggered resource preemption, it will be extended to the nearest impending Vsync pulse at 33.33 ms or 49.99 ms as the block 2 in Fig. 3 shows, which will degrade the performance of the AFS mechanism, and finally results in the overall oscillating frame rate.

The experiment above is further extended to two different interaction objects and three other mobile devices. The results listed in Table 1 demonstrate that there is a unique rendering workload threshold T for each interaction object and the mobile device. Through the above experiments and analysis, the random behavior model of the frame rendering latency can be defined using Equation (6):

$$L = \begin{cases} 16.66 & RW \leq T \\ n * 16.66 & RW > T \end{cases} \tag{6}$$

where L is the frame rendering latency in milliseconds, n is a random positive integer. We are able to achieve a close to 100% steady 16.66 ms frame rendering latency if the rendering workload RW is less than or equal to the unique threshold T. If RW is higher than T, n is a random integer

TABLE 1
Workload Threshold T of Four Mobile Devices on Three Types of Interaction Objects

T (Amount of Objects)	'Building'	'Tree'	'TableSet'
Huawei Nova4	5	83	90
Moto G9 Plus	4	30	42
Vivo X60	16	43	176
iPhone 12	23	68	194

and the frame rendering latency will oscillate, which marks the exception of AFS mechanism. In general, the random behavior described using Equation (6) is named Random Discrete Latency (RDL).

4 Research Problem

The research problem of this paper is how to guarantee the performance of the AFS mechanism by adapting to the SEV and RDL phenomenon discussed in section 3.

For high adaptability to the SEV phenomenon, machine learning should be carefully used in the AFS mechanism, especially when designing the algorithm that has high-frequency interaction with users. Because machine learning method may incur extra computing latency, hardware resource competition, and high interaction latency discussed in section 3.1.

For high adaptability to the RDL phenomenon, we start with the target of the AFS mechanism defined in section 2.2, which is to locate the optimal radius r_o that minimizes both the local latency L_{Local} and the remote latency L_{Remote}. To locate the optimal radius r_o, the first step is to discover the relationship between radius r and L_{Local}. Equation (3) indicates L_{Local} is only consists of the foveated layer latency L_{FL}, and the behaviour of L_{FL} follows the RDL phenomenon described with Equation (6). Therefore, the method that can minimize L_{FL} and adapt to the RDL phenomenon at the meanwhile, is to find the radius r that controls the foveated layer's rendering workload RW below the specific rendering workload threshold T. However, the relationship between RW and r is actually random and it is difficult to obtain a fixed mathematical model that describes this relationship. In some cases, reducing r does shrink the foveated layer, but a tiny part of one object is still visible to the camera, and the CPU will still submit the whole

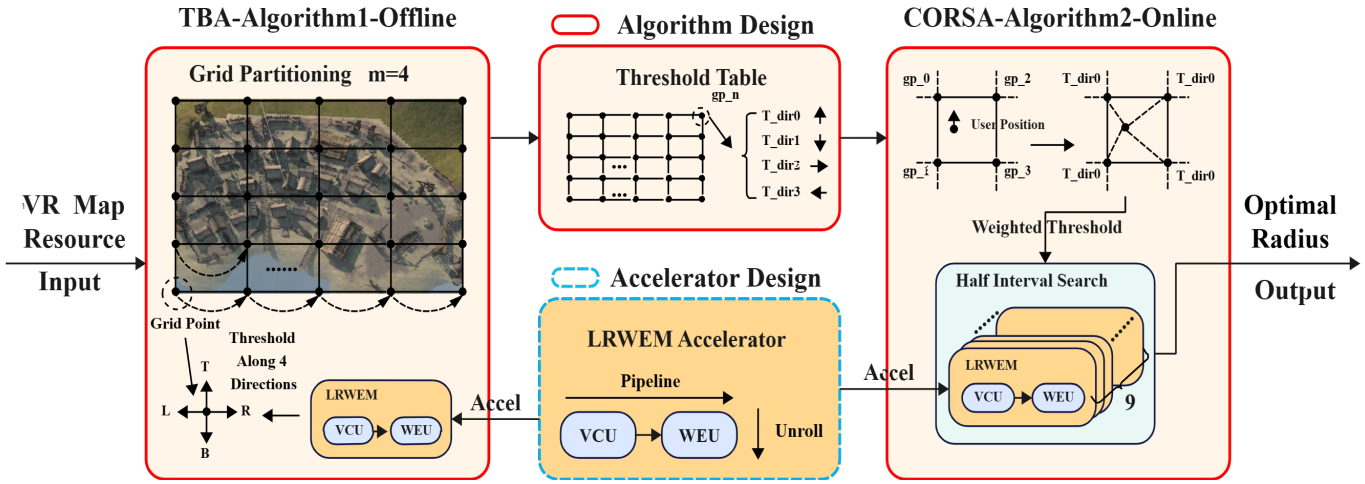

Fig. 5. **Algorithm Design and Accelerator Deign of the Proposed Efficient Radius Search Method**

mesh of this object to GPU to perform rendering, thus the RW remains unchanged. In another case, the objects are distributed with a big enough interval within the scene, reducing the r may linearly reduce RW. Therefore, we extend Equation (6) to identify the relationship between r and L_{Local} in Equation (7):

$$L_{Local}(r) = \begin{cases} 16.66 & RW(r) \leq T \\ n * 16.66 & RW(r) > T \end{cases} \tag{7}$$

The second step is to explore the relationship between radius r and L_{Remote}. In Collaborative Foveated Rendering, the peripheral layer will be rendered by cloud server and transmitted back to mobile device through the network. As indicated in Equation (2), the larger r takes the smaller D_{PL}, thereby bringing in the lower L_{NT} and the possible lower or unchanged L_E, L_{NT}, and L_D, finally resulting in the lower L_{Remote}. Therefore, we define Equation (8) to identify the relationship between r and L_{Remote}:

$$L_{Remote}(r) = L_{PL,E,D}(r) + \frac{D_{PL}(r)^2 \; P_D \; C_R}{8 \; B_W} \tag{8}$$

where $L_{PL,E,D}(r)$ indicates the latency sum of peripheral layer rendering, encoding, and decoding operations, which does not belong to the research scope of this paper. The latter part refers to the network transmitting latency L_{NT} of the encoded data, where P_D is the bit depth of each pixel, C_R refers to the compression rate of the selected codec, B_W is the network bandwidth between the cloud server and the mobile device in $Bytes/s$.

As indicated in Equation (2), radius r is inverse proportional to D_{PL}, thus the larger r brings the smaller L_{Remote} according to Equation (8). Therefore, referring to Equation (7), the optimal radius r_o should be defined as the **largest** r that controls $RW(r)$ below the rendering workload threshold T, so that both L_{Local} and L_{Remote} are minimized and the adaptability to the RDL phenomenon can be significantly improved. Equation (9) describes the mathematical model of the optimal radius:

$$\begin{aligned} & r_o = \max(\underset{r}{\operatorname{argmin}}(L_{Local}(r), L_{Remote}(r))) \\ & \text{s.t. } r \in [0.05, 0.5]. \end{aligned} \tag{9}$$

r_o is searched within the normalized range of 0.05 and 0.5, where 0.05 refers to the minimal 5° central area indicated in [22], while 0.5 represents the whole screen.

5 Efficient Radius Search Method

As shown in Fig. 5, we propose the Efficient Radius Search (ERS) method to tackle the research problem defined in section 4. The ERS method takes the VR application map as the input and output the optimal radius r_o for each frame. The algorithms and the accelerator design of the ERS method will be discussed separately in the following subsections.

5.1 Algorithm Design

The rounded rectangle boxes circled by the solid line in the left, up middle, and right side of Fig. 5 represent the algorithm design of the ERS method. The first functional module is the Threshold Baking Algorithm (TBA), which is responsible for automatically baking the rendering workload threshold table TT for the input VR application scene. TT will be delivered to the second functional module, namely Conditional Optimal Radius Search Algorithm (CORSA), to search and output the optimal radius r_o during VR application run-time.**It should be noted that the offline TBA algorithm is only for the VR application developing team when testing the their product on mainstream mobile devices, and the online CORSA algorithm is integrated in the VR application rendering pipeline, which will be automatically performed when user use the VR application.**

5.1.1 *Threshold Baking Algorithm*

The value of the rendering workload threshold T along different areas varies with the VR application scene complexity distribution, thus adopting only one global T for the whole application cannot always satisfy Equation (7). Threshold Baking Algorithm (TBA) based on Grid Partitioning (GP) is proposed to record T for the selected grid points [1] within

1. **Grid point is a physical location in the VR application used to provide threshold value when user get closer.**

the scene. **As shown in Fig. 5, the VR application scene is divided by the parameter grid partitioning density m, for example in , if $m = 4$, the scene is divided into 4×4 pieces, with 25 grid points in total.** Then bake T of 4 directions including top, bottom, right, and left for each grid point, and finally store the results in the threshold table TT. Algorithm 1 illustrates the details of TBA.

For each direction in each grid point, r will keep reducing from 0.5 to 0.05 under a certain step si to locate r that can bring the minimal average local latency $avgL_{Local}(r)$, until $avgL_{Local}(r)$ has already been close to the minimal value 16.66 ms (reciprocal of 60 FPS) or r has reached to 0.05. The final r will be fed to Lightweight Rendering Workload Estimation Module (LRWEM) to calculate the rendering workload. The calculated rendering workload under final r is the rendering workload threshold T and should be stored in the threshold table TT.

Algorithm 1 Threshold Baking Algorithm

Input: GPs: The group of grid points
si: Radius searching interval
Output: TT: The table that stores T of all grid points

for each gp in GPs **do**
  for each dir in 4 $Dirs$ of current gp **do**
    $r_{final} = 0$, $MinAvgL_{Local}(r) = 9999$
    for $r = 0.5$ to 0.05 with step si **do**
      $sumL_{Local}(r) = 0$, $avgL_{Local}(r) = 0$
      for $i = 1$ to 10 do **do**
        $camera.Render(r)$
        $sumL_{Local}(r) += L_{Local}(r)$
      end for
      if $i = 10$ **then**
        $avgL_{Local}(r) = sumL_{Local}(r)/10$
        if $(avgL_{Local}(r) < MinAvgL_{Local}(r))$ **then**
          $MinAvgL_{Local}(r) = avgL_{Local}(r)$
          $r_{final} = r$
        end if
      end if
      if ($avgL_{Local}(r) == 16.66$ or $r == 0.05$) **then**
        $T =$ **LRWEM**(r_{final})
        $TT.Add(T)$
        $break$
      end if
    end for
  end for
end for

5.1.2 Conditional Optimal Radius Searching Algorithm

Conditional Optimal Radius Searching Algorithm (CORSA) based on Half Interval Search (HSA) is proposed to efficiently search the optimal radius r for each frame during run-time. With the offline baked threshold table TT from TBA, CORSA can synthesis the reasonable threshold T for any positions within the scene by weighting the T of the nearest 4 grid points.

As outlined in Algorithm 2, CORSA firstly looks up the closest 4 grid points according to the user's position and rotation, then fetch the corresponding rendering workload of each grid point from the input threshold table TT. For example in Fig. 5, the user is facing forward and locating among 4 gird points including gp_0, gp_1, gp_2, and gp_3, hence T_{dir0} of those 4 grid points are fetched from the threshold table TT. Then use Equation (10) and Equation (11) to calculate the respective weights ω_0, ω_1, ω_2, and ω_3, and calculate the weighted T to be the threshold for the current position.

$$\beta_n = \frac{dis_n}{\sum dis_n} \tag{10}$$

$$\omega_n = \frac{1}{1 + \frac{\beta_n}{\prod \beta_m} \sum \frac{1}{\beta_m}} \quad \text{s.t. } n \in [0,1,2,3],\ m \neq n \tag{11}$$

β_n is the intermediate variables. Finally, the Half-Interval Search Algorithm (HSA) is adopted to efficiently locate the optimal radius r_o. The key principle of HSA is to half the searching interval si after each search, and the searching direction sd depends on the comparison results between the weighted T and the rendering workload RW output by LRWEM. Therefore, useless intervals are quickly skipped and the searching efficiency is greatly improved compared with exaustive search. The mechanism of HSA guarantees the final radius is always the maximum value whose RW is just below the weighted T, so that Equation (9) is satisfied. Besides, the radius error is smaller enough than 0.005 if searching times $st = 9$.

Algorithm 2 Conditial Optimal Radius Searching Algorithm

Input: TT: The table that stores T of all grid points
Output: r_o: The optimal radius that satisfies Equation (4)

$dis_0, dis_1, dis_2, dis_3 = Distance(gp_0, gp_1, gp_2, gp_3, T)$
$\omega_0, \omega_1, \omega_2, \omega_3 = Weights(dis_0, dis_1, dis_2, dis_3)$
$T = \omega_0 * T_{dir0} + \omega_1 * T_{dir0} + \omega_2 * T_{dir0} + \omega_3 * T_{dir0}$
Half Interval Search:
HSA parameters: $st = 8$, $si = 0.45$, $sd = -1$, $r = 0.5$
for $i = 0$ to st **do**
  $RW =$ **LRWEM**(r)
  if $(RW < T)$ **then**
    $sd = 1$
  else if $(RW == T)$ **then**
    break
  else
    $sd = -1$
  end if
  si = sd * si / pow(2, i)
  r += si
end for
$r_o = r$
$camera.Render(r_o)$

5.1.3 Lightweight Rendering Workload Estimation Module

As shown in Algorithm 1 and 2, both TBA and CORSA will use Lightweight Rendering Workload Estimation Module (LRWEM) to generate unified rendering workload estimation. LRWEM is designed based on the Occlusion Culling (OC) and Level of Details (LOD) technologies applied in the rendering pipeline's application stage.

OC technology will perform a series of calculations so that only the objects that are visible to the camera will be submitted from CPU to GPU for rendering. Therefore, the rendering workload on CPU RW_{CPU} can be simply described using Equation (12):

$$RW_{CPU} = M \tag{12}$$

where M is the visible objects calculated by the Visibility Checking Unity (VCU), whose details are outlined in Algorithm 3. The visibility of all objects $Vis[N]$ is finally obtained by comparing their Clip Plane Coordinates (CPC) with default camera View Port Coordinates (VPC).

Algorithm 3 Visibility Checking Unit (VCU)

Input: $Objs[N]$: All N Objects, r: Input Radius
Output: $Vis[N]$: Vsibility of N Objects, 1: Visible; 0: Invisible

Projection Matrix: PM, Culling Matrix: CM, World to Camera Matrix: WCM, Clip Plane Coordinates: CPC, View Port Coordinates: VPC
$CM = PM(r) * WCM$
for each obj in $Objs[N]$ **do**
  $CPC = obj.Pos * CM$
  if (CPC is within the range of VPC) **then**
    $Vis[obj] = 1$
  else
    $Vis[obj] = 0$
  end if
end for

LOD technology will dynamically select the mesh with different triangles amounts in each object according to its distance to the camera, which means if the object is close to the camera, the mesh with more triangles will be rendered by GPU to provide users with clearer details, if the object is far away from the camera, the mesh with fewer triangles will be rendered to avoid additional GPU rendering workload brought by unnecessary details. Therefore, the rendering workload on GPU RW_{GPU} can be simply described using Equation (13):

$$RW_{GPU} = \sum_{i}^{M} Tri_i * (1 - \frac{D_i}{D_r}) \tag{13}$$

Where Tri_i refers to the triangles amount of the current object, D_i represents the distance between the current object and the camera, and D_r is the maximum distance in the current VR scene. Algorithm 4 illustrates the rendering workload estimation module designed based on Equation (12) and Equation (13). Only the visible objects from VCU are counted for final rendering workload estimation.

5.2 Accelerator Design

The rounded rectangle boxes circled by the dotted line in the down middle side of Fig. 5 represent the accelerator design of the ERS method. LRWEM should execute as fast as possible to minimize the additional latency introduced to CORSA, as it will be executed in each frame during runtime. To tackle this issue, we design the High Level Synthesis (HLS) oriented accelerator for LRWEM. The LRWEM

Algorithm 4 Workload Estimation Unit (WEU)

Input: $Vis[N]$: Visibility of All N Objects
Output: RW_{CPU}: Rendering Workload of CPU
  RW_{GPU}: Rendering Workload of GPU

for each obj in $Vis[N]$ **do**
  if ($Vis[obj] == 1$) **then**
    $RW_{CPU} += 1$
    $RW_{GPU} += obj.Tri * (1 - \frac{obj.Dis}{D_r})$
  end if
end for

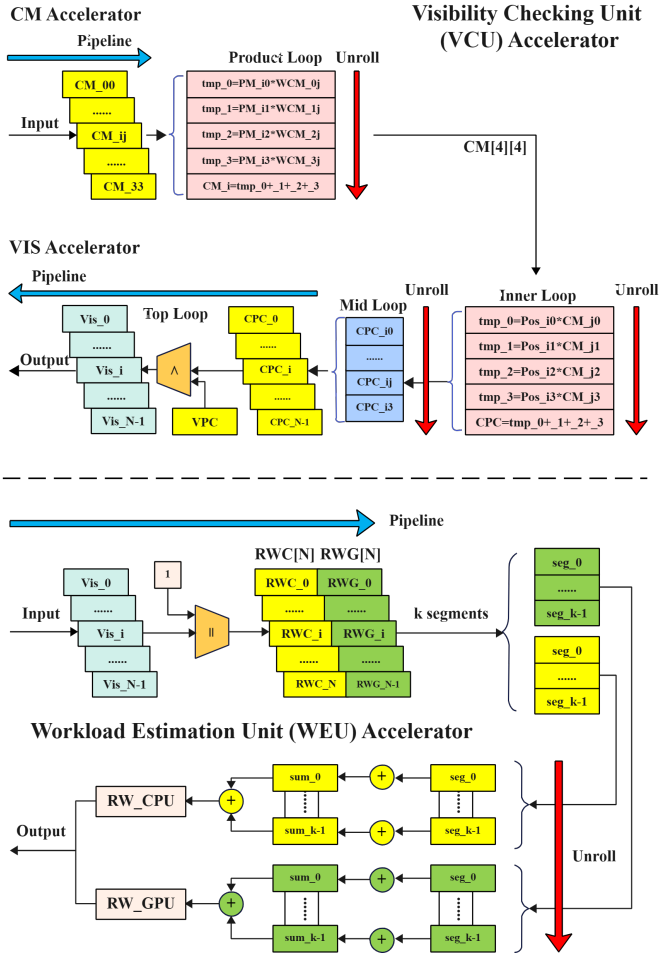

Fig. 6. Visibility Checking Unit and Workload Estimation Unit Accelerator

accelerator is consists of the VCU accelerator and WEU accelerator that accelerate Algorithm 3 and 4, respectively. The architectures of those two sub accelerators are shown in Fig. 6

5.2.1 VCU Accelerator

VCU accelerator includes VIS accelerator and CM accelerator, which are used for accelerating different steps in Algorithm 3. CM accelerator is responsible for accelerating standard 4x4 matrix multiplication $CM = PM * WCM$, which can be implemented as three overlapped for loops in the C programming language. We firstly add UNROLL directives to the inner loop to enable parallel execution of the $+=$ operation, then add PIPELINE directives to

upper loops to increase the throughput. The output CM by CM accelerator will be delivered to the VIS accelerator to calculate CPC, then compare with VPC to obtain the visibility $Vis[N]$ of all N objects. N may be extremely large in complex VR applications, therefore the strategy we adopt is to add PIPELINE directives to the for loop so that the additional hardware resources required for the throughput improvement are ignorable.

5.2.2 WEU Accelerator

As shown in Algorithm 4, WEU takes $Vis[N]$ from VCU as the input, then calculates and sum RW_{CPU} and RW_{GPU} based on Equation (12) and Equation (13), which is the main performance bottleneck in LRWEM. Due to the access order restrictions to RW_{CPU} and RW_{GPU} BRAMS under $+=$ operation and the large N value, accelerating WEU requires tricks. We rewrite Algorithm 4 for the convenience of HLS acceleration, the sum operation $+=$ is separated from the calculation operation of RW_{CPU} and RW_{GPU}, thus the PIPELINE directive can be applied to the calculation operation. Then we divide the separated $+=$ operation into k segments so that HLS will automatically conduct the intra-segment rendering workload summation for all k segments in parallel, finally perform the inter-segment rendering workload summation to get the final RW_{CPU} and RW_{GPU}.

6 Performance Evaluation

6.1 Prototype Implementation

To evaluate the performance of the proposed ERS method, we compare it with the state-of-the-art AFS mechanism implementation, Q-VR [21]. The algorithm design of both the ERS method and the Q-VR method are implemented on Unity 2020.2.7f1c1 [40] using C# script inherited from Unity $Monobehaviour$ base class [41], $UnityEngine$ API [42], and $.NetStandard2.0$ system libraries [43]. The effect of the Foveated Rendering is achieved by adjusting the projection matrix of the camera API provided by $UnityEngine$.

The accelerator design of the ERS method is implemented through the Vivado Design Suite. The implementation of HSA algorithm is firstly converted from C# to C program. Secondly, Vivado HLS 2017.4 [44] is adopted to apply the HLS directives illustrated in Section 5.2 to the converted C program, add the AXI interface, run synthesis to generate the corresponding accelerator, and export it as an IP core for Vivado 2017.4. Thirdly, the Processing System IP core and the accelerator IP core are instantiated in Vivado 2017.4, and the AXI connection between them is built, so that the accelerator can be controlled through CortexA9 CPU in Vivado SDK 2017.4.

6.2 Experiments Setup

We deploy the Q-VR method and the proposed ERS method on 4 commercial mobile devices and three commonly adopted open-source VR applications in former research [2] [3]. **Mobile devices include Huawei Nova4, MotoG9 Plus, Vivo X60, and iPhone 12, whose chipsets and OS configurations are listed in Table 2.** VR applications include Nature [45], Office [46], and Viking Village [38], whose statistic information including the total number of objects and triangles are summarized in Table 3.

TABLE 2
Statistic Info of Four Tested Mobile Devices

Devices	Huawei Nova4	Moto G9 Plus	Vivo X60	iPhone 12
SoC	Kirin 970	Snapdragon 730G	Exynos 1080	A14 Bionic
OS	Android 8	Android 10	Android 11	iOS 14

TABLE 3
Statistic Info of Three Tested VR Applications

VR Applications	Nature	Office	Viking Village
Number of Objects	322	450	1206
Number of Triangles	0.189M	0.206M	4.693M

To analyse the rationality and effectiveness of the algorithm design in the ERS method, for each VR application, we choose scattered 100 test points that do not overlap with grid points in the TBA algorithm. And for each test point during run-time, we record the variables in the CORSA algorithm including the rendering workload RW on CPU and GPU (RW_{CPU} and RW_{GPU}) at the optimal radius r_o and the average frame rate $avgFR_{r_o}$ at the optimal radius r_o. The ground truth of those variables above are baked using a similar method as Algorithm 1. More specifically, for each test point, we keep decreasing r from 0.5 to 0.05 with a certain stride, records the maximum radius r which can bring the maximum average frame rate as the ground truth radius r_{gt} and average frame rate ground truth $avgFR_{gt}$, and the corresponding rendering workload threshold under r_{gt} is recorded as $T_{CPU_{gt}}$ and $T_{GPU_{gt}}$.

The effectiveness of ERS method should ultimately be reflected on its ability to adapt to the SEV phenomenon and the RDL phenomenon. Due to the existence of the RDL phenomenon, the frame rate improvement evaluation strategy adopted in the previous research [21] is inaccurate because the instant frame rate is a random value if the rendering workload exceeds the threshold. Therefore, the radius must be introduced to the evaluation strategy to better reflect the adaptability to the RDL phenomenon. Based on the r_o, $avgFR_{r_o}$, and their ground truth value recorded above, the frame rate precision P is defined using Equation (14):

$$P = \frac{\sum \left(|avgFR_{r_o} - avgFR_{gt}| \leq \Delta_{FR} \ \ \& \ \ |r_o - r_{gt}| \leq \Delta_r\right)}{N} \tag{14}$$

where $avgFR_{gt}$ and r_{gt} are the ground truth average frame rate and radius just mentioned, Δ_{FR} and Δ_r are allowed errors. Due to the RDL phenomenon, we hope at least 70% of instant Frame Rate $FR(r)$ that involved in the calculation of $avgFR(r)$ should be equal to ground truth FR_{gt} (7 frames in 60 FPS, 2 frames in 30 FPS, and 1 frame in 20 FPS can get average 50 FPS), which results in $\Delta_{FR} = 10$ after conversion ($60 - 50 = 10$), so Δ_{FR} is fixed to 10 in all experiments. In short, the frame rate precision P calculated by Equation (14) can be interpreted as the percent of N test points whose r_o searched by radius obtaining method and the corresponding average frame rate $avgFR_{r_o}$ are close to their ground truth values r_{gt} and $avgFR_{gt}$.

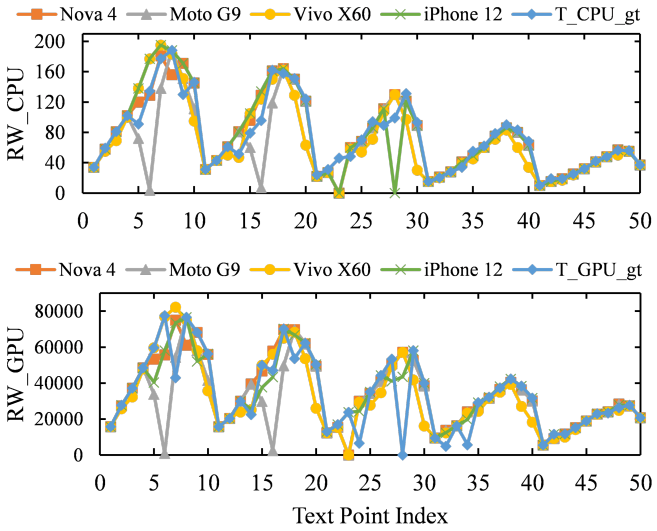

Fig. 7. **Comparison between the rendering workload RW_{CPU} and RW_{GPU} under the optimal radius r_o and the corresponding ground truth rendering workload threshold $T_{CPU_{gt}}$ and $T_{GPU_{gt}}$. The curve is obtained from the Nature VR application at 50 test points deployed on 4 mobile devices when $m = 12$**

The implementation of the accelerator design in the ERS method is deployed on Xilinx Zynq-7000 development board AX7Z035 [47], which takes Xilinx XC7Z035-2FFG676 as the core chip. As illustrated in Section 5.2, the number of objects N and the segment amount k all have a significant impact on the LRWEM accelerator performance. To exactly quantify this impact, we adopt $N = 1000, 3500, 7000$ to represent three levels of high complexity VR applications, set $k = 3, 6$, record the FPGA hardware resource utilization and HSA execution latency on PS and PL of the Zynq-7000 development board, where the resource utilization can be reflected through the Vivado HLS 2017.4 synthesis report, and the execution latency can be obtained through C time library.

6.3 Performance Evaluation to Algorithm Design

This section will firstly evaluate the design rationality of the proposed ERS method, including the offline TBA algorithm and the online CORSA algorithm. Then compare the adaptability of the ERS method and the state-of-the-art Q-VR method to the SEV phenomenon. Finally demonstrate the superiority of the ERS method on the adaptability to the RDL phenomenon compared with the Q-VR method.

6.3.1 Algorithm Design Rationality

Fig. 7 offers an intuitive comparison among the rendering workload RW_{CPU} and RW_{GPU} under the optimal radius r_o at 50 test points and the ground truth rendering workload threshold T_{gt}. The data is obtained from the Nature VR application deployed on four mobile devices when $m = 12$. Remind that the parameter m represents the grid partitioning density adopted in the offline TBA algorithm.

The first phenomenon in Fig. 7 is that all threshold rendering workload curves are jagged and downward, which is caused by the distribution of test points in the test VR applications. The location of the test point starts with the left bottom side of the test scene and horizontally shifts to the right bottom side under a certain stride, then vertically shifts move a distance and repeats the horizontal movement, finally reaching the right top side. The test point on the bottom side can capture more game objects than the top side, which is why all threshold rendering workload curves are generally trending downward. Similarly, the reason why threshold rendering workload curves are jagged is that the test point located in the middle part of each repeated horizontal movement can capture more game objects than two other sides. The above analysis demonstrates that the common module LRWEM within the offline algorithm TBA and the online algorithm CORSA can indeed reflect the rendering workload variations when the surrounding environment changes.

The second phenomenon in Fig. 7 is that, for all mobile devices, the rendering workload RW_{CPU} and RW_{GPU} under the optimal radius r_o overlaps with the ground truth rendering workload threshold $T_{CPU_{gt}}$ and $T_{GPU_{gt}}$ under most test points. This is benefited from the design rationality of the offline algorithm TBA and the online algorithm CORSA. Firstly, TBA and the threshold weighting strategy in CORSA together provides a reasonably weighted rendering workload threshold T that is very close to the ground truth rendering workload threshold for the current test point. Secondly, the subsequent HSA algorithm can efficiently search the optimal radius r_o that controls the rendering workload RW_{CPU} and RW_{GPU} within the foveated layer just below this weighted rendering workload threshold.

There are some test points whose rendering workload have a huge difference with the ground truth value, for example, for Moto G9 Plus, the test point 6 and the test point 16. This is because the test point is located within the rendering object, hence is not visible to the camera and will not be counted to the rendering workload in the online CORSA algorithm. However, this rendering object is visible to the surrounding grid points and will be counted to the rendering workload in the offline TBA algoithm, thereby leading to the huge rendering workload difference under current test point.

6.3.2 Adaptability to SEV Phenomenon

Fig. 8 display the differences between the ground truth radius r_{gt}, the corresponding optimal radius r_o output by the proposed ERS, and the state-of-the-art Q-VR method at 50 test points. The data is obtained from the Viking Village VR application deployed on 4 mobile devices when the grid partitioning density $m = 12$. The environment complexity changes among the 50 test points are similar to that of the Nature VR application described in the section 6.3.1, which all simulate the SEV phenomenon that is quite common during the user use.

As it is shown in Fig.8, under the four tested mobile devices, the optimal radius r_o output by the proposed ERS method have a good fit with the ground truth radius r_{gt}, which may be further improved by increasing the grid partitioning density m value. From the theoretical perspective, the higher m will help offline TBA algorithm provide the online CORSA algorithm with the finer threshold

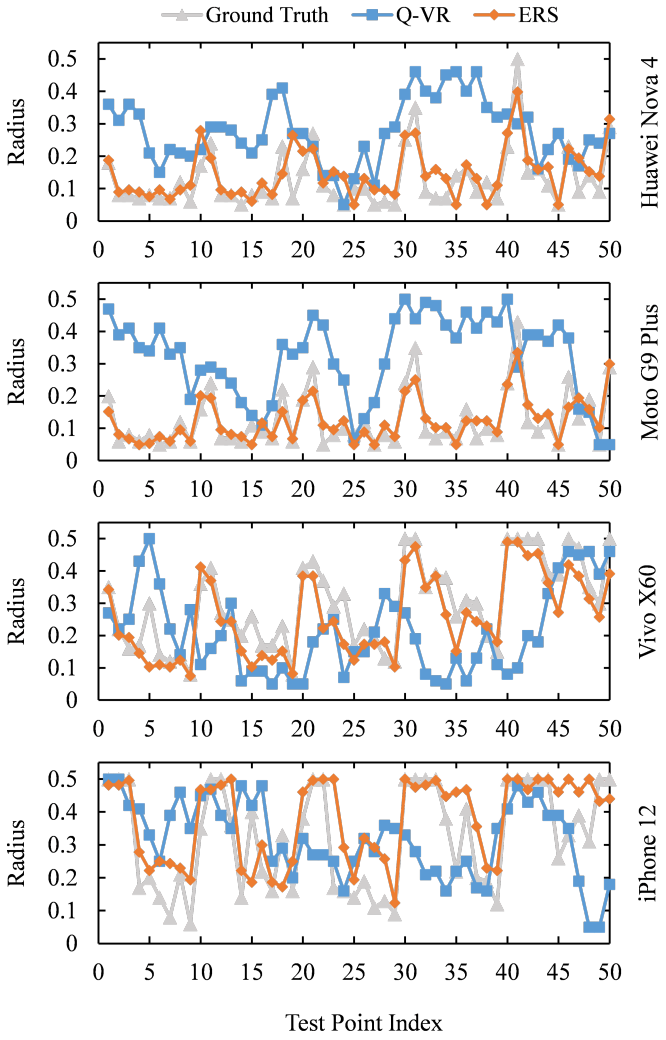

Fig. 8. **Comparison between the optimal radius r_o output by the ERS method, Q-VR method, and the corresponding ground truth radius r_{gt} at 100 test points (zoom in on the first 50 points). The curve is obtained from the Viking Village VR application deployed on 4 mobile devices when $m = 12$**

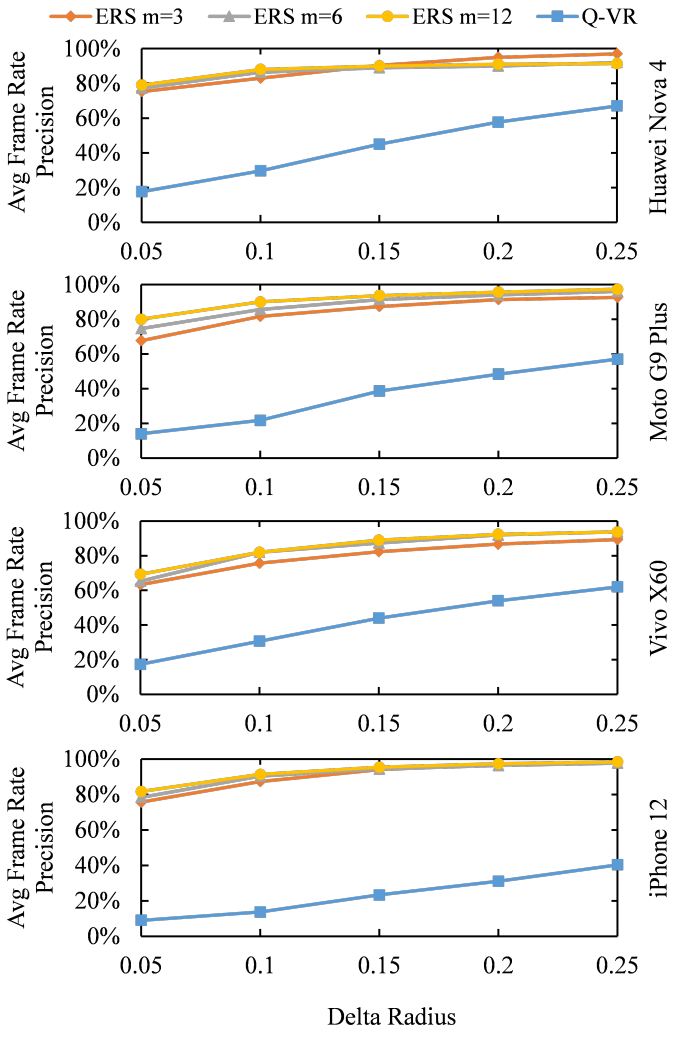

Fig. 9. **Frame rate precision comparison between ERS under different m value and the state-of-the-art method Q-VR ($\Delta_{FR} = 10$, $\Delta_r = [0.05, 0.10, 0.15, 0.20, 0.25]$)**

table, thereby decreasing the difference between r_o and r_{gt}.

Q-VR has difficulty to follow the aggressive environmental changes compared with ERS, which is reflected by the huge difference between the optimal radius r_o output by Q-VR and the ground truth radius r_{gt} as indicated in Fig.8. The Q-learning algorithm adopted by Q-VR does not requires the offline baking process like ERS, but it comes with the price that the online learning mechanism has high reaction latency to the environment.

In general, compared with the state-of-the-art Q-VR method, the experiment above demonstrates the superiority of the proposed ERS method on the adaptability to the SEV phenomenon.

6.3.3 Adaptability to RDL phenomenon

As discussed in Equation (14), the adaptability to the RDL phenomenon can be better quantified by the frame rate precision. Fig. 9 shows the frame rate precision of ERS method and Q-VR method on 3 VR applications including Nature, Office, and Viking Village, and on 4 mobile devices including Huawei Nova4, Moto G9 Plus, Vivo X60, and iPhone 12. The ERS method are tested in three versions of the grid partitioning density including $m = 3$, $m = 6$, $m = 12$. Δ_{FR} is fixed to 10 as indicated in section 6.2, and Δ_r is set to the value in [0.05, 0.10, 0.15, 0.20, 0.25] array in turn.

For all test cases, the proposed ERS method under different grid partitioning density m settings have higher frame rate precision than the state-of-the-art Q-VR method. More specifically, for Huawei Nova4, the ERS method has 1.45× to 4.47× higher frame rate precision than the Q-VR method. For Moto G9 Plus, the ERS method has 1.71× to 5.71× higher frame rate precision than the Q-VR method. For Vivo X60, the ERS method has 1.51× to 4.00× higher frame rate precision than the Q-VR method. For iPhone 12, the ERS method has 2.44× to 9.07× higher frame rate precision than the Q-VR method. Another observation is that frame rate precision is tend to increase

when adopting higher m value, because the finer grid partitioning in TBA will provide CORSA with the more realistic weighted rendering workload threshold table.

In general, compared with the state-of-the-art Q-VR method, the proposed ERS method have 1.45× to 9.07× higher frame rate precision over all test cases in the experiments of this paper, which demonstrates the superiority of the proposed ERS method on the adaptability to the RDL phenomenon.

6.4 Performance Evaluation to Accelerator Design

Table 4 compares the accelerator performance under different number of objects N and the segment amount k mentioned in section 5.2. Firstly, the resource utilization of the proposed accelerator design is acceptable. With the increase of k, more DSP48E, and Flip-Flops are utilized because more float calculations and RAMs are established in the parallel calculation of different segments, for example, $k = 6$ brings a 1.37x higher average speedup ratio from 44.82× to 61.24× with a price of 1% FPGA resource utilization. Higher N requires much higher amounts of BRAMs to store the intermediate calculation results, thereby increasing BRAMs utilization from 5% to 12%. Secondly, when N becomes larger, the accelerator execution latency can still be controlled within 1 ms by increasing k, so as to minimize the impact of LRWEM on the rendering pipeline. For example, the accelerator latency under $N = 3500, k = 3$ is 1.192 ms, we can reduce it to 0.898 ms by increasing k to 6, same principles can be applied to $N = 7000, k = 6$.

In general, the accelerator design of the ERS method can control the algorithm level latency around 1 ms with acceptable additional hardware resources utilization.

TABLE 4
Accelerator Performance and Resource Utilization Comparison on Different k and N Values

k	N	PS (ms)	PL (ms)	SpeedUp	Resource (%)		
					BRAM	DSP	FF
3	1000	17.867	0.409	43.68×	5%	20%	8%
	3500	53.834	1.192	45.16×	7%	20%	8%
	7000	107.242	2.35	45.63×	12%	20%	8%
6	1000	18.135	0.296	61.27×	5%	21%	9%
	3500	55.111	0.898	61.37×	7%	21%	9%
	7000	107.732	1.763	61.11×	12%	21%	9%

7 Discussion

Before releasing the new VR application to the market, the developing team will perform extensive tests for their product on mainstream mobile devices, and the offline TBA algorithm of the proposed ERS method is designed for this stage. Given a VR application scene as the input, the offline TBA algorithm can automatically output the threshold table specific for current mobile device without human participation, then store it. When user purchases this mobile device and starts to run the VR application, the threshold table will be fetched by the online CORSA algorithm to guarantee the performance of the AFS mechanism. However, the running time of the offline TBA algorithm is still long in some cases, especially when adopting a large grid partitioning density m or running on a low processing capacity mobile device, which may limit the scalability of the ERS method. For example, for Huawei Nova4 tested in this paper, running TBA algorithm on the Viking Village VR application with $m = 24$ will take around 3 hours.

As indicated in Algorithm 1, the major time-consuming operation in the offline TBA algorithm is the repeated 10 times rendering for a single radius, whose purpose is to judge whether the rendering workload under current radius is 'affordable' or 'not affordable' for the tested mobile device. Therefore, to reduce the time consumption of the offline TBA algorithm, a machine learning model which directly performs the judgement above can be trained to gradually replace the repeated rendering operation.

Considering to the difficulty of obtaining large amount of training data and labels, the reinforcement learning adopted in Q-VR [21] is suitable for this binary judgement. In the early stage of the offline TBA algorithm running for the tested VR application and mobile device, if the rendering workload under current radius is judged by the reinforcement learning model as 'affordable', but the subsequent repeated rendering operation behaves in a unexpected frame rate, then a negative feedback will be added to the 'acceptable' judgement for this rendering workload. Then if a similar rendering workload under any radius is countered next time, the reinforcement learning model tends to provide 'not affordable' judgement. Gradually, the reinforcement learning model can provide more accurate judgement for a wide range of rendering workload, then the repeated rendering operations can be disabled to dramatically reduce the time consumption of the offline TBA algorithm, thereby improving the scalability of the proposed ERS method. The detailed design and validation to the idea above is left as the future work.

It should be noted that the application of the machine learning algorithm in the offline TBA algorithm will not affect the high adaptability of the proposed ERS method to the SEV phenomenon, because the SEV phenomenon only appears in the online CORSA algorithm which has high-frequency interaction with users.

8 Conclusion

In this paper, the existence of the SEV and RDL phenomenon that exist in the AFS mechanism are identified using theoretical analysis and experiments in the first time. To adapt to the SEV and RDL phenomenon, a novel, real-time, and effective implementation for the AFS mechanism, namely the ERS method, is proposed for the practical deployment. According to experiments on 4 mobile devices and 3 VR applications, the ERS method can achieve 2.44× to 9.07× higher frame rate precision than the state-of-the-art Q-VR method, and with an ignorable influence of around 1 ms on the rendering pipeline task scheduling under the support of the dedicated hardware accelerator. This performance is achieved by designing the algorithm that successfully adapts to two influence factors in the AFS mechanism. **Firstly, the algorithm does not adopt any online learning operations in the function which has high-frequency**

interaction with users, thus can make real-time reactions to any SEV phenomenon. Secondly, the algorithm controls the rendering workload within the foveated layer just below the rendering workload threshold, therefore will not be affected by the RDL phenomenon.

ACKNOWLEDGMENTS

We would like to thank all reviewers and editors for their insightful comments. This research was supported by Future System Architecture Lab, the University of Sydney.

REFERENCES

[1] W. Zhang, J. Chen, Y. Zhang, and D. Raychaudhuri, "Towards efficient edge cloud augmentation for virtual reality mmogs." New York, NY, USA: Association for Computing Machinery, 2017.

[2] V. R. Gaddam, M. Riegler, R. Eg, C. Griwodz, and P. Halvorsen, "Tiling in interactive panoramic video: Approaches and evaluation," *IEEE Transactions on Multimedia*, vol. 18, no. 9, pp. 1819–1831, 2016.

[3] J. Chakareski, "Vr/ar immersive communication: Caching, edge computing, and transmission trade-offs." New York, NY, USA: Association for Computing Machinery, 2017.

[4] J. Hooft, M. Vega, S. Petrangeli, T. Wauters, and F. Turck, "Tile-based adaptive streaming for virtual reality video," *ACM transactions on multimedia computing communications and applications*, vol. 15, no. 4, pp. 1–24, 2020.

[5] R. Ghaznavi-Youvalari, A. Zare, H. Fang, A. Aminlou, Q. Xie, M. M. Hannuksela, and M. Gabbouj, "Comparison of hevc coding schemes for tile-based viewport-adaptive streaming of omnidirectional video," in *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*, 2017, pp. 1–6.

[6] X. Liu and Y. Deng, "Learning-based prediction, rendering and association optimization for mec-enabled wireless virtual reality (vr) network," 2020.

[7] S. Zhang, M. Tao, and Z. Chen, "Exploiting caching and prediction to promote user experience for a real-time wireless vr service," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[8] M. S. Anwar, J. Wang, S. Ahmad, A. Ullah, W. Khan, and Z. Fei, "Evaluating the factors affecting qoe of 360-degree videos and cybersickness levels predictions in virtual reality," *Electronics (Basel)*, vol. 9, no. 9, pp. 1–, 2020.

[9] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," *IEEE/ACM transactions on networking*, vol. 28, no. 4, pp. 1698–1711, 2020.

[10] P. K. Yadav, A. Shafiei, and W. T. Ooi, "Quetra: A queuing theory approach to dash rate adaptation." New York, NY, USA: Association for Computing Machinery, 2017.

[11] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran, "Probe and adapt: Rate adaptation for http video streaming at scale," *IEEE journal on selected areas in communications*, vol. 32, no. 4, pp. 719–733, 2014.

[12] B. Rainer, S. Lederer, C. Muller, and C. Timmerer, "A seamless web integration of adaptive http streaming," in *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*. IEEE, 2012, pp. 1519–1523.

[13] R. Bhattacharyya, A. Bura, D. Rengarajan, M. Rumuly, S. Shakkottai, D. Kalathil, R. Mok, and A. Dhamdhere, "Qflow: A reinforcement learning approach to high qoe video streaming over wireless networks," in *Proceedings of the Twentieth ACM International Symposium on mobile ad hoc networking and computing*, ser. Mobihoc '19. ACM, 2019, pp. 251–260.

[14] C. Zhou, C.-W. Lin, X. Zhang, and Z. Guo, "Tfdash: A fairness, stability, and efficiency aware rate control approach for multiple clients over dash," *IEEE transactions on circuits and systems for video technology*, vol. 29, no. 1, pp. 198–211, 2019.

[15] Y. Qin, R. Jin, S. Hao, K. R. Pattipati, F. Qian, S. Sen, B. Wang, and C. Yue, "A control theoretic approach to abr video streaming: A fresh look at pid-based rate adaptation," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[16] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, N. Dai, and H.-S. Lee, "Furion: Engineering high-quality immersive virtual reality on today's mobile devices," *IEEE Transactions on Mobile Computing*, vol. 19, no. 7, pp. 1586–1602, 2020.

[17] X. Liu, C. Vlachou, F. Qian, C. Wang, and K.-H. Kim, "Firefly: Untethered multi-user VR for commodity mobile devices," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 943–957.

[18] J. Meng, S. Paul, and Y. C. Hu, *Coterie: Exploiting Frame Similarity to Enable High-Quality Multiplayer VR on Commodity Mobile Devices*. New York, NY, USA: Association for Computing Machinery, 2020.

[19] J. Kim, M. Moroz, B. Arcioni, and S. Palmisano, "Effects of head-display lag on presence in the oculus rift," in *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3281505.3281607

[20] S. Palmisano, R. Mursic, and J. Kim, "Vection and cybersickness generated by head-and-display motion in the oculus rift," *Displays*, vol. 46, pp. 1–8, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141938216300713

[21] C. Xie, X. Li, Y. Hu, H. Peng, M. Taylor, and S. L. Song, "Q-vr: System-level design for future mobile collaborative virtual reality," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 587–599. [Online]. Available: https://doi.org/10.1145/3445814.3446715

[22] B. Guenter, M. Finch, S. Drucker, D. Tan, and J. Snyder, "Foveated 3d graphics," vol. 31, no. 6, 2012.

[23] X. Meng, R. Du, M. Zwicker, and A. Varshney, "Kernel foveated rendering," *Proceedings of the ACM on computer graphics and interactive techniques*, vol. 1, no. 1, pp. 1–20, 2018.

[24] M. Stengel, S. Grogorick, M. Eisemann, and M. Magnor, "Adaptive image-space sampling for gaze-contingent real-time rendering," *Computer graphics forum*, vol. 35, no. 4, pp. 129–139, 2016.

[25] X. Meng, R. Du, J. F. JaJa, and A. Varshney, "3d-kernel foveated rendering for light fields," *IEEE transactions on visualization and computer graphics*, vol. 27, no. 8, pp. 3350–3360, 2021.

[26] P. Lungaro, R. Sjoberg, A. J. F. Valero, A. Mittal, and K. Tollmar, "Gaze-aware streaming solutions for the next generation of mobile vr experiences," *IEEE transactions on visualization and computer graphics*, vol. 24, no. 4, pp. 1535–1544, 2018.

[27] A. N. Angelopoulos, J. N. Martel, A. P. Kohli, J. Conradt, and G. Wetzstein, "Event-based near-eye gaze tracking beyond 10,000 hz," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 5, pp. 2577–2586, 2021.

[28] B. Mochocki, K. Lahiri, S. Cadambi, and X. Hu, "Signature-based workload estimation for mobile 3d graphics," in *2006 43rd ACM/IEEE Design Automation Conference*, 2006, pp. 592–597.

[29] M. Wimmer and P. Wonka, "Rendering time estimation for real-time rendering," in *Proceedings of the 14th Eurographics Workshop on Rendering*, ser. EGRW '03. Goslar, DEU: Eurographics Association, 2003, p. 118–129.

[30] J. Song, X. Li, B. Sun, Z. Cheng, C. Wang, and X. Zhou, "Fcm: Towards fine-grained gpu power management for closed source mobile games," in *2016 International Great Lakes Symposium on VLSI (GLSVLSI)*, 2016, pp. 353–356.

[31] D.-J. Zhang-Jian, C.-N. Lee, I.-J. Huang, and S.-R. Kuang, "Power estimation for interactive 3d game using an efficient hierarchical-based frame workload prediction," in *Proceedings : APSIPA ASC 2009 : Asia-Pacific Signal and Information Processing Association, 2009 Annual Summit and Conference*. Asia-Pacific Signal and Information Processing Association, 2009 Annual Summit and Conference, International Organizing Committee, oct 2009, pp. 208–215. [Online]. Available: http://hdl.handle.net/2115/39675

[32] Q. Li, W. Wu, L. Xu, J. Huang, and M. Feng, "A statistics based prediction method for rendering application," in *Network and Parallel Computing*, G. R. Gao, D. Qian, X. Gao, B. Chapman, and W. Chen, Eds. Cham: Springer International Publishing, 2016, pp. 73–84.

[33] Z. Cheng, X. Li, B. Sun, J. Song, C. Wang, and X. Zhou, "Behavior-aware integrated cpu-gpu power management for mobile games," in *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2016, pp. 439–444.

[34] U. Gupta, J. Campbell, U. Y. Ogras, R. Ayoub, M. Kishinevsky, F. Paterna, and S. Gumussoy, "Adaptive performance prediction for integrated gpus," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–8.
[35] P. Mercati, R. Ayoub, M. Kishinevsky, E. Samson, M. Beuchat, F. Paterna, and T. Rosing, "Multi-variable dynamic power management for the gpu subsystem," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.
[36] Y. Choi, S. Park, and H. Cha, "Graphics-aware power governing for mobile devices," in *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 469–481. [Online]. Available: https://doi.org/10.1145/3307334.3326075
[37] X. Li and G. Li, "An adaptive cpu-gpu governing framework for mobile games on big.little architectures," *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1472–1483, 2021.
[38] Unity Asset Store, "Viking village," Mar. 23, 2021 [Online]. [Online]. Available: https://assetstore.unity.com/packages/essentials/tutorial-projects/viking-village-urp-29140
[39] Qualcomm (developer network), "Snapdragon profiler," 2021 [Online]. [Online]. Available: https://developer.qualcomm.com/software/snapdragon-profiler
[40] Unity Technologies, "Unity 2020.2.7," 2021 [Online]. [Online]. Available: https://unity.cn/releases/full/2020
[41] ——, "Monobehaviour," 2022 [Online]. [Online]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html
[42] ——, "Unityengine," Jan. 22, 2022 [Online]. [Online]. Available: https://docs.unity3d.com/ScriptReference/
[43] Microsoft, ".net standard," Jan. 26, 2022 [Online]. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/net-standard?tabs=net-standard-1-0
[44] Xilinx, "Vivado design suite hlx editions 2017.4," Dec. 20, 2017 [Online]. [Online]. Available: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html
[45] Unity Asset Store, "Nature," Jan. 15, 2016 [Online]. [Online]. Available: https://assetstore.unity.com/packages/3d/environments/nature-starter-kit-2-52977
[46] ——, "Office," Jun. 3, 2021 [Online]. [Online]. Available: https://assetstore.unity.com/packages/3d/environments/sci-fi/free-sci-fi-office-pack-195067
[47] Alinx, "Ax7z035," Jun. 9, 2018 [Online]. [Online]. Available: http://www.alinx.com/index.php/default/content/123.html

Yufei Yang received his M.Sc. degree from the University of Sydney, Australia, in 2022. Now he is a Ph.D. student at Harbin Institute of Technology, China. His research interests include computer graphics, computer vision, and computer architecture.

Chenhao Xie received his M.S degree in electrical engineering from Washington University in St.louis, Saint Louis, MO, USA in 2013 and Ph.D degree in the University of Houston, Houston, TX, USA in 2018. Currently, he is an Associate Professor at School of Computer Science and Engineering, Beihang University. His research interests include computer architecture, computer graphics, high performance computing, energy-efficient computing on mobile devices, memory systems for heterogeneous processor.

Liansheng Liu received the B.Sc., M.Sc. and Ph.D. degrees all from Harbin Institute of Technology (HIT), in 2006, 2008, and 2017, respectively, where he is currently an Associate Professor at HIT. From 2012 to 2014, he was with McGill University as a Visiting Ph.D. Student supported by the China Scholarship Council. His research interests include energy-efficient computing, sensor data fusion, and cyber-physical systems. His research aims to develop advanced approaches for industrial application.

Philip H. W. Leong (SM'02) received the B.Sc., B.E., and Ph.D. degrees from The University of Sydney, Sydney, NSW, Australia, in 1987, 1989, and 1993, respectively. In 1993, he was a Consultant at ST Microelectronics, Milan, Italy. From 1997 to 2009, he was with The Chinese University of Hong Kong, Hong Kong. He is currently a Professor of Computer Systems at the School of Electrical and Information Engineering, The University of Sydney, a Visiting Professor at Imperial College London, London, U.K., a Visiting Professor at the Harbin Institute of Technology, Harbin, China, a Chief Technology Advisor at ClusterTech, Hong Kong, and a Visiting Scholar at Xilinx Research Labs, San Jose, CA, USA.

Shuaiwen Leon Song is currently a senior lecturer (associate professor in NA) at the School of Computer Science, University of Sydney, Australia and an affiliated professor at University of Washington, Seattle's EE Department, Seattle, Washington. Prior to his tenure at University of Sydney, Australia, he worked for U.S. Department of Energy Lab as a senior research scientist and technical lead. His research interests include holistic system design, software-hardware co-design, future system design exploration and high performance computing. His most recent works target future accelerator-driven system design for AI and planet-scale virtual reality. He is a Lawrence scholar, Paul Torgersen fellow, a recipient of 2021 Facebook Faculty Research Award, 2020 Australia's most innovative Engineers, IEEE Early Career Award in HPC, DOE Research Highlight and Pathway to Excellence Research Award. He broadly publishes in major computer architecture and HPC conferences, including ISCA, HPCA, MICRO, ASPLOS, and Supercomputing. His past works received a 2017 HiPEAC paper award and two Supercomputing best paper runner-ups.